Esta tesis doctoral contiene un índice que enlaza a cada uno de los capítulos de la misma.
Existen asimismo botones de retorno al índice al principio y final de cada uno de los capítulos.

Ir directamente al índice

Para una correcta visualización del texto es necesaria la versión de Adobe Acrobat Reader 7.0 o posteriores

Aquesta tesi doctoral conté un índex que enllaça a cadascun dels capítols. Existeixen així mateix botons de retorn a l'index al principi i final de cadascun deis capítols .

Anar directament a l'index

Per a una correcta visualització del text és necessària la versió d' Adobe Acrobat Reader 7.0 o posteriors.

Tesis Doctoral

WebSA: Un Método de Desarrollo Dirigido por Modelos de Arquitectura para Aplicaciones Web

Autor: Santiago Meliá
Director: Dr. Jaime Gómez

Volver al índice/Tornar a l'índex

Resumen

En los últimos años Internet ha evolucionado considerablemente, y las aplicaciones Web son cada vez más complejas en términos de contenido, estructura, comportamiento e interfaz. Además, la exigencia de una creciente competitividad del mercado requiere de una puesta en marcha muy rápida de estas aplicaciones. A pesar de ello, todavía en muchas ocasiones las aplicaciones Web se elaboran de una forma artesanal, reduciendo las posibilidades de éxito en su desarrollo y posterior mantenimiento.

En este sentido, surge dentro de la Ingeniería del Software una disciplina denominada Ingeniería Web, centrada en el desarrollo y estudio de las particularidades que presenta la familia de aplicaciones Web. Son muchas las propuestas definidas dentro de esta disciplina que son aplicadas con éxito sobre casos reales y que se centran principalmente en capturar los aspectos funcionales, es decir, contenido, navegación y presentación. Sin embargo, debido a la juventud de estas propuestas, existen ciertas carencias o aspectos que todavía quedan por cubrir: (1) la ausencia en la consideración de los aspectos arquitectónicos que permitan capturar aspectos como la distribución de los componentes, la escalabilidad del sistema, el mantenimiento, la conectividad con sistemas legados, etc. (2) Falta de trazabilidad desde los modelos de los métodos funcionales hasta la implementación. (3) La existencia de múltiples notaciones para representar los mismos conceptos funcionales en las diferentes metodologías.

Para resolver estas carencias, el presente trabajo de tesis define un proceso de desarrollo específico para las aplicaciones Web que destaca por la inclusión de artefactos de arquitectura y la introducción de mecanismos de automatización para acelerar la puesta en el mercado de las aplicaciones Web. Para ello, se define un conjunto de modelos de Arquitectura del Software que complementan los modelos funcionales provenientes de las metodologías de la Ingeniería Web, proporcionando así una especificación más completa de la aplicación. A partir de estos modelos de análisis se inicia un proceso automatizado y trazable mediante un conjunto de transformaciones modelo a modelo que realizan la integración en un modelo de diseño que contiene los aspectos arquitectónicos junto a los aspectos funcionales. En el último paso se define un conjunto de transformaciones modelo a texto que convierte el modelo de diseño integrado en las diferentes implementaciones correspondientes a las distintas plataformas. Para dar soporte a esta propuesta, se implementa una herramienta Web llamada WebTE que permite la definición de los distintos artefactos del proceso mediante el uso de estándares, obteniendo así interoperabilidad con otras herramientas.

Abstract

In the last few years, Internet has shown a continuous evolution and Web applications have become increasingly complex regarding content, structure, behavior and interface. Besides, the growing market competitiveness poses a shorter time-to-market demand for this kind of applications. In spite of this, too often Web applications are developed manually and this fact reduces their possibilities of being successfully developed and maintained.

In order to face this situation, a new discipline, called Web Engineering, has appeared as a new branch of Software Engineering. This discipline is focused on the development and study of common characteristics of Web applications. Many Web Engineering approaches have been applied successfully to the capture of the functional aspects of real cases, such as content, navigation and presentation. However, due to the novelty of these proposals, there are some aspects which have not been tackled yet: (1) the architectural aspects which would permit to obtain some quality attributes such as distributed computation, scalability, maintenance, connectivity with legacy systems, etc.,(2) the traceability from the design models of Web methods to the implementation and (3) the homogenization of the notations and semantics proposed by different Web methods when they aim at representing the same functional concepts.

To overcome these problems, this PhD Thesis defines a specific development process for Web Applications which regards software architecture artefacts as first-level citizens and introduces automation mechanisms that accelerate this process. To do so, this approach defines a set of architectural models which complements the functional models of other Web methodologies providing a complete specification of the application. Together with these models, this approach proposes an automatic and traceable process through a set of model-to-model transformations which carries out the integration of the architectural and functional aspects in a design model. Additionally, the process includes a set of model-to-text transformations which converts the integrated design model into different platform implementations. To give support to this approach and to increase the interoperability with other tools, we have used a Web Tool called WebTE which allows us to define different artefacts of this process using standard languages to represent both the models and the transformations.

Volver al índice/Tornar a l'índex

Agradecimientos

Quiero agradecer primero a mi mujer todo su amor y apoyo, porque sin ninguna duda es y será la principal fuente de motivación mi trabajo. Y a mi niña pequeña Celeste, que acaba de nacer, porque ha dado un nuevo sentido a nuestras vidas.

A mi familia, que me ha acompañado en esta época de cambios tan importantes, su amor y comprensión han sido imprescindibles para que todo saliera bien.

A Jaime Gómez, por el apoyo y confianza que siempre ha tenido en mi persona y mi trabajo. Le estaré siempre agradecido por darme la posibilidad de dar el salto del entorno empresarial al entorno universitario.

A Nora Koch que desde mi estancia en Alemania hasta hoy me ha aconsejado, ayudado y conducido por los senderos de la calidad y del trabajo bien hecho.

A Jose Luis Serrano que con su gran capacidad de aprendizaje y su inestimable trabajo como programador ha hecho posible materializar las ideas planteadas en una excelente herramienta.

A Cristina Cachero, amiga y compañera, desde el principio estuvo animándome cuando todavía mi tesis eran simples elucubraciones y confiando siempre en mis ideas.

A Antonio Vallecillo cuyos agudos comentarios han hecho mejorar sustancialmente la calidad del trabajo.

A Andrea Kraus que durante mi estancia en Munich mostró su hospitalidad y sus brillantes ideas sobre transformaciones.

A mis compañeros de despacho Rafa, Sonia e Irene que me han soportado los cambios de humor y el duro camino que significa escribir una tesis en un despacho tan reducido :).

A Fernando Llopis agradecerle su ayuda en mi faceta como docente y su confianza en la nueva aventura del Master profesional.

A mis compañeros del grupo IWAD Sergio, Norberto, Jesús, Jose y Juan Carlos para que en el futuro continuemos con este buen rollo y saquemos por fin la canción de grupo ;).

A mis amigos y excompañeros de la empresa Quino, Israel, Javi Java, Pedro, Guille, Tito, etc. que me enseñaron lo que significa la palabra compañerismo.

Finalmente, quiero dar mi más sincero agradecimiento a todos mis amigos que me han apoyado antes y durante la realización de esta tesis. Sin su apoyo nunca hubiera realizado este trabajo.

Santi

Sumario del Índice de Contenidos

Resumen ... 3
Abstract ... 4
Agradecimientos ... 5
Sumario del Índice de Contenidos ... 7
Índice de Contenidos ... 8
Índice de Figuras ... 15
PARTE I: Los Aspectos Preliminares de WebSA ... 19
CAPÍTULO 1: Introducción ... 21
CAPÍTULO 2: Estado de la Investigación Actual ... 31
CAPÍTULO 3: Fundamentos de WebSA ... 49
PARTE II: Proceso y Modelado Dirigido por la Arquitectura del Software para Web ... 83
CAPÍTULO 4: El Proceso de Desarrollo de WebSA ... 85
CAPÍTULO 5: Modelo de Subsistemas ... 111
CAPÍTULO 6: Modelo de Configuración de Componentes Web ... 133
CAPÍTULO 7: Modelo de Integración ... 165
PARTE III: Modelado e Implementación de las Transformaciones ... 185
CAPÍTULO 8: Modelado de Transformaciones en WebSA ... 187
CAPÍTULO 9: Una Herramienta MDA para Aplicaciones Web: WebTE ... 223
PARTE IV: Conclusiones y Apéndices ... 245
CAPÍTULO 10: Conclusiones y Trabajos Futuros ... 247
APÉNDICE I: Perfil UML 2.0 del Modelo de Subsistemas ... 257
APÉNDICE II: Topología de Componentes Web ... 263
APÉNDICE III: Perfil UML 2.0 del Modelo de Configuración ... 271
APÉNDICE IV: Los Estereotipos del Lenguaje de Transformaciones UPT ... 287
APÉNDICE V: Relaciones de la Transformación T1 en UPT ... 295
Referencias ... 315

Índice de Contenidos

Resumen 3
Abstract 4
Agradecimientos 5
Sumario del Índice de Contenidos 7
Índice de Contenidos 8
Índice de Figuras 15
PARTE I: Los Aspectos Preliminares de WebSA 19
CAPÍTULO 1: Introducción 21
1.1 Los Problemas Actuales en el Desarrollo de las Aplicaciones Web 21
1.1.1 Ausencia de la Arquitectura del Software 22
1.1.2 Falta de Trazabilidad entre las Fases de Desarrollo 22
1.1.3 Diferentes Notaciones Funcionales para Aplicaciones Web 23
1.2 Objetivos de la Tesis 24
1.2.1 Definición de la Arquitectura del Software para Aplicaciones Web 24
1.2.2 Ingeniería Dirigida por Modelos específica para Aplicaciones Web 24
1.2.3 Utilización de Diferentes Modelos Funcionales Web 25
1.2.4 Implementación de una Herramienta Dirigida Por Modelos 25
1.3 Estructura de la tesis 26
CAPÍTULO 2: Estado de la Investigación Actual 31
2.1 Motivación 31
2.2 Propuestas Funcionales para el desarrollo de Aplicaciones Web 32
2.2.1 OOHDM 32
2.2.2 WebML 33
2.2.3 UWE 33
2.2.4 W2000 34
2.2.5 WSDM 34
2.2.6 OOH 35
2.2.7 Comparativa de las Propuestas Funcionales 35
2.3 Propuestas de Arquitectura del Software para Aplicaciones Web 37
2.3.1 REST 37
2.3.2 Architecture Recovery of Web Applications 37
2.3.3 WAE: Web Application Extension of UML 38
2.3.4 WebArquitect 39
2.3.5 WAM (WebComposition Architecture Model) 39

2.3.6 OOHDM-Java 2 40
2.3.7 Comparativa de las propuestas de Arquitectura para Aplicaciones Web 40
2.4 Propuestas basadas en el Desarrollo Dirigido por Modelos para Aplicaciones Web 41
2.4.1 Model-Driven Development of Large-Scale Web Applications 42
2.4.2 MIDAS 42
2.4.3 Model-Driven Development Process for UWE 43
2.4.4 Consistent and Adaptable W2000 Models 43
2.4.5 Comparativa de las propuestas MDE para el desarrollo Web 44
2.5 Situación de WebSA en la Investigación Actual 46
2.6 Conclusiones del Capítulo 47
CAPÍTULO 3: Fundamentos de WebSA 49
3.1 Visión general de WebSA 49
3.2 Ingeniería Dirigida por Modelos 51
3.2.1 Definición de MDA 53
3.2.2 Conceptos Básicos de MDA 53
3.2.2.1 Modelo 53
3.2.2.2 Metamodelo 55
3.2.2.3 Transformación 57
3.2.2.4 Clasificación de Transformaciones 58
3.2.3 Los Estándares de MDA Utilizados en WebSA 60
3.2.3.1 MOF 61
3.2.3.2 UML 61
3.2.3.3 XMI 63
3.3 La Arquitectura del Software 64
3.3.1 Definición de la Arquitectura del software. 64
3.3.2 Conceptos Principales de la Arquitectura del Software 66
3.3.2.1 Unidad Arquitectónica 66
3.3.2.2 Vista y Perspectiva 66
3.3.2.3 Estilo Arquitectónico 67
3.3.2.4 Abstracción 67
3.3.2.5 Proceso Arquitectónico 68
3.3.3 Mecanismo de Reutilización: Los Patrones 69
3.3.3.1 Definición de Patrón y sus Tipos 69
3.3.3.2 Definición de Patrón de Arquitectura 69
3.3.3.3 Definición de Patrón de Diseño 70
3.3.4 Los Patrones en WebSA 70
3.3.4.1 Patrones de Arquitectura 71
3.3.4.2 Patrones de Diseño 72
3.4 La Ingeniería Web 73
3.4.1 Modelo de Vistas de una Aplicación Web 74
3.4.2 Arquitectura Especifica para las Aplicaciones Web 77
3.4.3 Integración de WebSA con las Propuestas de Diseño Funcional 79
3.5 Conclusiones 80
PARTE II: Proceso y Modelado Dirigido por la Arquitectura del Software para Web 83

CAPÍTULO 4: El Proceso de Desarrollo de WebSA 85
4.1 Motivación 85
4.2 Los Procesos de Desarrollo del Software Actuales 86
4.2.1 Método Ágile 86
4.2.2 Extreme Programming 87
4.2.3 Unified Software Development Process (UP) 87
4.3 Proceso de Desarrollo de WebSA 88
4.3.1 Características Principales 88
4.3.2 Flujos de Trabajo del Proceso de WebSA 89
4.3.3 Actores o Roles del Proceso de WebSA 92
4.3.4 Fases del Proceso de WebSA 93
4.4 Los Requisitos 94
4.4.1 Requisitos Funcionales 95
4.4.2 Requisitos No Funcionales 96
4.5 El Análisis 97
4.5.1 Análisis Funcional Web 97
4.5.2 Modelo de Dominio 98
4.5.2.1 Elementos del modelo 98
4.5.2.2 Metamodelo de dominio 98
4.5.2.3 Ejemplo de Modelo de Dominio (Petstore) 99
4.5.3 Modelo de Navegación (MN) 100
4.5.3.1 Elementos del modelo de navegación 101
4.5.3.2 Metamodelo del modelo de navegación 102
4.5.3.3 Ejemplo de Modelo de Navegación (Petstore) 103
4.5.4 Análisis de Arquitectura Web 105
4.5.5 Modelo de Subsistemas 106
4.5.6 Modelo de Configuración 106
4.6 El Diseño 106
4.6.1 La Transformación T1 107
4.6.2 Modelo de Integración 107
4.7 La Implementación 108
4.7.1 La Transformación T2 108
4.8 Las Pruebas 109
4.9 Conclusiones 109
CAPÍTULO 5: Modelo de Subsistemas 111
5.1 Motivación 111
5.2 El Modelo de Capas Extendido 112
5.3 La Distribución en las Aplicaciones Web 114
5.4 El Modelo de Subsistemas 117
5.4.1 Elementos de Modelado 117
5.4.2 Metamodelo de Subsistemas 118
5.4.2.1 Restricciones para el Subsistema InterfazUsuario 119
5.4.2.2 Restricciones para el Subsistema Presentación 120
5.4.2.3 Restricciones para el Subsistema ControlUsuario 120
5.4.2.4 Restricciones para el Subsistema Servidor 120
5.4.2.5 Restricciones para el Subsistema LogicaNegocio 120
5.4.2.6 Restricciones para el Subsistema Persistencia 121

5.4.2.7 Restricciones para el Subsistema ControlProcesos 121
5.4.2.8 Restricciones para el Subsistema ObjetosNegocio 121
5.4.2.9 Restricciones para el Subsistema AccesoDatos 121
5.4.2.10 Restricciones para el Subsistema DatosFisicos 121
5.5 El Método: Proceso de Descomposición en Subsistemas 122
5.5.1 La División de la Interfaz de Usuario 122
5.5.1.1 Interfaz de Usuario Remoto 122
5.5.1.2 Presentación Distribuida 123
5.5.2 La División del Servidor 123
5.5.2.1 Base de Datos Remota 123
5.5.3 La división de la Lógica de Negocio 124
5.5.3.1 Núcleo de Aplicación Distribuido 124
5.5.4 La División de la Persistencia 125
5.5.4.1 Base de Datos Distribuida 125
5.6 El perfil UML del Modelo de Subsistemas 126
5.7 Ejemplos de Uso: Diferentes Tipos de Subsistemas 127
5.7.1 MS de un Chat Corporativo 127
5.7.2 MS de una Agencia de Viajes Virtual 128
5.7.3 Tienda Electrónica : Petstore 129
5.8 Conclusiones del Capítulo 131
CAPÍTULO 6: Modelo de Configuración de Componentes Web 133
6.1 Motivación 133
6.2 Definición de componente Web 134
6.3 Topología de componentes Web 135
6.4 Modelo de Configuración (MC) 136
6.4.1 Elementos del Modelo de Configuración 137
6.4.1.1 CompWebCM 137
6.4.1.2 AtributosControl 138
6.4.1.3 OperacionWeb 138
6.4.1.4 PuertoWeb 139
6.4.1.5 InterfazWeb 139
6.4.1.6 ConectorWeb 139
6.4.1.7 FinalConectorWeb 140
6.4.1.8 ParteWeb 140
6.4.1.9 PatronWeb 141
6.5 Metamodelo de Configuración 142
6.5.1 Paquete Núcleo 143
6.5.2 Paquete Topología de Componentes 145
6.5.3 Paquete Tipos de Datos 148
6.5.4 Paquete de Vista de Configuración 149
6.6 Método 151
6.7 El Perfil UML del Modelo de Configuración 152
6.8 Aplicación a un caso de estudio 157
6.8.1 MC de la Agencia de Viajes Virtual 157
6.8.2 MC de la Tienda de Animales Electrónica: Petstore 160
6.8.3 MC del Patrón Modelo-Vista-Controlador 2 161
6.8.4 MC del Patron Façade 162

6.9 Conclusiones del Capítulo 163
CAPÍTULO 7: Modelo de Integración 165
7.1 Motivación 165
7.2 Modelo de Integración 167
7.2.1 Metamodelo del Modelo de Integración 168
7.2.2 Elementos de MI 170
7.2.2.1 ModuloWeb 170
7.2.2.2 CompWebIM 170
7.2.2.3 PuertoWebIM 171
7.2.2.4 InterfazWebIM 171
7.2.2.5 ConectorWebIM 172
7.2.2.6 AtributoWebIM 172
7.2.2.7 OperacionWebIM 172
7.2.2.8 ParametroWeb 172
7.2.3 Método del Modelo de Integración 172
7.2.4 El Perfil UML del Modelo de Integración 173
7.2.5 Casos de Estudio 176
7.2.5.1 Modelo de Integración de la Agencia de Viajes Virtual 176
7.2.5.2 Modelo de Integración de la Tienda Virtual Petstore 180
7.3 Conclusiones del Capítulo 184
PARTE III: Modelado e Implementación de las Transformaciones 185
CAPÍTULO 8: Modelado de Transformaciones en WebSA 187
8.1 Motivación 187
8.2 QVT: El Lenguaje Estándar para Transformaciones entre Modelos 189
8.3 UML Para Transformaciones (UPT) 189
8.3.1 Los elementos de UPT 190
8.3.1.1 Transformación 191
8.3.1.2 Relación 191
8.3.1.3 Dominio 192
8.3.1.4 PatrónDominio 193
8.3.2 Metamodelo 195
8.3.3 Definición de los Esterotipos de UPT 198
8.4 Transformación T1: Fusión de los Modelos Funcionales con los Modelos de Arquitectura 199
8.4.1 T1_1: Creación e Indexación de Módulos 201
8.4.2 T1_2: Creación de las Relaciones entre Módulos 202
8.4.3 T1_3: Creación de Componentes Independientes 203
8.4.4 T1_4: Creación de los Componentes Dependientes de Dominio .. 204
8.4.5 T1_5: Creación de los Componentes Dependientes de Navegación 206
8.5 Transformación T2: Integración de los Aspectos Dependientes de Plataforma 208
8.5.1 MOFScript: Lenguaje de Transformaciones Modelo a Texto 210
8.5.2 Estructura Principal de la Transformación T2 211
8.5.3 Reglas de transformación de T2 en MOFScript 214
8.6 Conclusiones 220
CAPÍTULO 9: Una Herramienta MDA para Aplicaciones Web: WebTE 223

9.1 Motivación ... 223
9.2 Descripción de la Herramienta WebTE ... 224
9.2.1 La Arquitectura de WebTE ... 225
9.2.2 La Funcionalidad de WebTE ... 227
9.2.3 La Interfaz de Usuario de WebTE ... 230
9.3 Módulo de Transformaciones UPT ... 237
9.3.1 Estructura Interna del Transformador UPT ... 238
9.3.2 El Proceso de Implementación y Ejecución del Transformador UPT 240
9.4 Módulo de Transformaciones de Modelo a Texto ... 242
9.5 Conclusiones ... 243
PARTE IV: Conclusiones y Apéndices ... 245
CAPÍTULO 10: Conclusiones y Trabajos Futuros ... 247
10.1 Conclusiones Finales ... 247
10.1.1 Características de WebSA ... 248
10.1.2 Aportaciones de WebSA ... 249
10.1.3 Principales Ventajas de WebSA ... 250
10.2 Principales Restricciones y Limitaciones de WebSA ... 251
10.3 Producción Científica ... 252
10.4 Trabajos Futuros ... 253
APÉNDICE I: Perfil UML 2.0 del Modelo de Subsistemas ... 257
APÉNDICE II: Topología de Componentes Web ... 263
A2.1 Tipos de Componentes Comunes ... 263
A2.2 Tipos de Componentes Específicos ... 265
A2.2.1 Interfaz de Usuario ... 266
A2.2.1.1 Subsistema Interfaz Usuario ... 266
A2.2.1.2 Subsistema Presentación ... 266
A2.2.1.3 Subsistema de ControlUsuario ... 267
A2.2.2 Lógica de Negocio ... 268
A2.2.2.1 Subsistema de Procesamiento de Control ... 268
A2.2.2.2 Subsistema de Objetos de negocio ... 269
A2.2.3 Persistencia ... 270
APÉNDICE III: Perfil UML 2.0 del Modelo de Configuración ... 271
A3.1 Perfil de Núcleo de Componentes Común ... 271
A3.2 Perfil Elementos Modelo Configuración ... 276
A3.3 Perfil Componentes Comunes ... 278
A3.4 Perfil Componentes Interfaz Usuario ... 282
A3.5 Perfil Componentes Servidor ... 284
APÉNDICE IV: Los Estereotipos del Lenguaje de Transformaciones UPT ... 287
APÉNDICE V: Relaciones de la Transformación T1 en UPT ... 295
A5.1 T1_1: Creación e Indexación de Módulos ... 295
A5.2 T1_2: Creación de las Relaciones entre Módulos ... 299
A5.3 T1_3: Creación de Componentes Independientes ... 302
A5.4 T1_4: Creación de los Componentes Dependientes de Dominio ... 305
A5.5 T1_5: Creación de los Componentes Dependientes de Navegación ... 309
Referencias ... 315

Índice de Figuras

Figura. 1. Espacio de WebSA en la Investigación Actual 47
Figura. 2. Mapa de Áreas de Conocimiento de WebSA. 50
Figura. 3. Arquitectura de Cuatro Capas de MOF. 56
Figura. 4. El Modelo de Vistas de WebSA. 75
Figura. 5. Flujos de Trabajo y Artefactos del Proceso de Desarrollo de WebSA 91
Figura. 6. Fases del Proceso de Desarrollo de WebSA 93
Figura. 7. Caso de Uso de una Tienda Virtual 96
Figura. 8. Metamodelo de Modelo de Dominio. 99
Figura. 9. Modelo de Dominio de Petstore 100
Figura. 10. Metamodelo del Modelo de Navegación de OOH. 102
Figura. 11. Metamodelo del Modelo de Navegación de UWE. 103
Figura. 12. Modelo de Navegación de OOH para Petstore. 104
Figura. 13. Modelo de Navegación de UWE para Petstore. 105
Figura. 14. Refinamiento del Modelo de Capas 113
Figura. 15. Elementos de Modelado de MS. 118
Figura. 16. Metamodelo de Subsistemas 119
Figura. 17. Vista General de la División en Capas por los Patrones de Distribución 122
Figura. 18. El Perfil de SM. 127
Figura. 19. MS de Chat Corporativo y Agencia Inmobiliaria. 128
Figura. 20. Modelo de Subsistemas de Petstore. 131
Figura. 21. Notación del CompWebCM y sus Propiedades. 138
Figura. 22. Notación del ConectorWeb Directa. 139
Figura. 23. Notación del ConectorWeb mediante Interfaces. 140
Figura. 24. Representación de la ParteWeb B en el CompWebCM A 141
Figura. 25. Notación de la Definición de un PatronWeb 141
Figura. 26. Notación de la Aplicación del PatronWeb. 142
Figura. 27. Paquete Modelos de Componentes 143
Figura. 28. Paquete Núcleo del Metamodelo de WebSA 144
Figura. 29. Paquete de Componentes Comunes 145
Figura. 30. Paquete Componentes de Interfaz de Usuario 146
Figura. 31. Paquete Componentes de Servidor 147
Figura. 32. Paquete Tipos de Datos 149
Figura. 33. Paquete de Vista de Configuración. 151
Figura. 34. Estructura del Perfil MC 153
Figura. 35. Paquete PerfilNucleoComponentes. 154
Figura. 36. Paquete PerfilElementosMC 155

Figura. 37. Paquete de PerfilComponentesComunes 157
Figura. 38. MC de la Agencia de Viajes 159
Figura. 39. MC General de Petstore 161
Figura. 40. PatronWeb Modelo-Vista-Controlador 2 162
Figura. 41. PatronWeb Façade 163
Figura. 42. Metamodelo de MI 169
Figura. 43. Notación del ModuloWeb 170
Figura. 44. Notación del CompWebIM 171
Figura. 45. Estructura de Paquetes del Perfil MI 174
Figura. 46. Paquete PerfilNucleoMI 175
Figura. 47. Definición del Estereotipo EntidadWebIM 176
Figura. 48. ModuloWeb Presentación de la Agencia de Viajes 177
Figura. 49. ModuloWeb ControlUsuario de la Agencia de Viajes 178
Figura. 50. ModuloWeb LogicaNegocio y Persistencia de la Agencia de Viajes 179
Figura. 51. ModuloWeb PresentaciónCliente de PetStore 180
Figura. 52. ModuloWeb ControlUsuarioCliente de Petstore 181
Figura. 53. ModuloWeb LogicaNegocio de Petstore 183
Figura. 54. ModuloWeb PersistenciaPetstore 184
Figura. 55. La Notación de una Relación UPT 192
Figura. 56. Relación SituarOperacion de la Transformación T1 de WebSA 195
Figura. 57. Paquetes Principales del Metamodelo de UPT 196
Figura. 58. Paquete Transformaciones del Metamodelo de UPT 197
Figura. 59. Paquete Relaciones del Metamodelo de UPT 198
Figura. 60. Mapa de Transformaciones de T1 200
Figura. 61. Mapa de Composición de la Transformación T1_1 202
Figura. 62. Mapa de Composición de la Transformación T1_2 203
Figura. 63. Mapa de Composición de T1_3 204
Figura. 64. Mapa de Composición de la Transformación T1_4 205
Figura. 65. Mapa de Composición de la Transformación T1_5 207
Figura. 66. Mapa de Reglas Principales de la Transformación T2 211
Figura. 67. Mapa Simplificado de Reglas de ModuloWeb 212
Figura. 68. Mapa Simplificado de las Reglas de CompWebIM 213
Figura. 69. Regla de Transformación de Inicio de T2 215
Figura. 70. Regla de Texto mapearModuloWeb 215
Figura. 71. Regla de Texto mapearMServidorJ2EE 216
Figura. 72. Regla de Texto mapearMServidorJ2EE 217
Figura. 73. Regla de Texto mapearCompWebIM 218
Figura. 74. Regla de Texto mapearPaginaServidoraJ2EE 219
Figura. 75. Modelo de Subsistemas de WebTE 225
Figura. 76. Modelo de Configuración de WebTE 227
Figura. 77. Modelo de Dominio de WebTE 228
Figura. 78. Modelo de Navegación Principal de WebTE 229
Figura. 79. Modelo de Navegación de las Transformaciones de Texto 230
Figura. 80. Pagina Inicio de WebTE 231
Figura. 81. Página de Aplicaciones Web de WebTE 232
Figura. 82. Pagina Principal de Petstore 232
Figura. 83. Pagina de los Modelos de Petstore 233

Figura. 84. Pagina de las Transformaciones UPT de Petstore 234
Figura. 85. Página de Ejecución de Transformación T1 235
Figura. 86. Ejemplo de Reglas de una Tranformación de Texto T2 236
Figura. 87. Página para la Obtención de la Implementación 237
Figura. 88. Modelo de Estructura Compuesta del Transformador UPT 239
Figura. 89. Modelo de Secuencia del Transformador UPT 240
Figura. 90. El Proceso de Implementación y Ejecución del Transformador UPT 241
Figura. 91. Proceso de Implementación y Ejecución de Transformaciones de Texto 243
Figura. 92. Definición de los Esterotipos de UPT 288
Figura. 93. Relación SubsistemasAModulos 296
Figura. 94. Relación SubsistemaAModulo 297
Figura. 95. Relación InterfazUsuarioAModulo 297
Figura. 96. Relación LocalizarComponentesEnModulos 298
Figura. 97. Relación LocalizarComponenteEnModulo 298
Figura. 98. Relación LocalizarComponenteServidorEnModulo 299
Figura. 99. Relación SituarElementosEnModulos 300
Figura. 100. Relación SituarPuertoWebEnModulo 301
Figura. 101. Relación SituarConectorWebEnModulo 302
Figura. 102. Relación SituarCompWebCMIndyEnModulos 303
Figura. 103. Relación SituarPropiedadesCompWebIMIndy 304
Figura. 104. Relación CrearConectorWeb 305
Figura. 105. Relación SituarCompWebCMDominioEnModulo 306
Figura. 106. Relación SituarPropiedadesCompWebIMDominio 307
Figura. 107. Relación CrearPropiedadesDominioIM 308
Figura. 108. Relación CrearOperacionIMDeOperación 309
Figura. 109. Relación SituarCompWebCMNavEnModulo 310
Figura. 110. Relación SituarPropiedadesCompWebIMNav 311
Figura. 111. Relación CrearOperacionWebDeEnlaceNav 312
Figura. 112. Relación CrearParteWebDeEnlaceNav 313

PARTE I

Los Aspectos Preliminares de WebSA

La parte inicial pretende motivar al lector contextualizando el presente trabajo de tesis dentro de la investigación actual. Para ello, primero se introducen los problemas y los objetivos que se pretenden cubrir. Seguidamente, se hace un repaso sobre el estado del arte en las diferentes disciplinas involucradas y se realiza una comparativa de las diferentes aproximaciones con este trabajo de tesis. Por último, se presentan los conceptos principales de las diferentes disciplinas que intervienen en la definición de la presente tesis ayudando a una mejor comprensión del resto del trabajo.

CAPÍTULO 1

Introducción

"Investigar es ver lo que todo el mundo ha visto, y pensar lo que nadie más ha pensado"

[Albert Szent-Györgi (1893-1986)]

En este capítulo se exponen las razones principales que han animado a la realización del presente trabajo de tesis. Para ello, se explican que problemas existen actualmente dentro del desarrollo de las aplicaciones Web, cuáles son los objetivos que se plantean para resolverlos y cuál es su contenido de la tesis a través de una breve descripción de cada uno de los capítulos y los apéndices.

1.1 Los Problemas Actuales en el Desarrollo de las Aplicaciones Web

La rápida evolución de Internet en los últimos años presenta un mercado cada vez más competitivo donde ha aumentado de forma exponencial el volumen de negocio y el número de usuarios. Esto ha provocado que las aplicaciones Web sean cada vez más complejas y se deban desarrollar en el menor tiempo posible.

En este sentido, las aplicaciones Web presentan aspectos particulares que las distinguen del resto de aplicaciones software. Según Booch [15] las aplicaciones Web presentan como características más comunes (1) un reducido tiempo para su realización y puesta en marcha en el mercado que obliga a los desarrolladores a realizarlas a gran velocidad, (2) un elevado mantenimiento debido a una interfaz de usuario sometido a frecuentes cambios en el aspecto o en la información mostrada. Y (3) la necesidad para su implementación del conocimiento de muchas tecnologías diferentes, lo cual obliga a involucrar a un grupo de desarrolladores en muchas ocasiones interdisciplinario formado por diseñadores gráficos (requieren conocimientos de HTML[1], Javascript, Flash, etc.), expertos en Interfaz Web

[1] HTML: HyperText Markup Language

(conocimientos en JSP[2], ASP[3], etc.), expertos en componentes de servidor (requieren conocimientos de Enterprise Java Beans, COM[4]+, etc.) y finalmente expertos en bases de datos e integración con sistemas legados.

Para cubrir la demanda de un desarrollo específico para las aplicaciones Web, durante los últimos años se ha definido una nueva disciplina dentro de la Ingeniería del Software denominada Ingeniería Web [125]. Dentro la Ingeniería Web se ha propuesto un conjunto de técnicas, lenguajes, métodos y procesos para el desarrollo de las aplicaciones Web. Sin embargo, debido a la juventud de esta disciplina existe todavía un conjunto de problemas o carencias no resueltas que pretenden ser solucionadas por el presente trabajo. A continuación se presenta cada uno de estos problemas:

1.1.1 Ausencia de la Arquitectura del Software

Diferentes aproximaciones de Ingeniería Web como OOH [19], UWE [59], WebML [20], OOHDM [105], WSDM [119] han sido probadas con éxito para la especificación de aspectos funcionales y navegacionales de una aplicación Web, es decir, expresan cómo una aplicación Web almacena, recupera y presenta la información a los usuarios. Sin embargo, la propia idiosincrasia de las aplicaciones Web requiere que se especifiquen no solamente aspectos funcionales sino que también se establezcan restricciones sobre aspectos no funcionales como la distribución de los componentes, la escalabilidad del sistema, el mantenimiento, la conectividad con sistemas legados, etc. Para poder recoger estos aspectos, autores como Bachmann et al. [6] proponen el uso de técnicas bien conocidas de la disciplina de Arquitectura del Software para identificar y formalizar los subsistemas, conectores y componentes que deben constituir la aplicación. Sin embargo, por parte de los métodos de Ingeniería Web la definición de los aspectos de arquitectura ha sido ignorada o pospuesta hasta la implementación con la consiguiente desventaja de no poder establecer de forma explícita cuáles son los componentes adecuados de la arquitectura de una forma independiente de la plataforma destino o poder especificar los artefactos de arquitectura en función de requisitos no funcionales. Existe por tanto un *gap* o salto entre las especificaciones funcionales representadas por los actuales métodos de desarrollo de Ingeniería Web y la implementación final obtenida.

En conclusión, para asegurar la viabilidad de los métodos para el desarrollo de las aplicaciones Web, las diferentes propuestas deben incorporar de forma explícita los aspectos de arquitectura del software dentro de su proceso de desarrollo.

1.1.2 Falta de Trazabilidad entre las Fases de Desarrollo

Una de las principales características de una aplicación Web es su reducido tiempo de realización y puesta en marcha. Por lo tanto, se necesita disponer de un proceso de desarrollo que proporcione una aceleración en cada una de sus fases mediante técnicas de automatización.

En la actualidad existen propuestas en la Ingeniería Web (OOH [19], WebML [20]) que proporcionan mecanismos de automatización para diferentes fases del

[2] JSP: Java Server Pages
[3] ASP: Active Server Pages
[4] COM: Component Object Model

desarrollo a través de técnicas de generación de código. Sin embargo, los módulos de generación de código existentes en estas propuestas definen a nivel de implementación las conversiones directas desde el origen (normalmente un modelo de diseño funcional) al destino (normalmente código final). Esto produce varios problemas. En primer lugar el bajo nivel de abstracción y el enorme tamaño de las transformaciones del generador no permiten establecer una traza entre el origen y el destino, siendo cajas negras para las personas ajenas a la definición de los generadores. Además, los generadores para aplicaciones Web son programas de gran tamaño y muy difícil mantenimiento. Aún más, estos generadores no permiten la introducción de modificaciones dinámicas para aplicaciones específicas que necesitan un tratamiento especial. Por último, los generadores de código de las aproximaciones Web actuales establecen la arquitectura de forma predefinida sin permitir su modificación según las necesidades del cliente.

Además, en un mercado en constante cambio como el de las aplicaciones Web, es necesario evolucionar rápidamente la generación, ya sea introduciendo casos concretos para un determinado dominio o introduciendo nuevas plataformas.

Para solucionar estos problemas, es necesario definir un proceso que establezca de forma precisa cuáles son los pasos de refinamiento o transformaciones desde los modelos definidos en la fase de análisis, pasando por el diseño y llegando a la implementación. Este proceso, de una forma progresiva, debe especificar en cada momento cómo un elemento de una determinada fase se convierte en uno o más elementos en la siguiente fase. Esto permite mantener las transformaciones de forma sencilla y permite su extensión para casos específicos.

1.1.3 Diferentes Notaciones Funcionales para Aplicaciones Web

En los últimos años se han definido un gran número de métodos dentro de la Ingeniería Web, cada uno de las cuáles representa los aspectos funcionales mediante notaciones diferentes. Esta variedad de notaciones ha generado una baja compatibilidad entre cada una de las propuestas que impide que las representaciones de aplicaciones Web realizadas por una propuesta sean reutilizadas por proyectos que trabajan con una propuesta diferente.

Además, algunas aproximaciones como OOH [19], WebML [20], OOHDM [105], etc. utilizan notaciones propietarias que únicamente pueden representarse sobre una sola herramienta, limitando la posibilidad de que sus modelos sean utilizados por un mayor número de desarrolladores. Esta limitación no existe en aproximaciones como UWE [59] cuya notación se basa en perfiles que extienden el estándar de modelado UML [95].

En general, podemos concluir que existe la necesidad de proporcionar mecanismos de estandarización que permitan compartir los modelos propuestos por las diferentes aproximaciones y permitir su uso en un mayor número de herramientas.

Seguidamente, este trabajo de tesis plantea la forma de cubrir cada una de las necesidades detectadas mediante la especificación de objetivos que persigan su solución.

1.2 Objetivos de la Tesis

El presente trabajo de tesis plantea la solución de los problemas planteados previamente, definiendo para ello una nueva aproximación para el desarrollo de las aplicaciones Web denominada WebSA[5], que se centra en los siguientes objetivos:

1.2.1 Definición de la Arquitectura del Software para Aplicaciones Web

Según Grady Booch [15], "*la presencia (o la ausencia) de una arquitectura coherente es un excelente indicativo del éxito (o fracaso) de un proyecto Web*".

Partiendo de esta premisa, este trabajo propone la integración de la arquitectura del software con las diferentes propuestas funcionales dentro de la Ingeniería Web proporcionando un emparejamiento mucho más cercano entre el sistema modelado y la implementación final. Para ello, se define en la fase de análisis una arquitectura del software independiente a la funcionalidad que permita la combinación de diferentes modelos de arquitectura con diferentes modelos funcionales. La arquitectura propuesta es específica para el dominio o familia de aplicaciones Web y mediante la definición de estilos arquitectónicos se establecen topologías de componentes que permiten representar una arquitectura centrada en la solución final Web.

Considerar la arquitectura del software desde las fases iniciales del desarrollo de la aplicación Web permite dirigir su desarrollo y determinar la capacidad para evolucionar el sistema. Además, provee de un mecanismo de reutilización mediante el uso de patrones de arquitectura como respuesta de los requisitos no funcionales definidos por el usuario.

1.2.2 Ingeniería Dirigida por Modelos específica para Aplicaciones Web

En los últimos años se ha definido un nuevo paradigma denominado Ingeniería Dirigida por Modelos (MDE[6]) [55] centrado en la definición del sistema a través del uso intensivo de modelos. MDE, como aspecto realmente novedoso, presenta la posibilidad de dotar de trazabilidad al proceso de refinamiento de un modelo a otro modelo o de un modelo a código mediante un artefacto denominado transformación.

El presente trabajo se centra en la aplicación de MDE sobre el dominio de las aplicaciones Web. Para ello, se apoya en un conjunto de artefactos como son: (1) los modelos definidos por las otras aproximaciones de diseño funcional de la Ingeniería Web como OO-H [19] y UWE [61], (2) los modelos de arquitectura específicos para la Web, (3) las transformaciones que permiten la integración de los aspectos arquitectónicos y funcionales y (4) las transformaciones que permiten la obtención de la implementación final de la aplicación Web.

Con estos artefactos se establece un proceso de desarrollo iterativo definido como una instancia del Proceso Unificado (UP[7]), que presenta como características fundamentales: (1) estar dirigido por la arquitectura, (2) presentar flujos de trabajo (workflows) automatizados mediante el uso de MDE y (3) ser específico para las aplicaciones Web.

---

[5] WebSA: Web Software Architecture (acrónimo inglés de Arquitectura del Software para Web)

[6] MDE: Model-Driven Architecture (acrónimo inglés de Ingeniería Dirigida por Modelos).

El proceso obtenido permite establecer una traza entre cada una de las tareas o flujos de trabajo situados a diferentes niveles de abstracción, es decir, desde el análisis se establece la traza hasta el diseño y del diseño una traza hasta la implementación. Además, la automatización de las tareas ayuda a reducir el tiempo para la obtención de la aplicación Web.

1.2.3 Utilización de Diferentes Modelos Funcionales Web

Las distintas aproximaciones funcionales Web proporcionan no sólo una notación diferente, sino que introducen conceptos que en algunos casos no son equivalentes a los que se pueden encontrarse en las demás. Se considera importante respetar las notaciones para que los desarrolladores utilicen aquellos modelos con los que se sienten más cómodos o tienen más experiencia. Así se obtienen mejores resultados reduciendo el número de errores y aumentado la productividad.

Este trabajo permite la utilización de diferentes modelos funcionales debido a que se realiza una integración con la arquitectura mediante el uso del estándar del paradigma MDE denominado MDA[8] [89]. MDA establece como restricción la definición de los metamodelos mediante MOF[9] y el establecimiento del refinamiento mediante el uso de transformaciones. En nuestra aproximación, las transformaciones obtienen de los modelos funcionales la información que consideran relevante y la insertan en los componentes de arquitectura. Por lo tanto, una vez definidas las transformaciones, esta integración se realiza de una forma automatizada y no se necesita la intervención humana.

La restricción de que los modelos definidos por la aproximación funcional deben estar formalizados mediante un metamodelo MOF [88] no la cumplen todas las aproximaciones. Sin embargo, cada vez más aproximaciones como OOH [19], UWE [59] y WebML [20] definen su metamodelo mediante MOF.

1.2.4 Implementación de una Herramienta Dirigida Por Modelos

Para definir un proceso de desarrollo para aplicaciones Web basado en el paradigma MDE, es necesario contar con las herramientas necesarias para proporcionar la automatización o ejecución de las transformaciones. Por lo tanto, se hace necesario la definición de una herramienta que permita la lectura de los diferentes artefactos definidos a lo largo del proceso, ya sean modelos o transformaciones. A partir de los artefactos introducidos, la herramienta debe obtener el destino de las transformaciones, ya sea/n otro/s modelo/s o la implementación final.

Un estudio de Lewis & Wrage [66] demuestra que las herramientas actuales MDE solamente implementan parte de los conceptos que el estándar define, en concreto se centran principalmente en la generación de código a partir de los modelos. Sin embargo, fallan a la hora de obtener otras ventajas sobre aspectos tan importantes como la interoperabilidad, es decir, usar modelos y transformaciones que puedan ser compartidas con éxito entre las herramientas MDE sin perder información. Para conseguir este objetivo, es necesario que los artefactos utilizados utilicen una notación estándar.

---

[8] MDA: Model-Driven Architecture (acrónimo inglés de Arquitectura dirigida por Modelos). Definida en el capítulo 3.

[9] MOF: MetaObject Facility.

En este sentido, este trabajo plantea la implementación de una herramienta MDE con las siguientes premisas: (1) soporte para la especificación de los modelos y las transformaciones que propone la aproximación Web, (2) extensibilidad para que puedan añadirse y modificarse tanto los modelos como las transformaciones y (3) interoperabilidad entre diferentes herramientas para que puedan definirse los modelos y las transformaciones desde diferentes herramientas. Para ello, la herramienta se define sobre el estándar MDA que se basa en la utilización de los estándares OMG para la definición de sus artefactos.

1.3 Estructura de la tesis

El contenido del presente trabajo pretende cubrir los objetivos planteados en la sección 1.2. La tesis está constituida por 10 capítulos y 2 apéndices que se describen brevemente a continuación:

Capítulo 2: Estado de la Investigación Actual
Este capítulo presenta las diferentes aproximaciones que se pueden encontrar actualmente en el ámbito del desarrollo de aplicaciones Web. Para ello, se realiza una separación de las aproximaciones según los distintos aspectos tratados por la presente tesis. Primero, se revisan aquellas aproximaciones que han tratado la arquitectura del software para las aplicaciones Web. A continuación, se realiza un repaso sobre las diferentes aproximaciones definidas dentro de la Ingeniería Web. Por último, se repasan aquellas aproximaciones que, realizando un desarrollo dirigido por modelos, se han enfocado en las aplicaciones Web.

Capítulo 3: Fundamentos de WebSA
Este capítulo repasa los campos fundamentales en los que se basa nuestra aproximación WebSA y que son necesarios para la compresión del resto de los capítulos. La primera disciplina es la Ingeniería dirigida por modelos y, dentro de ella su estándar MDA, ya que es importante conocer su implicación en el proceso de desarrollo y cuáles son los artefactos que se pueden utilizar. La segunda disciplina es la Arquitectura del Software, ya que es importante describir qué significa y cuáles son sus conceptos principales. Además, se presentan los patrones de arquitectura y diseño como mecanismo de reutilización y conexión entre la arquitectura y los requisitos no funcionales. Por último, el tercer punto es la Ingeniería Web, como disciplina centrada en el desarrollo de las aplicaciones Web. En este punto se especifica el significado de Aplicación Web y cuáles son sus diferentes perspectivas y vistas.

Capítulo 4: Proceso de Desarrollo de WebSA
Este capítulo presenta un proceso de desarrollo que, siendo instancia del Proceso Unificado (UP), se adecua al desarrollo dirigido por modelos y se centra en el dominio de las aplicaciones Web. Para ello, se presentan primero las diferentes aproximaciones de desarrollo del software y se explica por qué UP es la que más se ajusta a nuestro objetivo. Seguidamente se indican cuáles son las principales características, fases, actores y tareas que constituyen el proceso. Por último, se

describen de forma detallada los diferentes flujos de trabajo o tareas del proceso incluyendo los distintos artefactos utilizados en cada uno de ellos.

Capítulo 5: Modelo de Subsistemas
El Modelo de Subsistemas es el primer modelo de arquitectura definido en el análisis arquitectónico. Representa un estilo arquitectónico definido para la representación de los diferentes subsistemas que constituyen la arquitectura de la aplicación Web. Para ello, se definen los diferentes patrones de distribución que establecerán un método de aplicación basado en un conjunto de divisiones del sistema que darán lugar a los diferentes subsistemas. Además, se formalizan sus elementos mediante el metamodelo MOF y se estandarizan mediante un perfil UML. Por último, se presentan dos casos de estudio que se irán mostrando para cada uno de los modelos de WebSA, por un lado, la conocida aplicación de comercio electrónico definida por SUN denominada Petstore [115], que permite la especificación de las mejores prácticas en la plataforma Java, y por otro lado un sistema online de Agencia de Viajes definido como propuesta en el workshop 1er workshop Model-Driven Web Engineering [.

Capítulo 6: Modelo de Configuración
El Modelo de Configuración es el segundo modelo definido para representar la arquitectura en el análisis. Este modelo define un estilo arquitectónico cuyo elemento principal es el componente Web. Para ello, define una topología de 19 componentes localizados en la familia de aplicaciones Web, que permitirán la definición de la arquitectura de forma independiente a la funcionalidad. A continuación se presentan los diferentes elementos del modelo y se formalizan mediante un metamodelo MOF. Seguidamente se procede a definir un perfil del Modelo de Configuración a partir del Modelo de Estructura Compuesta de UML 2.0 [95]. Por último, se muestra la representación de los casos de estudio Petstore y la Agencia de Viajes mediante el Modelo de Configuración.

Capítulo 7: Modelo de Integración
El tercer y último modelo de arquitectura propuesto en este trabajo es el Modelo de Integración. Este modelo, definido en el diseño, representa la estructura completa de la aplicación Web mediante la representación de los componentes Web a los cuáles se les ha insertado la funcionalidad de las aproximaciones Web. Para su descripción, primero se definen sus elementos, a continuación se define su metamodelo y perfil, y por último, se representan los ejemplos Petstore y Agencia de Viajes.

Capítulo 8: Modelado de Transformaciones en WebSA
En este capítulo se completan los artefactos definidos por la presente aproximación con la definición de las transformaciones T1 y T2 que han sido definidas dentro del propio proceso de desarrollo. Además, se presenta dentro de este trabajo un lenguaje para la definición de transformaciones modelo a modelo denominado UPT [76], que permite la representación de transformaciones utilizando una notación basada en UML. Así, la transformación T1 ha sido definida mediante la notación de UPT. La descripción de T1 se realiza a través de sus cinco subtransformaciones, cada una de ellas encargada de una tarea dentro del proceso de conversión del análisis al diseño. A continuación, se describe la transformación T2 que se ha basado en el lenguaje de

transformaciones modelo a texto MOFScript para definir la transformación de diseño a implementación.

Capítulo 9: Una Herramienta MDA para Aplicaciones Web: WebTE
Para dar soporte a este trabajo, es necesaria una implementación que materialice las diferentes ideas plasmadas en los capítulos anteriores. Para ello, se define una herramienta Web llamada WebTE (WebSA Transformation Engine), que consiste principalmente en dos motores de transformaciones, UPT (Perfil UML para Transformaciones) y Velocity, que obtienen mediante XMI los diferentes modelos y transformaciones de entrada y genera los modelos destino en XMI o la implementación en el código especificado.

Capítulo 10: Conclusiones y Trabajos Futuros
Este último capítulo presenta cuáles han sido las principales conclusiones del presente trabajo. Partiendo de ellas, se detecta qué aspectos son mejorables o incompletos y quedan como trabajo futuro de la presente tesis.

Apéndice 1: Perfil UML del Modelo de Subsistemas
Se presenta el perfil UML 2.0 completo del Modelo de Subsistemas. Se definen para ello de forma completa los estereotipos con sus valores definidos y sus restricciones.

Apéndice 2: Topología de Componentes Web
Este apéndice describe una topología de 19 tipos de componentes que han sido identificados dentro de la familia de aplicaciones Web. Estos tipos de componentes se utilizan para la definición de los modelos de Configuración y de Integración.

Apéndice 3: Perfil UML del Modelo de Configuración
Se presenta el perfil UML 2.0 completo del Modelo de Configuración. Se definen para ello de forma completa los estereotipos con sus valores definidos y sus restricciones.

Apéndice 4: El Perfil UML de UPT
Se presenta el perfil UML 2.0 completo del UPT. Se definen para ello de forma completa los estereotipos con sus valores definidos y sus restricciones.

Apéndice 5: Relaciones de la Transformación T1 en UPT
Se presenta la especificación de las 20 relaciones principales de la transformación T1 de WebSA usando la notación UPT.

Según el tipo de lector, la tesis presenta diferentes rutas de lectura entre los capítulos o secciones que le permiten centrase en distintos temas. Son los siguientes:

- Ruta de Arquitectura Web si se recorre el capítulo 3, 5, 6, 7 y los apéndices 1, 2 y 3 el lector se centrará en aquellos artefactos que están dedicados a la representación de la arquitectura del software para Web.
- Ruta de Desarrollo Dirigido por Modelos: se recomienda la lectura de la sección 3.3, el capítulo 4, 8.

- Ruta de las transformaciones: se recomienda los capítulos 8, 9 y el apéndice 5.
- Ruta de Ingeniería Web: se recomienda la lectura de los capítulos 2, 3.4 y 4.5.

Volver al índice/Tornar a l'índex

CAPÍTULO 2

Estado de la Investigación Actual

"Si he conseguido ver más lejos, es porque me he aupado en hombros de gigantes"
[Isaac Newton (1642-1727)]

2.1 Motivación

La investigación dentro de la Ingeniería Web ha evolucionado rápidamente en los últimos años mediante la incorporación de disciplinas y tendencias provenientes de otros campos de la Ingeniería del Software. En este sentido, este trabajo se ha basado en la introducción de la Arquitectura del Software y el desarrollo dirigido por modelos como las principales tendencias que pretenden solucionar algunos de los problemas que atañen al desarrollo de las aplicaciones Web. Para realizar un repaso del estado de la investigación actual se han separado las diferentes aproximaciones según las disciplinas o tendencias tratadas por la presente tesis.

En primer lugar se muestran las aproximaciones funcionales cuyo desarrollo está centrado en la especificación de las diferentes vistas funcionales, es decir, el modelado del dominio, navegación y presentación. Sobre estas aproximaciones se hace una comparativa que permite identificar qué aproximaciones son más adecuadas para representar los aspectos funcionales de una aplicación Web.

Seguidamente, se presentan las aproximaciones que se centran en la Arquitectura del Software para la especificación y/o desarrollo de las aplicaciones Web. Las aproximaciones existentes son muy heterogéneas y enfocan la arquitectura de forma muy diferente. Además, se realiza una comparativa entre las diferentes aproximaciones y WebSA para ir contextualizando las características del presente trabajo.

Por último, se muestran aquellas aproximaciones que se basan en el desarrollo dirigido por modelos para la especificación y desarrollo de las aplicaciones Web. Debido a la juventud de MDE, son muchas las aproximaciones que presentan un estado inicial, carente de herramientas que puedan dar un soporte real a sus propuestas. De nuevo se presenta una comparativa entre este grupo de aproximaciones y WebSA.

2.2 Propuestas Funcionales para el desarrollo de Aplicaciones Web

Esta sección se centra en aquellas propuestas dentro de la Ingeniería Web que se han preocupado principalmente en la representación de las diferentes vistas funcionales definidas por Retschitzegger & Schwinger [101] que establecían en 3 los niveles diferentes dentro de una aplicación Web: contenido o dominio, navegación y presentación. Dentro de estas propuestas existen diferentes generaciones que se han ido definiendo a lo largo de la historia de la Ingeniería Web, tal y como se presenta en Kappel et al. [54]. La primera generación sienta las bases de la Ingeniería Web y propone la separación de la navegación y el contenido o dominio dentro del proceso de desarrollo. En esta fase destacan dos aproximaciones orientadas a datos, HDM [40] y RMM [48], que extienden el modelo Entidad-Relación introduciendo primitivas para la definición de la navegación Web. Además destaca otra aproximación orientada a objetos, EORM [65], que propone el enriquecimiento de un modelo orientado al objeto mediante la representación adicional en forma de objeto de relaciones entre objetos (enlaces).

La segunda generación de propuestas funcionales de Ingeniería Web se define en la segunda mitad de los noventa, refina los primeros modelos e introduce soporte para funcionalidad básica (altas, bajas y modificaciones de datos) y los primeros esbozos de proceso. En esta segunda generación destacan los métodos como WSDM [119] y OOHDM [105].

La tercera generación definida del 2000 al 2003, ha profundizado en el papel del usuario en los métodos y ha avanzado hacia la estandarización de notaciones, procesos y lenguajes de especificación. Destacan propuestas como W2000 [8], WebML [20], UWE [61] y OOH [19].

La última generación definida a partir del 2004 sería la propia aproximación WebSA como muestra la figura 3-17 de Kappel et al. [54]. Para los objetivos del presente trabajo tienen especial interés las aproximaciones que representan los aspectos funcionales más avanzados, siguiendo además una notación estandarizada que permita su integración en nuestra aproximación. A continuación, se describen con mayor profundad dichas propuestas.

2.2.1 OOHDM

OOHDM [104] [105] es posiblemente la propuesta de Ingeniería Web que más ha influido a los diferentes trabajos en los pasados años. Esta propuesta se basa en HDM pero se adapta al paradigma de orientación a objetos.

OOHDM propone tratar los aspectos funcionales (conceptual o dominio, navegacional y presentación) de forma independiente, es decir, mediante la utilización de modelos separados. A partir de OOHDM, muchas propuestas posteriores han seguido esta misma tendencia.

El proceso de OOHDM comienza con la realización del modelo de clases conceptuales que representa la estructura estática del sistema. A continuación, se realiza un modelo de navegación del sistema. Este modelo ofrece una vista del modelo conceptual y expresa cómo se podrá navegar a través de la información representada en el modelo conceptual. En la tercera fase del proceso se realiza el modelo de interfaz abstracta. Este modelo ofrece una vista de cómo se va a presentar la información al usuario. De esta forma, el modelo de navegación es una vista del

conceptual y el de interfaz abstracta es una vista del navegacional. En la última fase, se realiza la implementación de los tres aspectos.

OOHDM ofrece lenguajes de modelado específicos para representar estos modelos. Este rasgo constituye el mayor handicap de esta aproximación en beneficio de otras aproximaciones como UWE [59], que se han basado en los estándares de modelado.

2.2.2 WebML

WebML [20] es una notación para especificar sitios Web complejos en el ámbito conceptual. Permite una descripción de los sitios Web desde los puntos de vista de dominio, navegacional y presentación.

El proceso de desarrollo comienza con el modelado conceptual del sistema, en el que mediante un lenguaje de modelado (WebML no exige ninguno en concreto, aunque recomienda EER o UML) se representa la estructura estática del sistema. Tras ello, se realiza el modelo de hipertexto en el que se describen uno o más hipertextos que pueden ser publicados en el sitio Web. Cada uno de estos hipertextos define una vista del sitio. La descripción de los hipertextos se realiza mediante dos modelos: el modelo de composición, que define las páginas que componen el sistema, y el modelo de navegación, que describe cómo se puede navegar a través de ellas. En el siguiente paso del proceso se describe el modelo de presentación, que define la apariencia física de las páginas. Por último, el modelo de personalización define cómo debe adaptarse el sistema a los diferentes roles de usuario.

WebML proporciona desde sus orígenes una herramienta CAWE WebRatio [126] que permite aplicar las técnicas propuestas y conseguir los resultados automáticamente. Además, está formalizado mediante la definición de un metamodelo MOF y la estandarización de los modelos mediante un perfil UML 2.0. [81], por lo que sus modelos pueden ser representados en cualquier herramienta con soporte UML 2.0.

2.2.3 UWE

La propuesta de Ingeniería Web basada en UML (UWE) [61] [59] es la primera propuesta que unifica una metodología estándar para el proceso de desarrollo de aplicaciones Web con una definición exhaustiva del proceso que debe ser utilizado. Define un proceso de carácter iterativo e incremental que abarca desde la decisión de construir el sistema hasta que la aplicación es entregada. Este proceso incluye tareas y fases que coinciden con el Proceso Unificado (UP) [50].

El proceso de UWE hace uso exclusivo de estándares reconocidos como UML [120] y el lenguaje de especificación de restricciones asociado OCL [93]. Para simplificar la captura de las aplicaciones Web, UWE define un perfil UML que se utiliza para definir los distintos modelos.

El proceso de UWE está dividido en cuatro pasos o actividades: análisis de requisitos (reflejado en un modelo de casos de uso), diseño conceptual (materializado en un modelo de dominio), diseño navegacional (origen de dos modelos: el modelo de espacio de navegación y modelo de estructura de navegación) y diseño de presentación (cuyo flujo se determina mediante modelos estándares de interacción UML).

Para la representación de cada uno de los artefactos o modelos utilizados a lo largo del proceso de UWE se utiliza el paradigma objetual y más concretamente el estándar UML. Además, UWE define un metamodelo MOF para sus modelos que está basado en la extensión ligera del metamodelo de UML.

Para representar los modelos de UWE, al haberse definido su sintaxis y su semántica como un perfil UML, se puede utilizar cualquier herramienta UML. Sin embargo, para validar los modelos y orientar el proceso completo se ha implementado una extensión de la herramienta opensource ArgoUML [2] que se ha denominado ArgoUWE [3].

2.2.4 W2000

W2000 [8] es el sucesor directo de la metodología HDM [40]. Esta propuesta define un marco para el diseño de aplicaciones Web mediante la combinación de elementos de HDM con elementos propuestos en el seno de UML, prestando especial atención al conjunto de dependencias entre las distintas tareas que lo componen. El lenguaje de modelado resultante de esta combinación de HDM y UML recibe el nombre de HDM2000.

El proceso comienza con una fase de análisis de requisitos basado principalmente en los casos de uso. Con el conocimiento adquirido durante la fase de requisitos, se pasa a la fase de diseño hipermedial. En ella, se realizan dos modelos: el modelo conceptual y el modelo navegacional. Para ello, los autores han modificado y extendido algunos modelos de UML como el diagrama de clases o el diagrama de estados. Por último, se pasa a una fase de diseño funcional, en el que se ha adaptado el diagrama de secuencia para expresar la funcionalidad del sistema.

En un reciente trabajo [7] se formalizan los modelos y las dependencias entre ellos mediante un metamodelo MOF, mientras que simultáneamente se propone la definición completa de los modelos mediante un perfil UML.

Por último, existe una herramienta llamada Jweb [13] que inició el soporte para HDM y se ha ido adaptando para W2000.

2.2.5 WSDM

Otra aproximación relevante es el Método de Diseño de Lugares Web (WSDM) [119]. Presenta un método orientado a objetos y se plantea un proceso dirigido por los requisitos de la audiencia.

Para conseguir su objetivo, WSDM propone cinco fases: especificación de la misión de la aplicación (y audiencia objetivo), modelado de la audiencia, diseño conceptual, diseño de implementación e implementación. Durante el modelado de la audiencia se clasifica y se caracteriza a los usuarios. El resultado es un modelo jerarquizado de usuarios con sus características y necesidades.

Durante la fase de diseño conceptual se modela tanto el espacio de información como la funcionalidad y la navegación requerida por cada tipo de usuario. El modelo conceptual final se consigue mediante la integración de los nodos de información y de los nodos funcionales en el modelo de navegación de la aplicación Web.

La fase de diseño de implementación se centra en la estructura y apariencia de las páginas que conforman la aplicación Web. La estructura de página se deriva del modelo de navegación. Además, si la aplicación tiene soporte (total o parcial) de una

Base de Datos (BBDD), durante esta fase se diseña también el esquema lógico de dicha BBDD (posiblemente a partir del modelo de información de negocio).

Por último, en la fase de implementación se materializan los modelos para el entorno de ejecución elegido. WSDM incluye un entorno CAWE para la generación semiautomática de la aplicación.

Este método define su propia notación gráfica para representar sus modelos, así como un conjunto de palabras reservadas para los objetos del modelo navegacional.

2.2.6 OOH

OOH [19] [42] es una aproximación orientada a objetos que captura la semántica necesaria para el modelado eficiente de interfaces de usuario y su implantación en la Web. OOH ofrece un marco metodológico que permite modelar sistemáticamente interfaces adaptativas y adecuadas al conjunto de usuarios del sistema.

Su ciclo de vida comienza con un análisis de requisitos que da paso a la fase de ingeniería. En esta fase de ingeniería se realiza el análisis y el diseño del dominio y de la navegación. Tras esto se pasa a una fase de construcción y adaptación en la que se obtiene, en base a un conjunto de plantillas, el sistema final. La última fase incluye la evaluación de la interfaz por parte del cliente. Todas estas fases se centran en el usuario. OOH se caracteriza por diseñar interfaces en las que se implementa la navegación del sistema adaptado a las necesidades que cada usuario plantea.

En el ciclo de vida de OOH se propone el uso principalmente de 3 modelos. Por un lado, (1) el diagrama de casos de uso (DCU) que permite recoger los requisitos funcionales de la aplicación Web. Por otro lado, (2) el modelo de diseño de dominio que utiliza como notación el diagrama de clases de UML. Por último, (3) el diagrama de acceso navegacional (DAN) definido mediante una notación propietaria. Estos modelos son explicados con mayor profundidad en el capítulo 4. Cabe destacar que los modelos de OOH han sido formalizados mediante la definición de un metamodelo expresado mediante el estándar MOF [88].

OOH es representado mediante una herramienta que soporta su ciclo de vida completo denominada VisualWade [124]. Sin embargo, se ha definido recientemente un perfil UML para el diagrama DAN que permite que todos modelos de OOH sean expresados en cualquier herramienta UML.

2.2.7 Comparativa de las Propuestas Funcionales

En los últimos años se han realizado multitud de comparativas entre las diferentes propuestas de la Ingeniería Web [61], [19]. El propósito que persigue esta comparativa (ver Tabla. 1) es por tanto complementar las anteriores, enfocándose en la compatibilidad existente entre las diferentes aproximaciones funcionales Web y nuestra propuesta WebSA.

El primer aspecto a analizar es la representación de los diferentes niveles funcionales identificados por [101], que han sido capturados por la totalidad de las aproximaciones. Además, se ha incorporado la representación de los requisitos como tareas que suele aparecen en la mayoría de los procesos de desarrollo. De las 6 aproximaciones, únicamente 3 de ellas UWE, OOH y W2000 representan los requisitos funcionales utilizando Casos de Uso. El resto directamente comienza el proceso a partir de la representación de las entidades del dominio.

Como se ha podido apreciar en la descripción de cada una de las aproximaciones, en los últimos años se ha producido una evolución hacia la estandarización de los distintos métodos con el objetivo de permitir que los modelos sean utilizados por un mayor número de personas, obteniendo así un mayor impacto en la comunidad científica. De las 6 aproximaciones solamente 2 aproximaciones OOHDM y WSDL no utilizan una notación estándar. Las otras 4 aproximaciones ya se basan en UML para representar los diferentes artefactos de sus procesos. De los métodos estandarizados, el pionero en utilizar una notación estándar desde su definición fue UWE [61], que optó por definir sus modelos mediante UML 1.3 [120]. W2000 ha adaptado la notación de HDM [40] a UML. OOH propone desde sus orígenes el uso de UML para todos sus modelos excepto el de navegación (DAN), que ha sido adaptado a UML más recientemente. WebML [81] ha sido la última de este grupo de metodologías que ha estandarizado sus modelos para representarlos mediante perfiles UML, aunque su entorno de desarrollo, muy implantado en la industria, sigue soportando la notación nativa.

Simultáneamente, el otro paso necesario para la estandarización es la formalización de los lenguajes de modelado definidos por las aproximaciones mediante metamodelos MOF [88]. De nuevo, UWE ha sido la pionera en realizarlo en el 2003[62]. UWE propone un metamodelo MOF realizando una extensión ligera del metamodelo UML. A continuación, fue OOH quien definió su metamodelo en MOF, y tras él fueron W2000 y finalmente WebML.

La última característica muestra la compatibilidad de las distintas aproximaciones hipermediales con WebSA. UWE, OOH, WebML y W2000 son compatibles para la definición de la parte funcional de WebSA. Sin embargo, tanto OOHDM como WSDM no cumplen los requisitos de estandarización necesarios para que puedan ser integradas dentro del proceso de WebSA.

Características	OOHDM	UWE	WebML	W2000	WSDM	OOH
Notación Requisitos	No tiene	UML	No tiene	UML	No tiene	UML
Notación Dominio	Propia	UML	Propia y UML	UML	Propia	UML
Notación Navegación	Propia	UML	Propia y UML	UML	Propia	Propia y UML
Notación Presentación	Propia	UML	Propia y UML	UML	Propia	XML
Metamodelo	No	MOF	MOF	MOF	BNF	MOF
Soporte CAWE	OOHDM-Web	ArgoUWE y UML	WebRatio y UML	Jweb	WSDM Prototipe Tool	VisualWade
Compatible WebSA	No	OK	OK	OK	No	OK

Tabla 1. Comparativa de las Propuestas Funcionales para Aplicaciones Web

2.3 Propuestas de Arquitectura del Software para Aplicaciones Web

Otro grupo de trabajos a considerar son aquellos que se han centrado en la definición de la arquitectura del software de las aplicaciones Web. A diferencia de lo que ocurre en la parte funcional de la aplicación Web, no existe una unificación de criterios en cuanto a qué aspectos es necesario capturar en la arquitectura Web. Por lo tanto, es importante mostrar los diferentes tipos de aproximaciones existentes, cuáles son sus principales características comunes y qué aspectos comparten con WebSA. De esta manera comienzan a identificarse las características que distinguen a WebSA del resto de aproximaciones que representan la arquitectura de las aplicaciones Web.

2.3.1 REST

El Representational State Transfer (REST) [31] consiste en un estilo arquitectónico para aplicaciones Web distribuidas. REST está basado en la definición de un conjunto de restricciones arquitectónicas centradas en el dominio de las aplicaciones Web. REST propone un proceso de definición de su estilo arquitectónico apoyándose en la introducción de restricciones sobre otros estilos arquitectónicos reconocidos dentro de la arquitectura del software como Client/Server, Layered, Cache, etc.

REST hace uso de tres vistas arquitectónicas (Proceso, Conector y Datos) para especificar la arquitectura Web. En su especificación ignora los detalles de la implementación del componente y la sintaxis del protocolo para enfocarse en los roles de los componentes, las restricciones de las interacciones con otros componentes y su interpretación de los elementos de datos significativos. Las restricciones de REST se fundamentan en los componentes, conectores y los datos que definen la base de la arquitectura Web.

El objetivo de REST es ser una guía para que los diseñadores de aplicaciones Web definan el conjunto de requisitos no funcionales relevantes para la aplicación. Ejemplos de este tipo de requisitos son la obtención de una buena escalabilidad, despliegue independiente, reducción de la latencia de interacción entre los componentes, refuerzo de la seguridad o la encapsulación de los sistemas legados.

La sintaxis utilizada para modelar la arquitectura no se basa en el uso de estándares de modelado, ni propone su formalización mediante metamodelos. Esto hace difícil su especificación mediante herramientas comerciales.

La clasificación de tipos de componentes realizada por el estilo arquitectónico REST ha servido de referencia para la definición de la tipología de componentes definida por los estilos de WebSA.

2.3.2 Architecture Recovery of Web Applications

Hassan y Holt [42] consideran que las aplicaciones Web se convertirán en los próximos años en sistemas legados de difícil mantenimiento. Partiendo de esta base, proponen una aproximación que realice una recuperación de la arquitectura de las aplicaciones Web a partir de su implementación. El objetivo de esta ingeniería inversa

de la arquitectura es mejorar el mantenimiento de la aplicación, al hacerla más entendible para los desarrolladores.

La aproximación define un conjunto especializado de parsers/extractores que analizan el código fuente y binario de las aplicaciones Web existentes y obtienen un conjunto de diagramas de arquitectura situados a diferentes niveles de abstracción.

La aproximación establece la definición de un conjunto de cinco componentes identificados como comunes dentro de la arquitectura Web: Static Pages (Paginas Estáticas), Active Pages (Paginas activas de servidor), Web Objects (componentes de servidor), Multimedia Objects (objetos multimedia como imágenes, video y sonido) y Databases (bases de datos).

La representación de la arquitectura se basa en un conjunto de tres modelos que de forma progresiva reducen los detalles obtenidos en la recuperación de datos, para mostrar únicamente la información relevante para la arquitectura. El primer modelo es ELS (Entity-Level Schema). Este modelo es el nivel de abstracción más bajo y muestra las relaciones entre los elementos que viven dentro de los componentes Web (objetos, tablas, variables, etc.). A partir de este modelo se definen las transformaciones necesarias para subir el nivel de abstracción hasta el modelo CLS (Component-Level Schema). CLS representa las relaciones entre los componentes de la aplicación Web (StaticPages, ActivePages, Databases, etc.). El último nivel de abstracción es el modelo ALS (Abstract-Level Schema) que representa las relaciones entre los elementos de mayor granularidad, los subsistemas y los componentes que contienen.

Los diferentes modelos representados en esta aproximación se basan en esquemas (Schemas) que son básicamente modelos Entidad-Relación (EER).

2.3.3 WAE: Web Application Extension of UML

La propuesta WAE [24] (Extensión para las aplicaciones Web) definida por Jim Conallen propone un conjunto de modelos UML cercanos a la implementación que, dentro del contexto del Proceso Unificado de Rational (Racional Unified Process [47]), giran en torno a la arquitectura de la aplicación.

Conallen basa la descripción de la arquitectura de las aplicaciones Web en el conocido trabajo de Kruchten *"The 4+1 View Model of Architecture"* [64], estableciendo los artefactos utilizados en cada una de las vistas que define Kruchten para el desarrollo de las aplicaciones Web.

Conallen, partiendo de la idea que una arquitectura nunca aparece de la nada, se basa en un conjunto de patrones de arquitectura para su definición en la fase de diseño. Por un lado, adapta patrones comunes que considera particularmente adecuados para las aplicaciones Web, como son: Façade de Gamma et al. [36], Page Composition y Template Page. Otros patrones específicos para la capa de presentación son Thin Web, Thick Web y Web Delivery.

Sin embargo, donde verdaderamente la arquitectura comienza a tomar relevancia es dentro del proceso definido por Conallen en la fase de diseño. En esta fase define WAE, que incluye un conjunto de tipos de componentes especializados en el dominio de las aplicaciones Web (p.e. Server Page, Client Page, HTML Form, etc.). La definición de cada uno de los componentes la realiza mediante el mecanismo de perfiles proporcionado por UML. WAE permite representar una aplicación Web muy

detalladamente, acercándose al nivel de implementación concreta, con la introducción de aspectos dependientes de ASP y JSP.

A partir de la representación en WAE, existe un mecanismo de generación automática que permite obtener el esqueleto de los diferentes componentes.

2.3.4 WebArquitect

WebArchitect [112] es otra propuesta que se centra en la arquitectura y las funciones de los lugares Web, más que en la apariencia de cada página. Este método comienza con una actividad de análisis en la que, mediante el modelo Entidad-Relación, se representar el dominio del problema. A continuación, un análisis de escenarios determina cómo los usuarios potenciales interactúan con la aplicación Web para cumplir los objetivos de negocio. A partir de las fases de análisis, la arquitectura de la aplicación es diseñada en la fase de diseño arquitectónico. La arquitectura es representada mediante un modelo denominado RMDMW (Relationship Management Data Model for Web-Based Information Systems) que consiste en una extensión del modelo propuesto por RMM [48], con la introducción de eventos, roles y productos. Mediante RMDMW el diseñador determina la navegación y el modo de mapear[1] la navegación a las diferentes páginas.

El método también define atributos para cada página, que son utilizados para el mantenimiento de la misma. La implementación y mantenimiento de la aplicación resultante es soportada por una herramienta del mismo nombre, que permite a los diseñadores manipular de forma directa meta-enlaces entre páginas organizadas en un árbol jerárquico. Por otro lado, la visualización de las aplicaciones resultantes se realiza mediante un cliente Web denominado Pilot-Boat, que navega y deja que los usuarios colaboren a través de los lugares Web.

2.3.5 WAM (WebComposition Architecture Model)

WAM [70] es una aproximación muy reciente basada en la extensión de la reconocida aproximación WebComposition [35]. WAM introduce una descripción arquitectónica que sirve como mapa para el mantenimiento de las trazas de las interrelaciones entre diferentes aplicaciones Web federadas. Las aplicaciones Web federadas se basan en la idea de las aplicaciones Web que comparten componentes y elementos realizados por múltiples proveedores a lo largo de la Web. Entre los artefactos modelados están los servicios Web, las propias aplicaciones Web y las zonas organizacionales de control que están sujetas a la evolución en el sentido de la aproximación WebComposition.

WAM persigue hacer comprensible mediante los modelos la estructura técnica de la federación a los actores (stakeholders) que intervienen en el desarrollo de la aplicación Web (p.e. arquitectos, desarrolladores y administradores).

WAM se basa en DSLs (Domain-Specific Languages) para representar los diferentes elementos identificados como relevantes en la arquitectura Web. Estos son: service, application, data provider, process unit, invocation y trust relationship. A partir de los elementos del modelo, se establece un mapeo a un lenguaje llamado

[1] Mapear: traducción del termino ingles map o mapping. Concretamente indica la acción de hacer corresponder o convertir la información desde un origen a un destino.

WAM-XML que sirve de base para su tratamiento en herramientas y para dar soporte a los sistemas.

2.3.6 OOHDM-Java 2

OOHDM-Java 2 [51] es un trabajo que propone una línea de producto en J2EE para simplificar el desarrollo de aplicaciones utilizando la conocida aproximación de Ingeniería Web OOHDM [106]. OOHDM-Java2 es una arquitectura que permite desacoplar las decisiones de diseño relacionadas con el modelo de dominio de aquellas relacionadas con la navegación y la arquitectura de interfaz.

OOHDM-Java 2 tiene asociado un framework J2EE que extiende el concepto del patrón de Buchmann et al. [17] Model-View-Controller y realiza una separación de los nodos de navegación de sus interfaces. Así, introduce la idea de objeto de navegación, y reconoce el hecho de que la navegación puede ser dependiente del contexto. La estructura de OOHDM-Java 2 contiene elementos que configuran una arquitectura que persigue las mejores prácticas y mejores resultados de mantenimiento. Este esqueleto (framework) es instanciado para las diferentes aplicaciones definidas en OOHDM, mediante un conjunto de tareas definidas que debe seguir el diseñador para su utilización.

2.3.7 Comparativa de las propuestas de Arquitectura para Aplicaciones Web

A primera vista, las diferentes aproximaciones revisadas tratan la representación y el papel de la arquitectura del software para las aplicaciones Web de forma bastante diferente. Por ello, el objetivo de esta comparativa es encontrar cuáles son los principales puntos en común entre ellas y WebSA, que es la propuesta del presente trabajo de tesis.

La primera característica a revisar es el estilo arquitectónico. Con respecto a él, solamente REST define un estilo arquitectónico que permite la representación de las aplicaciones Web independientemente de la funcionalidad. En este aspecto coincide con WebSA.

La segunda característica es el uso de Patrones de arquitectura o diseño. En este aspecto, como se ha indicado antes, WAE define un conjunto de patrones propios de la arquitectura Web. Por su parte, WebSA permite la representación de los patrones propuestos por WAE y otros patrones de arquitectura y diseño definidos por autores como Buchmann et al. [17], Gamma et al. [36], [118], etc.

La tercera característica es la tipología de componentes propios de la familia de aplicaciones Web. En este aspecto la mayoría de las aproximaciones coinciden en la necesidad de esta tipología, aunque cada una de ellas define sus propios tipos en función de sus necesidades. A destacar son las clasificaciones realizadas por REST y por WAM. WebSA se ha basado en estos trabajos para definir algunos tipos de componentes.

La cuarta categoría es el modelado de la arquitectura. Aquí todas las propuestas, a excepción de REST, proporcionan la posibilidad de modelar la arquitectura mejorando así la comunicación con los desarrolladores. En algunas ocasiones el modelado únicamente sirve de documentación (como en Architecture Recovery y WAM), en otros casos es interpretado por herramientas (como es el caso de WAM) y en otros se realiza un proceso de generación automática a partir de ellos (como es el

caso de WAE). En el caso de WebSA los modelos, al definirse dentro de un paradigma dirigido por modelos, son computables, y sirven para determinar en última instancia la implementación de la aplicación Web.

La quinta característica es la notación estándar. Aquí hay que destacar que únicamente WAE se preocupa de proveer una notación estándar como UML [120] que permita modelar su aproximación en cualquier herramienta comercial con soporte UML. El resto de aproximaciones son dependientes de una herramienta donde pueden ser modelados. En el caso de WebSA, sus modelos se definen mediante perfiles del estándar UML 2.0 [95].

La sexta característica que consideramos en esta comparativa es el papel que la arquitectura tiene dentro del proceso de desarrollo. En este punto existe bastante disparidad de criterios. En REST la arquitectura restringe el diseño que se definirá posteriormente. En Architecture Recovery, la arquitectura se obtiene mediante un proceso de ingeniería inversa donde, a partir del código, se obtiene la arquitectura que servirá de documentación. En WAE la arquitectura dirige el proceso de desarrollo al basarse en RUP [47]. En WebArch la arquitectura se define en la fase de diseño arquitectónico. En WAM la arquitectura se utiliza para definir de manera explícita un aspecto muy concreto: la federación de componentes en el seno de las aplicaciones Web federadas. Esta especialización diferencia a esta propuesta del resto de aproximaciones. Por último, en WebSA la arquitectura dirige el proceso de desarrollo al basarse en el proceso unificado (UP) [50].

Característica	REST	Architecture Recovery	WAE	WebArchitect	WAM	WebSA
Estilo Arquitectónico	Sí	No	No	No	No	Sí
Patrones	No	No	Sí	No	No	Sí
Tipología	Sí	Sí	Sí	No	Sí	Sí
Modelado	No	Sí	Sí	Sí	Sí	Sí
Notación Estándar	No	No	Sí	No	No	Sí
Papel en el Proceso Desarrollo	Guiar el diseño	Doc. Sistemas legados	Dirigir el proceso	Fase de diseño	Dirige el proceso	Dirigir el proceso

Tabla 2. Comparativa de las Propuestas de Arquitectura para Aplicaciones Web

2.4 Propuestas basadas en el Desarrollo Dirigido por Modelos para Aplicaciones Web

El último conjunto de propuestas de interés para este trabajo es aquel que se basa en el paradigma de ingeniería dirigida por modelos (MDE) [55] para el desarrollo de las aplicaciones Web. MDE proporciona como principales ventajas el dotar de un mecanismo de trazabilidad desde los modelos hasta la implementación mediante el uso de las transformaciones. Sin embargo, debido a la juventud de MDE son pocas las herramientas disponibles hasta el momento y la mayoría de las implementaciones realizadas por las aproximaciones no se basan en transformaciones formales, sino que las implementan directamente mediante generación de código. Otro de los aspectos a

considerar es el uso de los modelos definidos en la Ingeniería Web para representar las diferentes vistas: mientras que algunas aproximaciones definen sus propios modelos, otras se valen de trabajos reconocidos para definir sus modelos.

Seguidamente se describen cada una de las propuestas y se realiza una comparativa con WebSA.

2.4.1 Model-Driven Development of Large-Scale Web Applications

Tai et al. [111] definen una aproximación basada en MDA para el desarrollo de aplicaciones Web de gran tamaño. Para ello, proveen un conjunto de modelos basados en un metamodelo que es usado como contrato principal entre los desarrolladores.

El metamodelo juega un papel fundamental al realizar una división en subaplicaciones que provee la especificación y que todos los artefactos deben cumplir. El metamodelo se ha definido conforme a la arquitectura propuesta para la plataforma J2EE. En un futuro se pretende hacer que el metamodelo sea general para otras plataformas.

Los modelos definidos por la aproximación son: (1) el modelo de transición de páginas (navegación entre los nodos mediante un gráfico), (2) modelo de flujo de página (composición de la interfaz de una página) y el (3) modelo de datos (información que se almacena a partir de las páginas). Para la definición de estos modelos, Tai no se ha basado en los modelos definidos dentro de la Ingeniería del Software.

Acompañando a esta aproximación se ha definido una herramienta llamada WAST (Web Application development Support Tool). WAST provee un conjunto variado de generadores de código (básicamente esqueletos) y mecanismos de validación de código para comprobar que los artefactos definidos en las diferentes vistas (p.e. los JSPs (Java Server Pages)) son compatibles con el modelo.

Como parte del proceso de desarrollo se deben definen los diferentes actores (stakeholders), cada uno de los cuáles debe seguir las reglas impuestas por el metamodelo. Estos actores son el diseñador de pantallas, el mantenedor de modelos, el programador de lógica de negocio, el programador de objetos de servidor y el ensamblador de la aplicación. Los programadores deben rellenar aquellos huecos que los generadores de código dejan, siempre verificando que el código introducido respecta el modelo.

2.4.2 MIDAS

MIDAS [18] metodología dirigida por modelos para el desarrollo de aplicaciones Web. Esta metodología aplica un metamodelo MDA a la plataforma Web utilizando XML y tecnología objeto-relacional. MIDAS propone diferentes modelos PIM y PSM y define algunas reglas de mapeo entre los modelos.

Los modelos independientes del plataforma (PIM) propuestos por MIDAS están definidos utilizando el estándar UML [120]. Los modelos PIM están constituidos por contenido, navegación y presentación. Para representar los distintos modelos MIDAS se basan principalmente en UWE [61].

MIDAS también define un conjunto de modelos dependientes de plataforma (PSM) que representan cada una de las vistas definidas como PIM. Así, para representar el modelo de contenido dependiente de plataforma, se ha valido de la tecnología objeto-

relacional. Sin embargo, para representar la navegación y la presentación ha utilizado XML.

MIDAS propones únicamente las guías para realizar las transformaciones PIM-PIM, PIM-PSM y PSM-PSM necesarias para completar su desarrollo. Actualmente, está trabajando en implementar las transformaciones para obtener la aplicación final sobre plataformas como J2EE y .NET.

2.4.3 Model-Driven Development Process for UWE

El reciente trabajo [58] redefine el proceso de UWE estableciéndose como un proceso de desarrollo dirigido por modelos (ahora llamaremos MDD-UWE). Para su definición se ha basado en los principios de MDA, usando los estándares OMG para la definición de sus modelos y transformaciones. El proceso de desarrollo de MDD-UWE consiste en un conjunto de modelos y transformaciones cuya especificación está soportada por metamodelos y reglas de transformación. Los metamodelos son el metamodelo de MDD-UWE [59], el metamodelo Web Requirements Engineering metamodel (WebRE) [30] y además, utiliza el metamodelo de nuestra propia propuesta WebSA (Meliá & Gómez [75]).

El proceso de desarrollo especifica los requisitos funcionales basándose en el trabajo de [29] para su automatización de requisitos a contenido funcional. Los aspectos funcionales son especificados simultáneamente con los modelos de arquitectura especificados por WebSA, y posteriormente son integrados. Existe otra alternativa para la integración de la arquitectura con un modelo "*Big Picture*", es decir, un modelo que integra ya las tres vistas funcionales (presentación, contenido y navegación).

Las transformaciones definidas se basan en tecnologías y lenguajes diferentes, desde el lenguaje estándar de transformaciones QVT [91], ATL [5], transformaciones basadas en grafos, e incluso transformaciones implementadas en código java en la propia herramienta ArgoUWE [3].

2.4.4 Consistent and Adaptable W2000 Models

Esta propuesta ha evolucionado a través de un reciente trabajo [9] que establece el metamodelado de todos los modelos de W2000 mediante MOF. Además, propone la definición de las reglas de transformación que permitan a los usuarios acceder y controlar la consistencia de los artefactos producidos por el proceso W2000 y posteriormente adaptarlos en un modo controlado.

La propuesta no propone un proceso de desarrollo determinado ya que prefiere dotar de libertad a los desarrolladores para que elijan el que prefieran.

La corrección de los modelos definidos por los modeladores es controlada mediante la definición de los modelos como instancias de metamodelos MOF y mediante el establecimiento de las restricciones oportunas mediante el estándar OCL [93].

Por otro lado, se propone la definición de reglas de transformación definidas mediante el lenguaje de grafos AGG (Attributed Graph Grammar System) [28] que representan las reglas de transformación a nivel de metamodelo.

Además, la aproximación tiene el soporte de una herramienta realizada mediante un plug-in[2] de Eclipse [27] para su parte gráfica, un repositorio MOF comercial, MDR [79], que le permite la carga de los modelos mediante XMI y la herramienta proporcionada para el lenguaje AGG [28] para definir las reglas de transformación.

2.4.5 Comparativa de las propuestas MDE para el desarrollo Web

Debido a que la ingeniería dirigida por modelos es muy reciente, dentro de esta disciplina el número de trabajos localizado en el desarrollo de las aplicaciones Web es todavía muy reducido. Sí existen dentro de MDE trabajos genéricos de IS que pueden ser utilizados para desarrollar aplicaciones Web, pero no están adaptados a los modelos específicos que propone la Ingeniería Web. Todos estos trabajos quedan fuera de este estudio.

El objetivo de esta comparativa es mostrar un conjunto de características de las diferentes aproximaciones MDE para Web que se consideran de importancia para demostrar su estado de madurez y valía. Además, estas características son mostradas también para nuestra propuesta WebSA, permitiendo compararla con las demás e ir introduciendo ciertos aspectos.

La primera característica es si las aproximaciones definen un proceso completo de desarrollo para las aplicaciones Web. En este caso, de las cinco aproximaciones solamente W2000 no presenta un proceso completo alegando que únicamente presenta los modelos de diseño ya que debe ser del desarrollador quien elija el proceso que considere oportuno. El resto de aproximaciones con más o menos detalle presentan un proceso de desarrollo dirigido por modelos, y concretamente WebSA y MDD-UWE instancian además el proceso de desarrollo unificado (UP).

El segundo aspecto a tratar, y uno de los que se consideran de mayor importancia, es el uso de las transformaciones para realizar los diferentes mapeos en el proceso. De las aproximaciones presentadas, Tai no define transformaciones puesto que propone el uso de generación de código y MIDAS todavía no ha formalizado su trabajo mediante ellas. MDD-UWE y CA-W2000 han presentado transformaciones de ejemplo pero no se tiene constancia de que estén definidas todas las transformaciones necesarias para completar su trabajo. Como se mostrará en el capítulo 8, WebSA ha definido las transformaciones necesarias en cada uno de los pasos de su proceso de desarrollo.

Tai y MIDAS utilizan la generación de código para realizar sus mapeos. MDD-UWE también conserva algunos mapeos con generación de código dentro de su herramienta ArgoUWE [3]. En el caso de CA-W2000 y WebSA, ambas aproximaciones se basan completamente en las transformaciones.

La siguiente característica es el soporte de una herramienta para materializar su propuesta MDE. Este aspecto es de vital importancia puesto que MDE delega en la automatización muchos de los pasos a realizar. De las aproximaciones presentadas solamente MIDAS no presenta una herramienta que le dé soporte. Por su lado, MDD-UWE ha sido propuesta muy recientemente y ciertos pasos de transformaciones están pendientes de elaboración. Por su parte, Tai, CA-W2000 y WebSA presentan sus propias herramientas.

---

[2] Plug-in: programa que interactura con una aplicación principal para proveerle de cierta función normalmente muy específica.

La quinta característica es la Arquitectura del Software. De las propuestas definidas únicamente MDD-UWE y WebSA presentan la posibilidad de definir los modelos que representen la arquitectura. Precisamente, MDD-UWE se basa en WebSA para la definición de sus modelos de arquitectura.

La sexta característica son los estándares MDA definidos por la OMG que se utilizan dentro de las aproximaciones. Tai únicamente utiliza UML para la definición de sus modelos. MIDAS usa UML e incorpora MOF para definir el metamodelo. A estos dos estándares CA-W2000 incorpora OCL [93] para definir las restricciones del metamodelo. Por su parte, MDD-UWE incorpora además el uso de QVT [91] para definir las transformaciones. Por último, WebSA no utiliza QVT ya que define su propio lenguaje de transformaciones basado en un perfil de UML (UPT) [76], y utiliza otros dos estándares; XMI [97] para el transporte de los modelos y MOFScript [94] que es una propuesta de estándar OMG para la definición de las transformaciones modelo a texto.

Característica	Tai	MIDAS	MDD-UWE	CA-W2000	WebSA
Proceso Completo	Sí	Sí	Sí, basado en UP	No	Sí, basado en UP
Transformaciones	No	No	Sí	Sí	Sí
Generación de código	Sí	Sí	Parcial	No	No, con MOFScript
Soporte herramienta	Sí	No	Parcial	Sí	Sí
Arquitectura	No	No	Sí, modelos de WebSA	No	Sí
Estándares MDA	UML	UML, MOF	UML, MOF, OCL, QVT	UML, MOF, OCL	UML, MOF, OCL, XMI, MOFScript
Procedencia Modelos Funcionales	Propios	UWE	UWE	W2000	Genérico

Tabla 3. Comparativa de las propuestas MDE para Aplicaciones Web

La última característica que analiza la comparativa MDE es la indicación de la procedencia de los modelos funcionales utilizados por las propuestas. Tai propone la utilización de modelos funcionales propios de su propuesta. Por su parte, MIDAS y MDD-UWE utilizan modelos funcionales que se basan en la propuesta de UWE para definir el contenido, navegación y presentación. CA-W2000 también define sus propios modelos funcionales definidos en trabajos previos [8]. Finalmente, WebSA permite la utilización de modelos funcionales de diferentes propuestas que cumplan una determinada restricción, es decir, aquellas propuestas que definan un metamodelo MOF para formalizar sus modelos.

2.5 Situación de WebSA en la Investigación Actual

Realizando un ejercicio de revisión conjunta de las tres tendencias consideradas influyentes para el presente trabajo, es muy interesante destacar cuál ha sido la evolución en cada una de ellas y en qué punto se considera que se encuentra nuestra aproximación. La Figura. 1 muestra una representación del espacio basada en tres coordenadas, cada una de las cuáles representa una tendencia: Ingeniería Web, Arquitectura Web y MDE. Situándose en un estado evolutivo dentro de cada coordenada o tendencia, WebSA constituye un cubo cuyo volumen representa el espacio que ocupa dentro de la investigación actual.

La coordenada X está representada por la Ingeniería Web, y en ella se presentan las diferentes vistas que se han ido introduciendo a lo largo de su existencia. En ella puede apreciarse cómo las tres vistas funcionales son las primeras en ser capturadas: contenido, navegación y presentación. La cuarta vista, también capturada por WebSA, es la arquitectura estática. La única vista que no es representada por WebSA es la arquitectura dinámica, que representaría el comportamiento tanto externo como interno de los componentes. Esta tarea no ha sido cubierta por el presente trabajo.

La coordenada Y es ocupada por la ingeniería dirigida por modelos (MDE) y en ella se muestran los diferentes estadios por los que ha ido pasando: representación de las vistas mediante modelos (modelado), formalización de los modelos mediante el metamodelado, definición de las transformaciones modelo a modelo para establecer la traza entre los diferentes modelos del proceso y, por último, las transformaciones modelo a texto que permiten obtener la implementación a partir de los modelos. WebSA ha completado las diferentes tareas de MDE. Queda sin embargo un largo camino de interoperabilidad entre los modelos y transformaciones realizadas, que permitan establecer una red para compartir los recursos definidos en MDE.

Por último, la coordenada Z la representa la Arquitectura Web. En ella se muestra una serie de puntos que representan el papel que ha tenido la arquitectura dentro de las aproximaciones. En un principio la arquitectura únicamente se utilizaba para documentar. Poco a poco se ha ido extendiendo su uso como guía para definir el diseño. El último estadio, que es el ocupado por WebSA, es aquel en el que la arquitectura define los artefactos más importantes, dirigiendo el proceso de desarrollo.

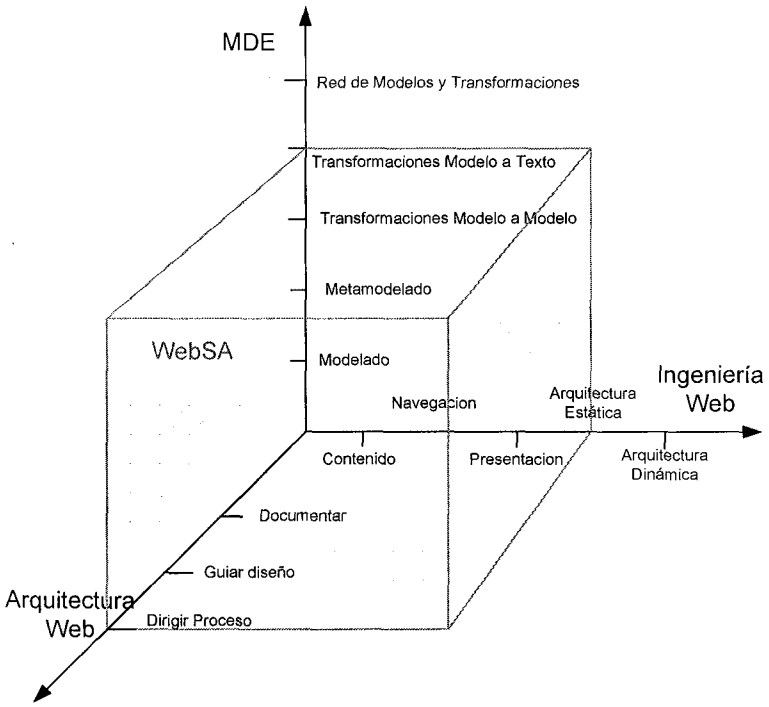

Figura. 1. Espacio de WebSA en la Investigación Actual

2.6 Conclusiones del Capítulo

En este capítulo se ha intentado plasmar cuál es la situación actual de la investigación relacionada con la presente tesis. En este sentido, WebSA es una aproximación que involucra diferentes disciplinas o tendencias y por lo tanto se hace necesario diferenciar cada una de ellas para mostrar los trabajos que se han considerado más relevantes.

Este repaso ha comenzado con las aproximaciones que se encargan de reflejar los aspectos funcionales de las aplicaciones Web, es decir, representan el contenido o dominio, la navegación y la presentación. La comparativa realizada ha tenido como objetivo el mostrar las aproximaciones que son compatibles o que pudieran ser complementadas por nuestra aproximación. En ellas, se ha apreciado una evolución hacia la utilización de los estándares OMG, y por tanto 4 de las 6 propuestas son compatibles con WebSA.

A continuación se ha realizado una comparativa entre las aproximaciones que definen la Arquitectura del Software para Web. Estas propuestas son bastante heterogéneas, pero se han encontrado puntos comunes en la intención de representar los elementos propios de las aplicaciones Web. De este modo, ha sido la definición de

topologías de componentes propios de la Web y el uso del modelado para la representación de la arquitectura los dos aspectos que destacan como comunes.

La última sección ha hecho un repaso de las aproximaciones que se basan en el desarrollo dirigido por modelos para definir los artefactos del desarrollo para Web. Estas aproximaciones, debido a que MDE es muy reciente, son escasas y presentan en su mayoría un estado inicial. De las aproximaciones mostradas, Tai y MIDAS utilizan técnicas basadas en generación de código que hace muy discutible su idoneidad dentro de una aproximación MDE. Por su lado, MDA-UWE es una propuesta muy ambiciosa pero se encuentra todavía en un estado inicial. CA-W2000 se centra en la corrección y validación de sus modelos y no propone técnicas para automatizar un desarrollo de aplicaciones.

El segundo objetivo de este capítulo ha sido introducir, junto a las propuestas, las características propias de WebSA. Para su correcta comprensión, en el siguiente capítulo se retoman sus características principales y se explican con detalle los fundamentos en los que se basa cada una de ellas.

CAPÍTULO 3

Fundamentos de WebSA

"Comprender las cosas que nos rodean es la mejor preparación para comprender las cosas que hay mas allá".

[HIPATIA (aprox. 370-aprox. 415)]

Este capítulo presenta un recorrido general sobre las áreas de conocimiento en las que se fundamenta WebSA, y así poder hacer comprensible los capítulos posteriores donde se detalla la aproximación. Inicialmente se define una visión general de WebSA y la influencia de las diferentes áreas.

3.1 Visión general de WebSA

WebSA (Arquitectura del Software para Web) es una aproximación cuyo principal objetivo es proponer un proceso para el desarrollo de aplicaciones Web centrado en la arquitectura del software. WebSA intenta cubrir el hueco existente entre los modelos de diseño Web tradicional y la implementación final obtenida. Para poder alcanzar este objetivo, se centra en la elaboración de un conjunto de modelos de arquitectura que complementan las actuales metodologías de la ingeniería Web tales como OO-H[19], UWE [59], WebML[20], etc.

La Figura. 2 presenta el mapa de conocimiento de las diferentes áreas o disciplinas que influyen e intervienen dentro de la propuesta WebSA. La figura hace referencia a las tres áreas de conocimiento o disciplinas principales: Ingeniería Web, Ingeniería dirigida por modelos y Arquitectura del Software.

La primera disciplina está encargada de la definición de un proceso para la obtención de una aplicación Web de calidad, es la llamada Ingeniería Web (ver definición en la sección 3.4). Los modelos propuestos por WebSA han sido estudiados para representar de forma adecuada los diferentes aspectos que representan una aplicación Web. Para ello, WebSA se basa en trabajos y métodos previos definidos dentro del paradigma de la Ingeniería Web.

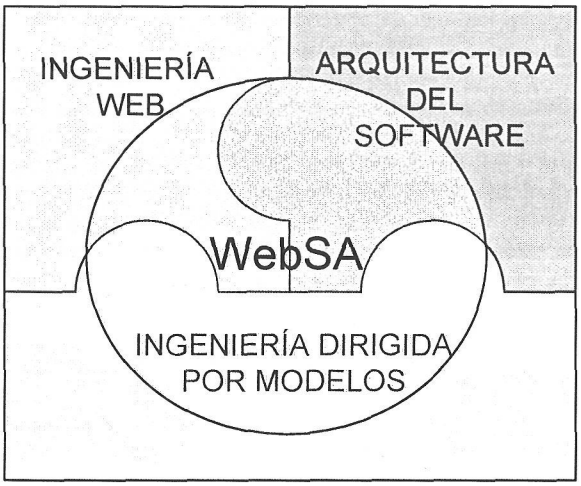

Figura. 2. Mapa de Áreas de Conocimiento de WebSA.

La segunda disciplina en la que se basa WebSA es la Arquitectura del Software (AS) (ver definición en la sección 3.3). El estudio de la arquitectura del software para la definición de una vista de arquitectura en el dominio de las aplicaciones Web, es una de las principales aportaciones de este trabajo. Para ello, se presentan las diferentes alternativas para representar las aplicaciones Web y se justifica la elección de los estilos arquitectónicos centrados en un dominio específico. WebSA considera a los artefactos de la vista de arquitectura como ciudadanos de primera clase, estableciendo un proceso de desarrollo dirigido por los artefactos de arquitectura (proceso unificado (UP) [50]).

Además, la definición de estilos arquitectónicos permite el uso de los patrones como mecanismo de reutilización y aceleración del proceso de desarrollo (ver sección 3.3.3). Por un lado, los patrones de arquitectura y diseño de carácter general que pueden expresarse en los modelos de WebSA, p.e. Buchmann et al.[17] y Gamma et al. [36]. Por otro lado, aquellos patrones que están más centrados en el dominio de las aplicaciones Web como Conallen [24].

La tercera disciplina en la que se basa WebSA es la Ingeniería dirigida por Modelos (MDE) [55] y más concretamente su instancia estandarizada por la OMG Model Driven Architecture (MDA) [89] (ver sección 3.2). WebSA define un proceso de desarrollo como una instancia proceso unificado estándar UP [50] pero que sigue la filosofía de MDE basada en el uso de los modelos (ver definición de modelo en la sección 3.2.2.1) como elementos principales del desarrollo e incorporando la trazabilidad entre las diferentes fases del desarrollo mediante el uso de las transformaciones. En este sentido, WebSA propone la utilización de modelos propios y modelos procedentes de otras propuestas para enriquecer el proceso, mediante las transformaciones permite la fusión del contenido de los modelos y la automatización desde el análisis hasta la implementación.

El uso del estándar MDA posibilita también la estandarización de cada uno de los artefactos definidos dentro de las aplicaciones Web. Para ello, WebSA hace uso de

MOF para el establecimiento de los metamodelos, UML para la definición de la notación de los diferentes modelos y para la definición de las transformaciones.

Como se aprecia en la Figura. 2, WebSA propone la integración de las tres disciplinas encajándolas como un puzzle en el que lejos de considerarse de una forma aislada, se establecen puntos de convergencia entre cada una de las ellas que en ocasiones se complementan o se restringen. Concretamente, la Arquitectura del Software interviene en el desarrollo de la Ingeniería Web, pero es la ingeniería dirigida por modelos quien tiene influencia sobre las otras dos áreas. Otro aspecto importante a considerar es que debido a la magnitud de las tres áreas, WebSA únicamente cubre la parte necesaria (representado mediante un círculo) que le permita el objetivo de la obtención de una aplicación Web.

Seguidamente se recorre cada una de las diferentes áreas de conocimiento haciendo hincapié en aquellos aspectos que son necesarios para la correcta comprensión de la aproximación WebSA. Para ello, las influencias entre las áreas mostradas en la Figura. 2 determinarán el orden. Primero, se abarca la Ingeniería Dirigida por Modelos debido a que influye en las otras dos áreas y es necesario introducirla primero. Seguidamente, la Arquitectura del Software debido a que interviene en la Ingeniería Web y para finalizar Ingeniería Web.

3.2 Ingeniería Dirigida por Modelos

WebSA se ha basado en la Ingeniería dirigida por modelos (MDE) [55] para la definición de los artefactos producidos a lo largo del proceso de desarrollo para la obtención de las aplicaciones Web. Y más concretamente, en su propuesta estandarizada por la OMG llamada MDA (Model Driven Architecture) [89] que permiten la definición de los artefactos producidos siguiendo los estándares.

En este sentido es importante conocer en que consiste el concepto MDE y su influencia en WebSA. MDE es un framework conceptual definido por Stuart Kent [55] "*aborda el escurridizo problema del desarrollo de sistemas, promoviendo el uso de los modelos como primer artefacto para su construcción y su mantenimiento*". Sin embargo, el artefacto realmente novedoso definido por MDE son las transformaciones que permiten establecer un mapeo trazable desde los modelos definidos hasta la implementación final. Esta nueva tendencia es llamada también Ingeniería de Modelos por Bezivin [12].

Dentro del framework MDE, WebSA se centra en la versión más estándar y conocida llamada MDA. MDA es tal vez la más prometedora y ambiciosa propuesta de la OMG para abordar los nuevos retos de los actuales sistemas software que se encuentran en constante cambio y con muy alto grado de interconexión. MDA se basa en los conceptos de MDE pero mediante la utilización de los estándares de la OMG como son: MOF [88], XMI[12] [97], OCL[13] [93], UML[14] [95], CWM[15] [86]. Con la combinación de estos lenguajes y el uso de mecanismos tales como abstracción,

[12] XMI: XML Metadata Interchange (acrónimo inglés de Intercambio de metadatos XML)
[13] OCL: Object Constraint Language (acrónimo inglés de Lenguaje de restricciones de objetos)
[14] UML: Unified Modeling Language (acrónimo ingles de Lenguaje de Modelos unificado)
[15] CWM: Common Warehouse Metamodel (acrónimo inglés de Metamodelo de almacenes común)

refinamiento y vistas, MDA define un conjunto de modelos, CIM[16], PIM[17] y PSM[18], los cuáles definirán el sistema desde diferentes perspectivas y niveles de abstracción. Además, MDA establece un proceso de refinamiento, llamado MDA *Development Process* [57] que provee de la trazabilidad entre los modelos y el código final, esto obtenido mediante un conjunto de transformaciones modelo a modelo y modelo a código.

WebSA no es la primera propuesta que propone el uso de MDA para el desarrollo de aplicaciones en el contexto de dominios específicos; de hecho se ha basado en otros trabajos como DSMDA (Domain specific MDA) [1], una aproximación concebida para guiar al desarrollador en el uso de MDA para conseguir una especificación del problema más exhaustiva y sencilla, mientras garantiza adecuada correspondencia con la solución final.

Para poder abordar esta aproximación, WebSA propone la formalización de los modelos por medio de la definición de los metamodelos MOF y un conjunto de restricciones OCL que juntas especificarán (1) las semánticas ligadas con cada elemento del modelo, (2) cuáles son las configuraciones validas y (3) cuáles son las restricciones aplicadas.

Por otro lado, MDA promueve el uso de un conjunto de estándares entre los que se encuentra el actual estándar para especificar modelos UML. En este sentido, en su última versión UML 2.0, se han incorporado una serie de propiedades que ayudan a la adaptación de MDA, como son:

- La incorporación de modelos como el Composite Structure que permiten especificar de una forma más precisa y escalable la arquitectura del software y el desarrollo basado en componentes.
- La posibilidad de definir extensiones de los modelos mediante un mecanismo de perfiles, que proporcionará la posibilidad de especializar el dominio de los modelos ayudando a ser más precisos en las especificaciones.
- Una mejora en la especificación del metamodelo de UML que proporciona un mayor alineamiento entre MOF, los modelos UML y los perfiles basados en UML. Esto ayudará a dotar de mecanismos de transformación sobre los metamodelos de una forma más correcta.

Por ultimo, resaltar la incorporación del estándar Query/Views/Transformations (QVT[19]) [91] que ha sido introducido para la especificación de las transformaciones entre los diferentes modelos. En WebSA, se ha trabajado con QVT, pero se han presentado ciertas carencias en su notación gráfica declarativa y en las herramientas que lo soportan. Por ello, se ha optado por definir un nuevo lenguaje de transformaciones llamado UML profile for transformations (UPT[20]). Que permite la definición de las transformaciones a nivel de metamodelo, utilizando un perfil sobre el diagrama de clases de UML. El capítulo 8 muestra como UPT permite formalizar las transformaciones entre los diferentes modelos, incorporar modelos de otras

[16] CIM: Computation Independent Model (acrónimo inglés de Modelo Independiente de Computación)
[17] PIM: Platform Independent Model (acrónimo inglés de Modelo Independiente de Plataforma)
[18] PSM: Platform Specific Model (acrónimo inglés de Modelo Específico de Plataforma)
[19] QVT: Query/Views/Transformations (acrónimo inglés de Consulta/Vista/Transformación)
[20] UPT: UML Profile for Transformations (acrónimo ingles de Perfil UML para Transformaciones)

aproximaciones como OO-H [19] y UWE [59] y así para obtener la parte funcional e integrarla con la arquitectura especificada por WebSA.

A continuación, se realiza una definición de MDA y de sus diferentes artefactos utilizados.

3.2.1 Definición de MDA

MDA se estableció por primera vez en la especificación definida por la OMG [89] en donde se indicaba que: "MDA define una aproximación para la especificación de sistemas de información que separa la funcionalidad del sistema de la implementación para una plataforma tecnológica específica". Para conseguir este fin, MDA define una arquitectura de modelos que provee las guías para estructurar las especificaciones expresadas como modelos. Esta separación entre la operación del sistema y los detalles específicos de la propia plataforma proporciona las siguientes ventajas:

- Especificar el sistema independientemente de la plataforma que la soporta.
- Especificar diferentes plataformas.
- Elegir una plataforma específica para el sistema.
- Transformar la especificación del sistema en una plataforma particular.

Como se indica la MDA guide [87], los tres principales objetivos de MDA son la interoperatibilidad, la usabilidad y la reutilización a través de la separación de aspectos.

MDA especifica una arquitectura de modelos que se incorpora a los procesos de desarrollo tradicionales, reduciendo el trabajo en las fases de elaboración y acelerando este proceso. WebSA define un proceso de desarrollo instancia del Unified Process (UP) [50] que será tratado extensamente en el capítulo 4.

A continuación, se muestran cuáles son los conceptos principales en los que se basa MDA.

3.2.2 Conceptos Básicos de MDA

A continuación se realiza un recorrido sobre aquellos conceptos más relevantes que constituyen la especificación MDA y sobre los que se basa WebSA para especificar sus artefactos dentro de su proceso de desarrollo.

3.2.2.1 Modelo

Como su propio nombre indica MDA es una arquitectura dirigida por modelos, enfatizando el hecho de que los modelos son el punto principal de MDA. MDA tiene en consideración los modelos que son relevantes para el desarrollo del software. Es importante indicar que estos modelos no solamente incluyen los que hacen referencia al software, sino que al estar el software realizado para soportar un negocio, el modelo de negocio también relevante.

Pero que se desea indicar exactamente cuando se utiliza la palabra modelo. Llegar a una definición que abarque todos los tipos de modelos es muy difícil. Además, la definición necesita ser lo suficientemente específica para que ayude a especificar la información necesaria para obtener una transformación automática de un modelo a otro.

Un modelo está siempre escrito en un lenguaje. Este lenguaje podría ser UML, un lenguaje natural, un lenguaje de programación o cualquier otro lenguaje que se pueda imaginar. Pero para permitir una transformación automática desde un modelo, en MDA se restringe a aquellos modelos que estén escritos en un lenguaje bien definido. Un lenguaje tiene una forma y una semántica bien definida cuando este puede ser interpretado automáticamente por un computador. En este sentido, no se pueden considerar los lenguajes naturales como bien definidos, porque no pueden ser interpretados por las computadoras. De esta manera, para los propósitos de este trabajo se adopta la siguiente definición de modelo:

"Una descripción (o parte) de un sistema escrito en un lenguaje bien definido". Kepple et al. [57].

Siendo un lenguaje bien definido un lenguaje con la sintaxis y la semántica bien definidas, cuya interpretación puede automatizarse por un computador. En este sentido aunque en WebSA se han definido los modelos basándose en el estándar UML, MDA no está restringido a UML.

Dentro de MDA existe una topología de modelos que permite identificarlos según su función dentro del proceso de desarrollo del software. Estos modelos no se definen de forma aislada sino que están vinculados entre ellos mediante transformaciones:

- **CIM (Modelo Independiente de la computación):** Este modelo es independiente de la manera en la que el sistema está implementado. Es un modelo que muestra el sistema en el entorno en el cual operará y así ayuda a representar exactamente qué se espera que el sistema haga.

  CIM es útil para ayudar a entender el problema y como fuente de un vocabulario compartido para otros modelos. En MDA los requisitos del sistema representados por un modelo CIM permiten la trazabilidad a los constructores de los modelos PIM y PSM que lo implementan.
- **PIM (Modelo Independiente de Plataforma):** Describe el sistema, pero no muestra los detalles de plataforma. En este sentido independiente de plataforma es un término relativo. Cuando se indica que un lenguaje o modelo es independiente de plataforma, se debe especificar de que plataforma tecnológica es independiente. Por ejemplo los modelos PIM presentados en WebSA, son independiente de las plataformas tecnológicas que implementan la Web, desde las tecnologías que se encargan de la presentación y lógica de presentación (HTML, ASP, JSP, etc.), también de las que se encargan del middleware (EJB, CORBA, COM+, etc.) y de la persistencia (bases de datos relacionales, orientadas a objetos, etc.). En este sentido, los modelos PIM se entiende que están en un escalafón de abstracción más alto que lo que normalmente entiende el programador.
- **PSM (Modelo específico de Plataforma):** Es un modelo computacional que especifica información o formato de tecnología, ya sea un lenguaje de programación, un componente middleware distribuido, etc. Igualmente al caso anterior, un modelo específico se trata de un término relativo. Se dice que un modelo es más específico de plataforma que otro cuando representa conceptos más concretos de la plataforma destino. En este sentido WebSA define un conjunto de modelos PSM cada uno de los cuáles son específicos a las posibles plataformas sobre las cuáles se puede

obtener la implementación de la aplicación Web. En WebSA, los modelos PSM son más específicos que el modelo de integración que se define sin llegar a especificar ningún detalle sobre las plataformas.
También es importante señalar, que un modelo PSM puede especificar tanto un diseño de bajo nivel como la propia implementación destino.

3.2.2.2 Metamodelo

Como se indicaba en el punto anterior, los lenguajes que requiere MDA deben ser lenguajes bien definidos, es decir, deben poder ser interpretados por el computador. En el pasado, esto estaba establecido representado el lenguaje mediante BNF [21] el cual describía la serie de tokens que seguían una expresión correcta. Sin embargo, BNF se restringe a aquellos lenguajes que son puramente texto y en este caso la mayoría de modelos que se utilizan en MDA tienen una sintaxis gráfica, como UML. Para ellos se utiliza un mecanismo denominado metamodelo.

Como se define en la especificación de UML 2.0 [95] *"un metamodelo es un modelo que especifica el lenguaje para expresar un modelo"*.

Según esta definición, todos los elementos que un modelador puede usar en su modelo deben estar definidos previamente en el metamodelo del lenguaje que el modelador usa. Así por ejemplo, en el modelo de clases de UML se pueden utilizar clases, atributos, operaciones, asociaciones, etc. gracias a que en el metamodelo de UML esta definido el concepto de clase, como esa clase contiene un conjunto de elementos llamados atributos, además como la clase se relaciona a través de conceptos como asociación o herencia. Es decir todo lo necesario para estructurar un modelo de clases.

Debido a que un metamodelo es también un modelo, el mismo metamodelo debe estar escrito en un lenguaje bien definido. Este lenguaje es llamado meta-metamodelo, así en MDA se define el estándar MOF que permite definir los metamodelos de cada uno de los lenguajes definidos como UML, CWM, XMI, etc. Cada uno de estos estándares de MDA y las relaciones existentes entre los ellos, la OMG define una arquitectura de cuatro capas. En la Figura. 3 se puede apreciar un gráfico donde aparece la infraestructura de capas de MOF que consiste en una jerarquía de modelos, donde cada uno de los modelos definidos en una capa es una instancia de los elementos que se encuentran en el modelo definido en la capa superior.

Por ejemplo, los elementos que existen en la capa M1 (clases, atributos y otros elementos) son las instancias de las clases de M2. Un elemento de la capa M2 especifica elementos de la capa M1.

En la parte superior, el nivel M3 permite representar los conceptos para definir los metamodelos, es el llamado meta-metamodelo y el lenguaje utilizado es MOF. Así, las instancias de la clase *Class* de MOF sirven para representar los metamodelos. En M2 se sitúan los metamodelos que sirven para definir los elementos y relaciones de los modelos, en este ejemplo la clase *Class* de UML con un nombre sirve para representar cualquier clase de UML. Así, una instancia de la metaclase *Class* será una clase de un modelo que representa a un sistema, en la figura es una clase artículo. Así, el nivel M1 representa el modelo y contiene las clases y elementos de otros modelos

[21] BNF: Backus-Naur Form (acrónimo inglés que especifica Forma de Juan Backus y Meter Naur)

que representan un problema en concreto. Por último, las instancias de los modelos que están en el nivel M0 representan los valores de las clases en cualquier momento de la ejecución.

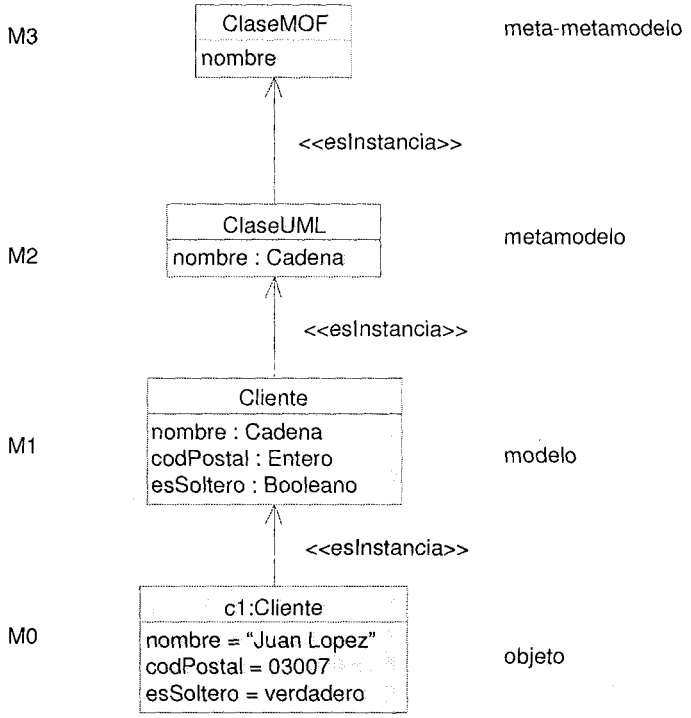

Figura. 3. Arquitectura de Cuatro Capas de MOF.

La nomenclatura seguida para definir cada una de las capas es M0, M1, M2 y M3. A continuación se detalla el significado de cada una de ellas.

- **Capa M0: El Objeto:** En esta capa se define el sistema en ejecución, en el vive un conjunto de objetos o instancias reales. Por ejemplo, si se especifica un sistema de comercio electrónico, entre otros conceptos existen los artículos del sistema. Una instancia mostrada en el ejemplo serían un cliente de nombre "Juan Lopez", su código Postal es 03007 y es soltero. Dependiendo de los conceptos que se estén representando, las instancias podrían ser los objetos de negocio de nuestra aplicación si se representa el dominio del sistema o componentes software si se representa la arquitectura de nuestra aplicación.
- **Capa M1: El modelo:** Esta capa contiene el modelo que representa un sistema software concreto. El modelo puede tratarse de un modelo de clases UML, un modelo de navegación o un modelo que cumpla la condición de que se pueda metamodelar cada uno de los elementos que lo

componen. Siguiendo el ejemplo anterior del comercio electrónico, el modelo de dominio tiene que representar a la clase Cliente que contendría los atributos nombre de tipo cadena, codPostal de tipo entero y esSoltero de tipo booleano. En el caso de WebSA, cualquiera de los modelos que representa la arquitectura y la funcionalidad Web sería un modelo de la capa M1.

- **Capa M2: El metamodelo:** Los modelos definidos en esta capa son denominados metamodelos. La función del metamodelo es modelar los elementos que constituyen los modelos del sistema. Es decir, definir el lenguaje con el que se escriben los modelos de los sistemas, especificando las restricciones entre ellos y las posibles relaciones. Un ejemplo es el metamodelo definido por la superestructura de UML 2.0 [95], en la que mediante un modelo de clases, de definen todos los elementos que definen los 11 modelos de UML 2.0. En la Figura. 3 se muestra el elemento Class del metamodelo de UML. En WebSA a la hora de formalizar los modelos de arquitectura se ha especificado un metamodelo en el cual se describen cada uno del los elementos que forman parte de los modelos, cuáles son sus relaciones y restricciones a la hora de representarlos. Además, estos metamodelos sirven para definir las transformaciones de los modelos.
- **Capa M3: El meta-metamodelo:** Si se sube todavía una capa más en la jerarquía de modelos, se necesita definir cuáles son los elementos y las relaciones que se definen en los metamodelos. Para ello, se define otro lenguaje con el que se define un metamodelo, en este caso se llama meta-metamodelo. Para este modelo de la capa M3, la OMG propuso un estándar denominado MOF [88]. Así todos los lenguajes de modelado propuestos por la OMG, como son UML, CWM, XMI, etc. son instancias de MOF. En la Figura. 3 se muestra como se define el concepto de Class MOF que define el Class del metamodelo de UML. Por último, indicar que se podría seguir definiendo más capas, es decir, se pueden modelar los conceptos que se representan en el modelo M3 en una capa M4, y así sucesivamente. Sin embargo, se ha demostrado que no es práctico. La OMG propone que todos los elementos de la capa M3 se deben definir como instancias de los mismos elementos de la capa M3.

3.2.2.3 Transformación

Dentro del paradigma de la Ingeniería dirigida por Modelos, las transformaciones tienen un papel fundamental ya que establecen el vínculo que permite unir cada uno de los modelos especificados en el proceso de desarrollo. Hasta ahora se han mostrado los modelos y metamodelos utilizados para especificar el sistema. Sin embargo, el fin último del proceso de desarrollo de MDE es poder obtener el código final de la aplicación a partir de los modelos. Para poder conseguir este objetivo, se necesita un mecanismo de refinamiento que permita trazar de una forma precisa, como llegar desde los modelos situados en el nivel de abstracción más alto hasta el código de la aplicación. Para ello, se establece el artefacto transformación. En MDA guide [87] se define una transformación de modelos como *"el proceso de transformar un modelo origen en otro modelo destino del mismo sistema"*. Esta definición hace más hincapié en el proceso en sí, que en la forma de especificar la transformación. Sin

embargo, hay que aclarar que MDA promueve que la definición de la transformación se especifique como otro modelo más, de manera que haya modeladores encargados de especificar las transformaciones, modificarlas y mantenerlas. Se entiende la transformación como *"un conjunto de reglas que describen como un modelo en el lenguaje origen es transformado en un modelo en el lenguaje destino"*. En esta definición destaca la palabra lenguaje, que como ha sido comentada en la sección anterior, los lenguajes de modelos son especificados mediante metamodelos. Por ello, existe una clara vinculación entre el metamodelado y las transformaciones, ya que estas últimas definen cada una de sus reglas a partir de los elementos definidos en el metamodelo del modelo origen y el metamodelo del modelo destino.

3.2.2.4 Clasificación de Transformaciones

Seguidamente se realiza una clasificación de los diferentes tipos de transformaciones. Esto permite indicar a que tipo de transformación se ajustan las transformaciones definidas dentro de WebSA. Se clasifican las transformaciones según diferentes criterios:

Según los modelos de origen y destino:

- **Tienen el mismo modelo origen que destino:** La transformación se define entre dos modelos del mismo metamodelo. La transformación realiza una evolución de un modelo en otro modelo que ha sido modificado debido a una evolución del sistema, refinamiento, etc.
- **Tienen diferente modelo origen y destino:** Sería el caso por ejemplo, de una transformación que tiene un modelo UML que representa el domino del problema y lo transforma en un modelo Entidad-Relación para proceder al almacenamiento.

Según el tipo de destino:

- **Modelo a Modelo:** La transformación se define entre los elementos de un modelo que son transformados en elementos de otro modelo.
- **Modelo a Texto:** La transformación define como los elementos de un modelo se transforman en texto o en código.

Según el nivel de abstracción una transformación puede ser (France & Bieman [33]):

- **Transformación Vertical**: ocurren cuando un modelo origen es transformado en un modelo destino en un nivel de abstracción diferente.
- **Transformación Horizontal**: cuando las transformaciones ocurren entre un modelo origen y destino en el mismo nivel de abstracción.

Según el rol del modelo dentro del proceso de MDA:

- **Transformar un modelo PIM a un modelo PSM:** Tal vez la más común dentro de MDA, dicha transformación incorpora a los elementos independientes de plataforma, la información necesaria para generar los elementos que son específicos de la plataforma. Por ejemplo, a partir de un modelo de dominio se genera un modelo java, el cual puede obtener conceptos como entity bean, session bean, etc.
- **Transformar un modelo PIM en un modelo PIM:** Esta transformación se define cuanto se evoluciona un modelo independiente de plataforma, para mejorarlo o filtrarlo durante el proceso de desarrollo.

- **Transformar un modelo CIM en un modelo PIM:** Esta transformación se define cuando se especifica el sistema en un modelo CIM independientemente de la implementación de este y se transforma en un PIM en el que representa los artefactos del sistema de forma independiente de la plataforma destino.

Según los mecanismos utilizados para especificar una transformación (especificado en la MDA guide [87]):

- **Transformación basada en marcas**: Las marcas son utilizadas para marcar elementos en el modelo de origen y así guiar la transformación al modelo destino. Por ejemplo, se puede marcar una clase del dominio como persistente y con ello se indica que dicha clase se ha de transformar en una tabla de base de datos.

- **Transformación basada en el metamodelo**: La especificación de este tipo de transformación se realiza en términos de un mapeo entre los elementos de los metamodelos. De esta manera se debe trabajar con el metamodelo del modelo origen (que en MDA debe estar especificado en MOF) y el metamodelo del modelo destino que de la misma manera esta expresado con el lenguaje MOF. Por ejemplo, si se transforma un modelo de clases de UML en un modelo entidad-relación (ER) de bases de datos, una regla de transformación seria convertir la clase "class" del metamodelo de UML a la clase "Table" del metamodelo de ER.
- **Transformación basada en modelo**: La especificación de esta transformación esta expresada en términos de un mapeo entre los tipos del modelo de origen y los tipos de los modelo de destino. Esta aproximación difiere principalmente de la basada en metamodelo en que los tipos especificados en el modelo son usados para el mapeo, en lugar de los conceptos especificados en el metamodelo.
- **Transformación basada en patrones**: Esta transformación se puede plantear como una extensión de las transformaciones basadas en modelos y en metamodelos, las cuáles incorporan el uso de patrones ya sea mediante tipos o mediante conceptos del lenguaje de modelado. Un ejemplo sería si se utiliza el patrón de arquitectura MVC, dentro de un modelo PIM y se genera un modelo PSM en el destino que implementa este patrón utilizando la tecnología de Struts (específica para este).
- **Fusión de modelos**: La fusión de modelos implica que en lugar de tener un único modelo origen, se tienen dos o más modelos origen que se van a fusionar en un solo modelo destino. Esto establece que las reglas de transformación interrogan conceptos sobre dos o más modelos en origen y a partir de ellos, generarán los conceptos en un modelo destino.

Hay que indicar que en la última clasificación cada uno de los tipos no son excluyentes, es decir, una transformación puede ser de más de un tipo simultáneamente, por ejemplo, una transformación basada en metamodelo puede utilizar la fusión de modelos a la hora de recoger la información de origen.

Una vez expresado un conjunto de clasificaciones de transformaciones, se puede indicar en que situación se encuentran las transformaciones definidas en WebSA. En

el proceso definido por WebSA (ver capítulo 4) existen básicamente 2 transformaciones. La transformación T1 (ver capítulo 8), en la que se realiza una fusión de los modelos funcionales y los modelos de arquitectura definidos en la fase de análisis. Y la transformación T2 (ver capítulo 8), que realiza una transformación del modelo de integración en los diferentes modelos dependientes de las plataformas .NET, J2EE, etc. Entonces, según las diferentes clasificaciones de transformaciones expresadas anteriormente, las transformaciones T1 y T2 de WebSA tienen las siguientes características:

Transformación T1 (Integración de Funcionalidad y Arquitectura):

- Distinto origen y destino. Se transforman los modelos definidos en la fase de análisis en los modelos definidos en el diseño.
- Modelo a Modelo. Parte de los modelos de análisis y los convierte en un modelo de diseño.
- PIM a PIM. Transforma los modelos independientes de plataforma en otros modelos que siguen siendo independientes de plataforma.
- Basada en los metamodelos. A la hora de especificar las transformaciones, estas se realizan utilizando los metamodelos de los modelos de origen y destino.
- Fusión de modelos. Realiza la fusión de los modelos definidos en la perspectiva funcional Web con los modelos definidos en la perspectiva arquitectónica Web.

Transformación T2 (Conversión del diseño a la plataforma):

- Distinto origen y destino. Se transforma el modelo de diseño en las implementaciones en J2EE y .NET.
- Modelo a Texto. Se transforma de un modelo a código mediante el uso de plantillas definidas en MOFScript o Velocity.
- PIM a Código. Se transforma el modelo de diseño independiente de plataforma en código de las diferentes plataformas.
- Basada en los metamodelos. Para especificar las transformaciones se realiza consultando el metamodelo de origen y estableciendo el mapeo al destino.
- Insertan aspectos técnicos. En T2 se especifica como introducir los aspectos dependientes de plataforma a partir del modelo de diseño.

3.2.3 Los Estándares de MDA Utilizados en WebSA

Un aspecto no menos importante que caracteriza MDA, es el conjunto de estándares que la OMG ha propuesto para su implementación. Las principales ventajas que el uso de los estándares proporcionan son: (1) realizar un desarrollo dirigido por modelos apoyándose en las herramientas existentes (comerciales o no), (2) los modelos utilizados son comprensibles para un mayor número de personas y (3) se reduce la curva de aprendizaje de cualquier desarrollador que quiera ponerse al día en esta forma de trabajo. Por ello, se describen los estándares más importantes de MDA que son utilizados por WebSA y que corresponden con los conceptos especificados en la sección anterior. Estos estándares permiten la definición de metamodelos, modelos, transformaciones, consultas e intercambio de datos. Los estándares presentados en

esta sección son MOF, UML, XMI. UPT hace uso de UML (ver capítulo 8) para la definición transformaciones.

3.2.3.1 MOF

Uno de los estándares más importantes usados en MDA es el Meta Object Facility o MOF [88]. Tecnología adoptada por el OMG para definir metadatos y su objetivo es proporcionar un marco y un conjunto de servicios para permitir realizar sistemas dirigidos por modelos y metadatos. El término metadato representa datos que describen información. Una información puede ser representada en un sistema computador (por ejemplo, ficheros, bases de datos, instancias de programas en ejecución,...), o que puede ser una descripción de algún aspecto del sistema como una parte de su diseño.

MOF soporta cualquier tipo de metadato que pueda ser descrito usando técnicas de modelado de objetos. Estos metadatos pueden describir cualquier aspecto de un sistema y la información que contiene, y puede describir a cualquier nivel de detalle y rigor dependiendo de los requisitos.

Los metadatos son un tipo de información y pueden describirse de acuerdo a otros metadatos. En la terminología MOF, un metadato que describe metadatos se llama meta-metadato, y un modelo que sea un meta-metadato se llama metamodelo.

El metamodelo MOF define la sintaxis abstracta de los metadatos en la representación MOF de un modelo. Ya que hay muchos tipos posibles de metadatos en un sistema típico, el marco MOF necesita soportar muchos metamodelos MOF diferentes. MOF integra estos metamodelos definiendo una sintaxis abstracta común para definir metamodelos. Esta sintaxis es el Modelo MOF, y es el modelo para definir metamodelos, es decir, es un meta-metamodelo. Además el marco de metadatos MOF es descrito con una arquitectura de cuatro capas (ver Figura. 3).

MOF ofrece diversa funcionalidad adicional:

- Interfaz del Repositorio MOF. Se trata de una interfaz que permite obtener información sobre modelos de nivel M1 de un repositorio basado en MOF.
- Intercambio de Modelos. Para el intercambio de modelos, MOF se basa en un formato de intercambio basado en XML llamado XMI. Ya que MOF se define usándose a sí mismo, puede usarse también XMI para generar formatos de intercambio de metamodelos.

Mientras MOF y UML están diseñados por dos tipos diferentes de modelado, es decir, metadatos frente a modelado de objetos, MOF y el núcleo del metamodelo UML están estrechamente ligados en sus conceptos de modelado. Gracias a ello, se permite usar la notación UML para expresar metamodelos basados en MOF. Como se muestra en la especificación, cada uno de los modelos utilizados dentro de WebSA, es formalizado mediante la representación de sus metamodelos. Estos metamodelos son representados mediante la notación UML para el diagrama de clases, que permite representar las metaclases y sus relaciones de asociación y herencia. Sin embargo, para completar la definición de los metamodelos se necesitan establecer restricciones que indiquen algunas combinaciones que no pueden restringirse con el metamodelo.

3.2.3.2 UML

Para representar los modelos que especifican la arquitectura específica de las aplicaciones Web, WebSA se ha decantado por la utilización de un estándar gráfico como UML [95], que proporciona una serie de ventajas como (1) disponer de una semántica y sintaxis precisa, permitiendo conocer de forma no ambigua lo que el diagrama indica, (2) UML tiene una alta capacidad expresiva lo que posibilita representar todos los aspectos de la arquitectura Web, (3) su capacidad para representar el sistema a diferentes niveles de abstracción, así cada uno de los miembros de desarrollo, arquitectos, diseñadores y analistas pueden enfocarse en los problemas que les ocupan, olvidándose de los demás aspectos, (4) los modelos pueden ser intercambiables entre diferentes herramientas que tengan soporte de UML, así los modelos pueden ser reutilizados e intercambiados entre diferentes equipos y compañías.

A pesar del éxito que ha tenido UML, las herramientas de desarrollo han sido lentas en darse cuenta de su gran capacidad. Además, la industria del software ha evolucionado considerablemente en los últimos años, y la primera versión de UML (UML 1.X) ya no se encuentra actualizada. Dos ejemplos dejan patente este hecho, la capacidad de expresar el desarrollo basado en componentes como J2EE, COM+ y .NET, y segundo, el uso común de otros lenguajes de modelado maduros (SDL, ROOM, etc.) para especificar componentes y arquitecturas de tiempo real. Si se intenta modelar cualquiera de estos aspectos con UML 1.X [120], no cubren bien estos paradigmas y su complejidad. En este contexto, ambos dominios se encuentran relacionados con las aplicaciones Web actuales, ya que dichas aplicaciones necesitan de un desarrollo basado en componentes cuando se trata de abordar la parte servidora de aplicaciones de comercio electrónico. Por lo tanto, surge la necesidad de abordar correctamente estos aspectos.

Afortunadamente, el proceso de estandarización para crear la nueva versión UML 2.0 ha terminado y ya es el estándar de facto (Selic [107]). UML 2.0 entre muchas cosas más, ha incluido un número importante de características que le permiten ser usado como un lenguaje de descripción de arquitecturas (ADL). La base de esas capacidades provienen de trabajos previos en ADLs como ACME [37], RT-UML [108] y SDL [49]. Estos lenguajes soportan un modelado jerárquico de estructuras de tiempo real de comunicación de componentes. Estos conceptos están definidos generalmente de forma recursiva, así que los componentes de un nivel en el sistema jerárquico pueden ser descompuestos en estructuras de grano fino de componentes colaboradores. Esto permite modelar los sistemas en un número arbitrario de niveles de abstracción. Estas propiedades han sido consideradas por WebSA para representar la arquitectura de las aplicaciones Web a diferentes niveles de abstracción.

Concurrentemente con la publicación de UML 2.0, OMG lanzó su iniciativa MDA (Model-Driven Architecture), estando las propuestas vinculadas fuertemente. Por un lado, UML 2.0 representa uno de los pilares de MDA, permitiéndole usar los lenguajes de modelado no solamente para documentar y hacer bosquejos gráficos, sino que sus modelos incluyen las habilidades para permitir una trazabilidad hasta la implementación o realizar análisis formales complejos para determinar el buen estado y la validez de los diseños propuestos. Por otro lado, UML 2.0 ha tenido en cuenta MDA ya que incluye capacidades como la mejora en la formalización de los modelos mediante la separación entre infraestructura y superestructura, y un mejor

planteamiento para abordar la especialización en determinados dominios mediante los perfiles.

3.2.3.3 XMI

El principal propósito de XMI (XML Metadata Interchange) [97] es permitir el fácil intercambio de metadatos entre herramientas de modelado basadas en UML y repositorios de metadatos basados en MOF en entornos distribuidos. XMI integra tres estándares principales:

- XML – eXtensible Markup Language, un estándar del W3C
- UML – Unified Modeling Language, un estándar de modelado del OMG
- MOF – Meta Object Facility, un estándar de repositorio de metadatos y metamodelado del OMG

La integración de estos tres estándares dentro de XMI casa lo mejor de las tecnologías de modelado y metadatos del OMG y del W3C[22], permitiendo a los desarrolladores de sistemas distribuidos compartir modelos y otros metadatos a través de Internet. XMI, junto con MOF y UML forman el núcleo de la arquitectura de repositorio de la OMG.

El estándar UML define un rico lenguaje de modelado orientado a objetos, que se apoya por una gama de herramientas de diseño gráficas. El estándar MOF define un marco extensible para definir modelos para metadatos y proporciona herramientas para guardar y acceder a esos metadatos de un repositorio. XMI permite a los metadatos ser intercambiados como flujos o archivos con un formato estándar basado en XML.

XMI define un formato de intercambio basado en XML que es ampliamente utilizado para compartir objetos usando XML. Además, es aplicable a una amplia variedad de objetos: análisis (UML), software (Java, C++), componentes (EJB[23], CORBA Component Model, etc.), y bases de datos (CWM).

XMI define muchos de los aspectos involucrados en la descripción de objetos en XML:

- La representación de objetos en términos de elementos y atributos XML.
- Debido a que los objetos normalmente están interconectados, XMI incluye mecanismos estándar para enlazar objetos dentro del documento actual o en un documento externo.
- La identidad de objeto permite a los objetos ser referenciados desde otros objetos en términos de IDs y UUIDs.
- La versión de los objetos y sus definiciones están manejados por el modelo XMI.
- La validación de documentos XMI usando DTD[24]s y Schemas.

La especificación XMI proporciona dos componentes principales. El primero de ellos son las Reglas de Producción para obtener DTDs o Schemas que codifiquen metadatos. Estos proporcionan la sintaxis de los documentos XMI y permiten a las herramientas XML genéricas utilizarlos para componer y validar documentos XMI. El

[22] W3C: World Wide Web Consortium (acrónimo inglés de Consorcio de la telaraña mundial)
[23] EJB: Enterprise Java Beans (acrónimo inglés de Componentes Java de Negocio)
[24] DTD: Document Type Definition (acrónimo inglés de Documento de definición de tipos)

segundo componente son las Reglas de Producción de Documentos XML para codificar metadatos en un formato compatible XML. Las reglas de producción pueden aplicarse en inverso para decodificar documentos XMI y reconstruir el metadato.

XMI soporta el intercambio de cualquier tipo de metadato que pueda ser expresado usando la especificación MOF, incluyendo tanto la información del modelo como la del metamodelo. Además soporta la codificación del metadato tanto como modelos completos o como fragmentos de modelo, así como extensiones al metadato. XMI tiene soporte opcional para el intercambio de metadatos en formas diferenciales y para intercambio de metadatos con herramientas que tienen un conocimiento incompleto del metadato.

La siguiente sección se centra en el área de la Arquitectura del Software y en los diferentes aspectos necesarios para su correcto conocimiento.

3.3 La Arquitectura del Software

Para entender en que consiste la propuesta WebSA, se debe conocer que significan las dos últimas siglas, "SA", es decir, Software Architecure (Arquitectura del Software en inglés). La Arquitectura del Software ha sido en ocasiones la gran desconocida dentro de la Ingeniería del Software, pero en los últimos años esta volviendo a tomar gran relevancia.

Actualmente la AS es reconocida ampliamente como uno de los factores de éxito cuando se utiliza en familias de productos, reutilización de componentes software y evolución de aplicaciones de gran escala.

Por ello, se ha planteado una aproximación que capture a través de un conjunto de modelos UML las diferentes vistas de arquitectura en las que se compondría una aplicación Web. WebSA se basa en la definición de una arquitectura software que sea específica para las aplicaciones Web, utilizando una aproximación del tipo DSSA [25]centrada en el estudio del dominio de las aplicaciones Web para restringir el espacio del problema y así reducir el espacio de la solución obtenida, en este caso la aplicación Web.

A continuación, se presenta la definición precisa que WebSA entiende por arquitectura del software.

3.3.1 Definición de la Arquitectura del software.

Existen múltiples definiciones de AS, el Software Engineering Institute's (SEI) [23] recoge 75 definiciones distintas del término Arquitectura del Software. Por este motivo, se hace imprescindible elegir la definición que se ajuste más a la acepción que toma esta propuesta y especializarla en el dominio de las aplicaciones Web.

De todas las existentes WebSA se centra en dos definiciones que más han influido en sus modelos:

Definición de IEEE Architecture Working Group [113]:

[25] DSSA: Domain-Specific Software Architecture (acrónimo ingles de Arquitectura del software de un dominio específico)

"La arquitectura es la organización fundamental de un sistema expresado mediante sus componentes, las relaciones entre cada uno de ellos y con su entorno, y los principios que guían su diseño y su evolución"

Se trata de una definición consensuada por un organismo internacional, por ese motivo es un intento de simplificación y unificación de las definiciones existentes por los diferentes expertos en la materia. Lo primero que destaca de esta definición, es que establece al componente como la unidad arquitectónica utilizada a la hora de representar la arquitectura de la aplicación. El componente es un concepto demasiado ambiguo y que en ocasiones puede hacer referencia desde un subsistema hasta un componente cliente.

Además, es importante resaltar el hecho de que la definición señala a la AS como responsable de establecer los mecanismos para guiar el diseño y su evolución, ya sea mediante la captura de los requisitos no funcionales o mediante el uso de patrones. De esta manera, Booch [15] señala la importancia que tiene la AS en la evolución de un sistema, indicando que la presencia de una arquitectura estable en un sistema, asegura las bases sobre las cuáles un sistema puede evolucionar continuamente con los mínimos desajustes y trabajo.

Sin embargo, buscando una definición que fuera menos ambigua y aportara un poco más contenido a la hora de modelar la arquitectura, WebSA se basa también en la siguiente:

Definición de Buschmann [17]:
"*La arquitectura es la descripción de los subsistemas y componentes de un sistema software y las relaciones entre ellos, típicamente representado mediante vistas que muestran las propiedades funcionales y no funcionales más relevantes*".

En esta definición de AS, aparecen los subsistemas junto con los componentes como unidades arquitectónicas. WebSA se basa en esta aproximación capturando tanto los subsistemas en el modelo de subsistemas (ver cap. 5), como los componentes en los modelos de integración (ver cap. 7) y de configuración (ver cap. 6), a la hora de representar la arquitectura Web.

Otro aspecto a destacar de la definición, es indicar que la AS debe representar los aspectos "más relevantes" del sistema, abstrayéndose de cierta información menos importante (de otra manera no se estaría representando la arquitectura, sino que se mostraría todo el sistema). Siendo así la base para proveer suficiente información para el análisis, ser ayuda en la toma de decisiones, y por lo tanto reducción de riesgos.

Hay que recordar que la arquitectura no es solamente estructural, (en ocasiones es entendida erróneamente como la estructura de la aplicación), sino que provee un mecanismo natural de integrar varias vistas del sistema. Aquí aparecen el concepto de vista y perspectiva de un sistema, que en el caso de una aplicación Web, WebSA ha establecido en 8 diferentes vistas. Estas vistas pueden ser tanto estructurales, funcionales y de comportamiento. Como se muestra en la sección 3.4 de Ingeniería Web, en WebSA se utilizan vistas funcionales definidas por otras aproximaciones de Ingeniería Web que son representadas de forma concurrente a la arquitectura propuesta.

Una arquitectura software esta presente durante todo el ciclo de vida, desde la concepción inicial a través del desarrollo, despliegue, uso, evolución reutilización hasta el eventual desmantelamiento del sistema. La arquitectura siempre existe, incluso si es trivial o caótica. Y abarca tanto su definición a alto nivel como a nivel de código.

La arquitectura se puede considerar como una plantilla para las familias de productos, una base para la reutilización y la comunicación con sistemas legados, y necesaria para una correcta evolución de un sistema.

3.3.2 Conceptos Principales de la Arquitectura del Software

Una vez conocida la semántica que tiene AS para WebSA, a continuación se profundiza en los principales conceptos que la constituyen y que van a ser necesarios para poder comprender nuestra aproximación.

3.3.2.1 Unidad Arquitectónica

En la mayoría de las ocasiones, la unidad arquitectónica aparece con el nombre de "componente". Se trata de una unidad autocontenida, en el sentido de que se trata de una entidad para todos los propósitos y actividades de la Ingeniería del Software. Una unidad arquitectónica aparece siempre rigurosamente encapsulada tanto para el diseño y como el análisis. Este punto es particularmente importante, con vistas al problema de la explosión de espacio. Para permitir un adecuado modelado del las entidades del dominio del problema, una unidad arquitectónica debe ser una entidad también lo suficientemente grande para que sean escalables y manejables los modelos de arquitectura.

WebSA define al Subsistema Web y al componente Web como elementos en el modelado de la arquitectura de una aplicación Web. Inicialmente en la fase de análisis, el modelo de subsistemas (ver capítulo 5) define un estilo arquitectónico para la definición de las diferentes capas que constituyen la aplicación Web, utilizando el Subsistema Web como unidad arquitectónica. Simultáneamente en la fase de análisis, el modelo de configuración define un estilo arquitectónico donde el componente Web es una entidad que representa una abstracción de uno o más componentes que comparten la misma funcionalidad en una aplicación Web (ver capítulo 6). En la fase de diseño, es el modelo de integración el que de forma independiente a la plataforma destino utiliza el componente Web como unidad arquitectónica. Este componente representa a un componente software de la aplicación, aportando las características propias de la arquitectura Web junto a las características funcionales del dominio del problema.

3.3.2.2 Vista y Perspectiva

Dos conceptos muy importantes a la hora de entender y representar la AS de una aplicación son la vista y la perspectiva de un sistema. Basándose en las definiciones aportadas por the IEEE[26] Architecture Working Group [113], la perspectiva para WebSA es "una especificación de los convenios necesarios para construir y usar una

[26] IEEE: The Institute of Electrical and Electronics Engineers (acrónimo inglés del Instituto de Ingenieros Eléctricos y Electrónicos)

vista", siendo una vista "una representación del sistema basada en un conjunto de artefactos relacionados".
Definir las diferentes vistas y las perspectivas de un sistema tiene tres ventajas principales: (1) Las vistas y perspectivas promueven la separación de aspectos mejorando así la eficiencia del proceso de desarrollo. (2) Las inconsistencias pueden surgir de tener vistas diferentes del mismo sistema. Muchas de esas inconsistencias pueden ser detectadas automáticamente, permitiendo la posibilidad de implementar una herramienta con soporte a la depuración del diseño. Esto incrementa la validación de los modelos y puede ahorrar gran cantidad de trabajo en fases posteriores. (3) El uso de vistas y perspectivas para una catalogación y recuperación automática de software desde bases de datos de componentes.
Existen trabajos previos que han abordado con éxito la especificación de la arquitectura de una aplicación tratando por separado las vistas que la constituyen, los más importantes son los de Kruchten [64] el cual define 4+1 vistas concurrentes para definir un sistema, y posteriormente Hofmeister [46] que separa en 4 vistas de la arquitectura software de una aplicación. Sin embargo, ninguno de los trabajos ha abordado dicha separación de vistas para el dominio de las aplicaciones Web. Por ello, WebSA ha definido un modelo de vistas (ver sección 3.4) en el que se establece por un lado la perspectiva funcional junto con las vistas propias de las metodologías de Ingeniería Web, como son navegación, presentación, proceso y conceptual. Además, a este modelo le añade la perspectiva de arquitectura en el que se definen las vistas de arquitectura lógica que va a contener los modelos de componentes definidos en el presente trabajo y la vista de arquitectura física.

3.3.2.3 Estilo Arquitectónico

Existe una gran variedad de opiniones al respecto al significado de estilo arquitectónico. Una concisa definición proporcionada por Shaw & Clements [109], define un estilo arquitectónico como "*un conjunto de reglas de diseño que identifica los tipos de componentes y conectores que pueden ser usados para componer un sistema junto con las restricciones en el modo en que la composición es hecha*". Una similar posición es expresada por Medvidovic [69]. De manera que un estilo es entendido como un vocabulario para expresar arquitecturas. Basándonos en estas definiciones queda claro, que se establece un vínculo entre el estilo arquitectónico y la unidad arquitectónica.

En WebSA se establece en la fase de análisis dos estilos arquitectónicos que representan el sistema basándose en diferentes unidades arquitectónicas y relaciones. Por un lado, el modelo de subsistemas sigue el estilo de capas Buchmann et al. [17], donde es el subsistema la unidad arquitectónica, y las relaciones de dependencia son los conectores. De forma paralela se define el modelo de configuración que sigue un estilo arquitectónico, donde es el componente Web la unidad arquitectónica, y en este caso son los conectores mediante puertos e interfaces el pegamento entre cada uno de los componentes.

3.3.2.4 Abstracción

El concepto de abstracción tiene muchas acepciones con un núcleo de significados común. Las diferencias en el uso del concepto de abstracción ayudan también a

identificar las diversas corrientes en el seno de la AS. La definición que proporciona Grady Booch [16], por ejemplo, revela que el autor identifica la abstracción arquitectónica con el encapsulamiento propio de la tecnología de objetos: "*Una abstracción* –escribe Booch– *denota las características esenciales de un objeto que lo distinguen de otras clases de objeto y provee de este modo delimitaciones conceptuales bien definidas, relativas a la perspectiva del observador*". Tanto para la IEEE como para Rumbaugh, Shaw y otros autores, la abstracción consiste en extraer las propiedades esenciales, o identificar los aspectos importantes, o examinar selectivamente ciertos aspectos de un problema, posponiendo o ignorando los detalles menos sustanciales o irrelevantes.

La idea de abstracción forma parte de lo que acaso sea la pieza conceptual más importante de la AS, el concepto de estilo arquitectónico; un estilo se identifica a grandes rasgos o como se dice habitualmente, en un estilo "menos es más". Clements & Northrop [21] sostienen que el trabajo de Garlan y Shaw sobre los estilos arquitectónicos enseña que aunque los programas pueden combinarse de maneras prácticamente infinitas, hay mucho que ganar si se restringe a un conjunto relativamente pequeño de elecciones cuando se trata de cooperación e interacción. Las ventajas incluyen mejor reutilización, mejores análisis, menor tiempo se selección y mayor interoperabilidad. Conceptos como el de estilo o marcos revelan su naturaleza arquitectónica en su abstracción y en su generalidad.

Siguiendo la idea de abstracción, WebSA define la arquitectura a diferentes niveles de abstracción. Comenzando por el modelo de subsistemas, que puede considerarse el más abstracto al tratarse de la unidad arquitectónica más genérica, pasando por el modelo de configuración que define el concepto de componente Web que encapsula la tarea o conjunto de componentes que tienen la misma tarea dentro de una aplicación Web, y el modelo de integración, donde el componente Web representa realmente a un componente software de la aplicación pero de una forma independiente de plataforma.

3.3.2.5 Proceso Arquitectónico

De acuerdo con Sommerville [110], el proceso de desarrollo del software "*es un conjunto de actividades y resultados asociados que producen un producto software*". Está ampliamente aceptado por la comunidad científica que un proceso de desarrollo ordenado tiene una enorme importancia en la calidad y el control de la creación y el mantenimiento del software. Existen actualmente varios estándares para la mejora del proceso software (el Modelo de capacidad de madurez (CMM) CMU SEI CMM [22]), y los métodos industriales como Catálisis (Cook & Daniels [25]) y proceso estándar unificado (UP) Jacobson et al. [50].

Dentro de los diferentes tipos de procesos de software, WebSA define un proceso de desarrollo instancia del proceso estándar unificado (UP) [50] situándose en aquellos que son denominados "centrados en la arquitectura" o proceso arquitectónico, es decir, aquel proceso donde "la arquitectura del sistema es usada como artefacto principal para conceptuar, construir, manejar y evolucionar el sistema en desarrollo". De este modo, en un proceso arquitectónico, la arquitectura debe ser creada y usada con la debida diligencia, es decir, las actividades involucradas necesitan estar bien definidas y descritas de forma precisa. Así, se debe tener una adecuada noción de arquitectura en primer lugar; cuando se define la arquitectura

dentro del proceso, cuáles son las perspectivas que contiene; si trata con arquitecturas, estilos y líneas de productos; si está estructurada en patrones, etc.

En WebSA la AS es definida desde el flujo de trabajo de análisis, donde serán los estilos de arquitectura quienes dirigirán las transformaciones para obtener el diseño arquitectónico establecido por el modelo de integración, el cual finalmente fijará cual va a ser la implementación final.

Una vez definido los conceptos principales que definen el concepto de AS, se presenta el tipo de aproximación AS utilizado para representar una aplicación Web.

3.3.3 Mecanismo de Reutilización: Los Patrones

Los patrones dotan a WebSA principalmente de un mecanismo de reutilización en los modelos de arquitectura. A diferencia de otras aproximaciones donde los patrones son utilizados únicamente en la fase de diseño y de implementación. En WebSA, la reutilización de patrones se establece desde el análisis, en la que se define el análisis arquitectural mediante estilos arquitectónicos propuestos por el modelo de subsistema (ver capítulo 5) y el modelo de configuración (ver capítulo 6). Los patrones definen así, las mejores prácticas a la hora de proponer la arquitectura, guiando al arquitecto y acelerando el desarrollo de posteriores aplicaciones Web.

A continuación se especifica lo que WebSA entiende como patrón y los diferentes tipos que pueden identificarse.

3.3.3.1 Definición de Patrón y sus Tipos

Se entiende como patrón software "*un artefacto software que presenta una solución para resolver un problema recurrente que surge en un determinado contexto*", Gamma et al. [36]. Así, permite fijar la solución más adecuada a este problema. Se obtienen una serie de beneficios como: (1) documentar las experiencias probadas, (2) especificar una configuración de elementos y un comportamiento concreto para la solución, (3) proveer un vocabulario común y un concepto que ayuda al entendimiento, (4) apuntar a aspectos de calidad o requisitos no funcionales que mueven la aplicación de dicha solución.

La definición de patrón es genérica y aplicable a cualquier contexto. Sin embargo, todo el mundo acepta que existen diversas clases de patrones: de análisis, de arquitectura, de diseño, de organización o proceso, de programación y los llamados idioms, etc. Cada autor escribe sobre el asunto de los patrones agrega una clase diferente, y los estándares en vigencia no hacen ningún esfuerzo para poner un límite a la proliferación de variedades y ejemplares.

Sin embargo, WebSA se centra en aquellos patrones que puedan expresarse mediante los modelos propuestos. Por un lado, los patrones de arquitectura y diseño de carácter general, principalmente los patrones de arquitectura definidos por Buchmann et al. [17] y de diseño de Gamma et al. [36]. Y por otro lado, aquellos patrones que están más centrados en el dominio de las aplicaciones Web como Conallen [24].

A continuación, se define lo que se entiende exactamente por cada uno de estos tipos de patrones.

3.3.3.2 Definición de Patrón de Arquitectura

Por patrones de arquitectura Buchmann et al. [17] define que: "*Un patrón de arquitectura expresa el esquema de organización de la estructura fundamental para los sistemas software. El patrón provee de un conjunto predefinido de subsistemas, especifica sus responsabilidades e incluye reglas y guías para la organización de las relaciones entre ellos*".
La principal razón por la cual WebSA se centra en esta definición de patrón de arquitectura, es porque establece una clara relación entre el estilo arquitectónico y el patrón. Según Buchmann et al. [17] "*Los estilos se aplican en la fase de análisis arquitectónico en términos de patrones de arquitectura*". Esto permite que a partir de los estilos arquitectónicos propuestos por WebSA se puedan realizar representaciones de los diferentes patrones y vincular así en la fase de análisis de arquitectura estos modelos, con los requisitos no funcionales que están vinculados con cada uno de los patrones.

Dentro de los patrones de arquitectura destacan aquellos que han sido definidos dentro del dominio de las aplicaciones Web. En concreto, los patrones de arquitectura definidos por Conallen [24]. Estos patrones representan como participantes, a los componentes Web que interactúan para definir su arquitectura. Un ejemplo de patrón es el Template Page (Pagina Plantilla) que establece el componente "Pagina Servidora" como elemento que sirve como plantilla principal para componer las diferentes paginas de la aplicación, a partir de las referencias dinámicas que dicha plantilla realiza sobre otras instancias "Paginas Servidoras".

3.3.3.3 Definición de Patrón de Diseño

A la hora de definir el concepto de patrón de diseño, WebSA se basa en los autores del catálogo más reconocido Gamma et al. [36]. Según ellos: "*Un patrón de diseño proporciona un esquema para refinar los subsistemas o componentes de un sistema software, o las relaciones entre ellos. Describe una estructura recurrente de comunicación entre componentes que resuelve un problema general de diseño en un contexto particular*". Estos patrones de diseño se concentran en la estructuras de más bajo nivel del sistema que los patrones de arquitectura propuestos por Buchmann et al. [17]. Esto implica que en determinadas ocasiones, un patrón de arquitectura pueda contener a un patrón de diseño.

Los patrones de diseño de software propuestos por Gamma et al. [36] buscan codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. En este aspecto comparte propiedades similares a los patrones de arquitectura, sin embargo, a diferencia de los anteriores, en muchas ocasiones los patrones de diseño son soluciones más concretas, que pueden estar o no representados mediante los estilos arquitectónicos. En concreto, en WebSA se representan un conjunto de patrones de diseño, que pueden ser representados mediante el modelo de configuración. Esto se debe, a que el nivel de granularidad del modelo de configuración son los componentes y en ellos se pueden aplicar determinados patrones como Façade, DAO (Objeto de acceso a datos), etc.

3.3.4 Los Patrones en WebSA

Dentro de nuestra aproximación los patrones de arquitectura y de diseño han sido aplicados en la fase de análisis de arquitectura a través de los estilos arquitectónicos

definidos para la representación de las aplicaciones Web. Estos modelos permiten los mecanismos adecuados para su representación y su reutilización. A continuación se describe un conjunto de patrones que pueden representarse dentro de los diferentes modelos de arquitectura de WebSA.

3.3.4.1 Patrones de Arquitectura

Comienza la descripción con los patrones de arquitectura de mayor granularidad, en este caso los llamados patrones de distribución, definidos por Renzel & Keller [100]. Un patrón de distribución consiste en un conjunto de patrones donde cada uno realiza una descomposición del sistema en dos subsistemas, con los roles de cliente y servidor. La aplicación de cada uno de los patrones, tiene como consecuencia una serie de ventajas, como una separación de tareas que ayudan a la reutilización, mayor mantenibilidad, rendimiento, etc. Sin embargo dicha descomposición, trae consigo un conjunto de problemas como la mayor complejidad, mayor retardo de comunicación, etc. La aplicación de cada uno de estos patrones tiene asociado un conjunto asociado de fuerzas que se corresponden a requisitos no funcionales propuestos por el cliente para la aplicación Web. A partir de la prioridad de los requisitos, se decide aplicar o no cada uno de los patrones.

Los patrones de distribución son explicados dentro del método de aplicación del modelo de subsistemas descrito en el capítulo 5. Se define un proceso de descomposición que se basa en la aplicación o no de cada uno de los patrones de distribución en función la existencia o no de las fuerzas en la aplicación Web. Los patrones de distribución (Renzel y Keller [100]) aplicados son los siguientes:

- Presentación distribuida.
- Interfaz Remoto
- Base de datos Remota
- Núcleo aplicación distribuida
- Base de datos distribuida.

El siguiente grupo de patrones de arquitectura, son los definidos por el otro estilo arquitectónico de WebSA, es decir, el modelo de configuración. Este modelo permite definir patrones de arquitectura que han sido propuestos por diversos autores Buchmann et al. [17], Conallen [24], Trowbridge & Mancini [118]. En este caso, la forma de representación de un patrón se realiza mediante una determinada configuración de componentes que se aplica para resolver un determinado problema en cualquiera de las partes de la aplicación.

A continuación se enumeran los patrones representados por diferentes autores y son separados en función de las capas en las que son aplicados:

- **Master-View-Controller (Modelo-Vista-Controlador)** (Buchmann et al. [17]): patrón que permite separar el modelado del dominio, la presentación y las acciones basadas en las interacciones del usuario en 3 componentes principales. (1) Modelo: que maneja el comportamiento y los datos del dominio de la aplicación, respondiendo a las peticiones de información sobre su estado (usualmente desde la vista) (2) Vista: muestra la información al usuario final, (3) Controlador: recibe las peticiones del usuario, informando al modelo y/o la vista para cambiar. Existen diferentes variantes que pueden introducir cambios de comportamiento

para los componentes. Se puede encontrar su representación con el modelo de configuración en el capítulo 6.

- **Page Template (Página Plantilla)** (Conallen [24]): Este patrón define una única página plantilla que genera todas las páginas Web que obtiene el cliente. Su característica especial es que la página plantilla referencia a cada una de las partes o fragmentos de página que contiene dinámicamente. Además, la configuración de la plantilla puede estar almacenada en un fichero independiente, lo que permite gestionar la apariencia sin recompilar el código.
- **Page Controller** (Trowbridge & Mancini [118]): Es una especialización del MVC, que establece un controlador común llamado BaseController que contiene todos los componentes o partes de la página Web que son comunes al resto de controladores de cada una de las páginas. Consigue reutilizar aquellos componentes o acciones que se van a repetir en múltiples controladores de página.
- **Front Controller** (Trowbridge & Mancini [118]): Otra versión del MVC, propone que se establezca un único controlador que centralice todas las peticiones, y que esté separado en dos aspectos: un manejador de las peticiones del cliente y disparador de las peticiones. Este patrón de arquitectura hace uso del patrón de diseño Command (Gamma et al.[36]) para gestionar las peticiones que el controlador recibe de la interfaz.

Por otro lado, debido a que el estilo arquitectónico permite modelar estos patrones, esto proporciona al arquitecto introducir pequeñas variantes a cada uno de ellos, sin perder la finalidad última del patrón. Por ejemplo, la evolución del MVC en MVC2, donde se independiza la navegación de la vista, mejorando la independencia de la vista con el controlador.

3.3.4.2 Patrones de Diseño

Los patrones de diseño típicamente representan micro-arquitecturas que pueden ser aplicadas independientemente del dominio del problema. Por ello, la definición de los patrones de diseño en WebSA, puede ser realizada en el modelo de configuración (ver capítulo 6) que propone un diseño inicial en el análisis arquitectónico. Una vez definido el patrón, en el modelo de integración (ver capítulo 7) los patrones se integran con la funcionalidad y pueden plasmarse en un diseño independiente de la plataforma. Se aborda la especificación de los diferentes patrones de diseño en el modelo de configuración.

En algunas ocasiones se representa un patrón mediante uno o dos componentes dentro del modelo que define la vista general de la arquitectura de la aplicación Web. En otras, se necesita definir un elemento PatronWeb (ver capítulo 6), que permite establecer una configuración formada por más de un componente en un modelo independiente y posteriormente aplicarlo a cualquier modelo donde se quiera instanciar.

A continuación se muestra un conjunto de patrones de diseño que pueden representarse con WebSA, junto a una breve descripción de la solución que aportan:

- Distributed Facade (Gamma et al. [36]): Permite establecer una interfaz más reducida y escalable por parte de la lógica de negocio al interfaz de

usuario. Permite así, reducir la comunicación y la dependencia entre el cliente y el servidor.

- Data Access Component (Gamma et al. [36]): Se define el componente de acceso a datos que realiza la tarea de separar la lógica de negocio del acceso a datos.
- Data Transfer Object (Fowler [32]): establece una interfaz de grano grueso entre componentes distribuidos que permite el paso de objetos completos por valor, reduciendo así el número de invocaciones remotas.
- Broker (Buchmann et al. [17]): Se define un componente que permita realizar la tarea de separar los detalles de invocación de un servicio remoto.
- Coarse-Grained Interface (Interfaces de grano grueso) (Trowbridge & Mancini [118]): Intenta definir interfaces remotos que oferten pocos servicios autocontenidos, requiriendo así menos llamadas remotas y aumentando el rendimiento.
- Service Gateway (Trowbridge & Manzini [118]): Se utiliza en el caso de que nuestra aplicación tenga la necesidad de conectarse con otras aplicaciones que le implementan parte de su funcionalidad. El patrón propone la encapsulación del código cliente al sistema legado.
- Service Interface (Trowbridge & Manzini [118]): provee un punto de entrada a los clientes que acceden a la funcionalidad expuesta por la aplicación. Implementado normalmente por la vista legada (ver capítulo 6).

Una vez definidos los patrones de diseño, recordar que en WebSA el modelo de configuración permite representar estos patrones de diseño, y además modificarlos o definir patrones nuevos.

La siguiente sección se centra en la familia de aplicaciones sobre la que WebSA realiza los esfuerzos de desarrollo y representación, es decir, las aplicaciones Web.

3.4 La Ingeniería Web

La última y no menos importante disciplina en la que WebSA ha obtenido conocimiento e incorporado su aportación científica es la Ingeniería Web. Como definió Murugesan et al. [82] "La Ingeniería Web es el establecimiento, uso de la ingeniería, disciplina y aproximación sistemática para el desarrollo con éxito, de la creación, despliegue y mantenimiento de las aplicaciones Web de alta calidad". En este sentido, esta disciplina tiene como objeto el estudio de una familia de aplicaciones software que presenta unas particularidades especiales, como son las aplicaciones Web.

Es difícil encontrar una definición concreta para establecer lo que la comunidad científica entiende por aplicación Web. No solamente por la gran variedad de aplicaciones que aparecen dentro de Internet que se salen del concepto entendido por aplicación Web (por ejemplo, un juego java en Internet no se considera una aplicación Web). Sino porque este tipo de aplicaciones está en constante evolución y a lo largo los años de vida de Internet han ido introduciéndose nuevas arquitecturas y funcionalidades.

Steve Jobs [53] observó que las aplicaciones Web se están moviendo a través de diferentes estadios de evolución. En el nivel cero las aplicaciones Web representaban su estado más primitivo, donde muchos sitios estaban centrados únicamente en el documento presentado y eran esencialmente estáticos. Con la llegada de los applets Java, el mundo de las aplicaciones Web se movió al estado -1, es decir, dio un paso atrás, puesto que muchos proyectos usaron esas tecnologías que les llevaron al fracaso, ya que estaban desvinculadas del navegador y eran muy lentas de descargar en el cliente. Sin embargo, otras organizaciones pasaron rápidamente del nivel 1 al 2, donde el cliente y el servidor se unieron para definir las páginas dinámicas. Esto implicaba realizar mecanismos de enlace ejecutados en el servidor entre la interfaz de usuario y las bases de datos, haciendo accesible los datos dinámicamente a través de un navegador. Hoy en día, muchos sitios ya han llegado al nivel tres, tienen más semántica asociada a los datos (usando muchas veces XML) y con una explotación mayor del potencial de la computación distribuida, es decir, aparecen servidores de aplicaciones para aplicaciones Web, que no solo permiten introducir componentes de la parte cliente, sino que pueden utilizar componentes de servidor distribuidos que corren a través de un broker. Por último, la fase cuatro, aun no alcanzada llegará con la maduración de la tecnología de agentes y la computación pervasiva.

Por lo tanto, y a pesar de la constante evolución que tienen las aplicaciones Web. Se debe hacer un esfuerzo de abstracción y fijar cual es el tipo de programa software que WebSA entiende por aplicación Web. Siguiendo la definición de Cachero [19] una aplicación Web es *"un sistema de información donde existen una gran cantidad de datos volátiles, altamente estructurados, consultados, procesados y actualizados mediante navegadores"*. Destaca además, como características principales de las aplicaciones Web: (1) su alto grado de interacción con el usuario. (2) La complejidad de la funcionalidad ofertada por dichas aplicaciones es variable, puede ir desde la más básica navegación/consulta, altas, bajas, modificaciones, hasta los complejos servicios transaccionales como los encontrados tras las aplicaciones de comercio electrónico. (3) El diseño de interfaz está condicionado por las necesidades de claridad y simplicidad más que por un diseño impactante. Esta interfaz debe estar estructurada para que se oriente a cada tipo de usuario en función de sus necesidades particulares de información/funcionalidad a través del espacio de información.

Una vez identificada la familia de aplicaciones sobre la cual WebSA ha centrado el esfuerzo de definición de los modelos de arquitectura y establecido su proceso. A continuación, se describen cuáles son las diferentes vistas identificadas dentro de una aplicación Web. Esto permite fijar cuáles son los modelos y aspectos que se tratan dentro de la aproximación WebSA y cuáles son obtenidos por otras aproximaciones.

3.4.1 Modelo de Vistas de una Aplicación Web

Hasta la fecha, las diferentes aproximaciones propuestas dentro de la Ingeniería Web, se han preocupado principalmente en la representación de las diferentes vistas funcionales, que han sido definidas por autores como Retschitzegger & Schwinger [101]. Estos autores establecen que existen 3 niveles diferentes dentro de una aplicación Web: contenido, navegación y presentación. Sin embargo, como se especifica en la sección 3.2, WebSA considera necesario representar la arquitectura del software de una aplicación Web, permitiendo así representar una gran variedad de tipos de aplicaciones Web y poder recoger los aspectos no funcionales presentes en

ellos. Por lo tanto, con el objetivo de establecer de forma clara los aspectos a cubrir dentro de una aplicación Web, WebSA (Meliá et al. [72]) propone un modelo de vistas compuesto de 8 vistas diferentes. Como se observa en la Figura. 4, las vistas están agrupadas en tres perspectivas (ambos conceptos especificados en la sección 3.1.1.2). El modelo de vistas establece en la parte superior la perspectiva de requisitos. Esta perspectiva obtiene las características o requisitos establecidos por el cliente del sistema. Se divide a su vez en dos vistas que representan la vista de requisitos funcionales y la vista de requisitos no funcionales. En la vista de requisitos funcionales para las aplicaciones Web, existen trabajos como NDT [29] o OOWS [122] que han definido un conjunto de modelos que recogen los requisitos funcionales mediante modelos basados en tareas y han utilizado MDA para definir un mapeo que permite obtener los modelos de la perspectiva funcional.

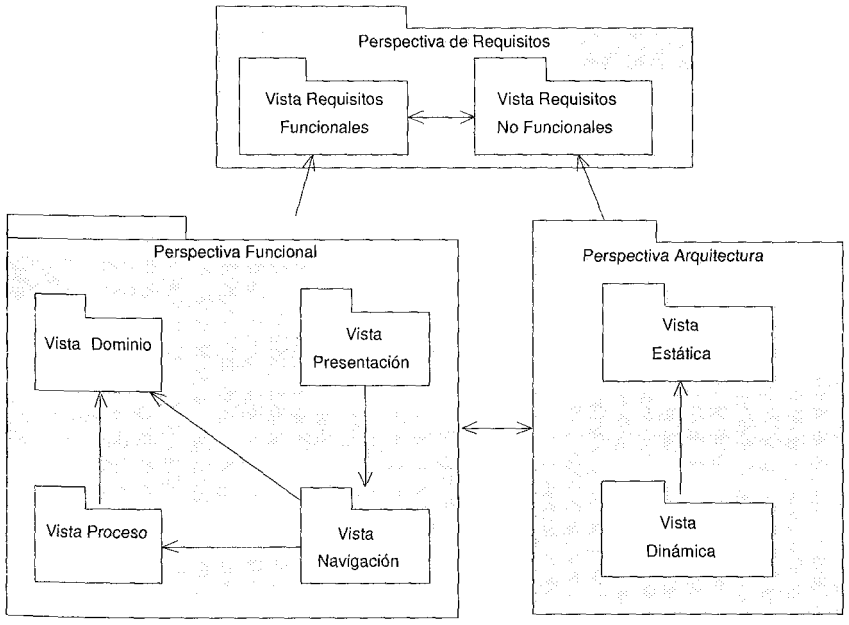

Figura. 4. El Modelo de Vistas de WebSA.

En la parte no funcional existen trabajos como List & Korherr [67] que tratan los requisitos no funcionales a partir de los requisitos de negocio. Sin embargo, a día de hoy no se ha realizado un estudio para establecer las transformaciones desde los requisitos hasta los modelos de arquitectura.

Siguiendo la descripción, la perspectiva funcional es donde mayor número de trabajos han existido tradicionalmente y constituyen el núcleo tradicional de la Ingeniería Web. Se encuentran metodologías como OO-H [19], UWE [59], WebML [20], OOHDM [105], WSDM [119], W2000 [8], etc. que han definido todas o casi todas las vistas definidas en la perspectiva funcional de la Figura. 4. Empezando por el modelo de dominio, llamado también modelo conceptual o de estructura, que

representa las entidades del dominio de la aplicación. Tradicionalmente, este modelo se ha representado mediante diagramas de clases por aproximaciones orientadas a objeto como OOH o UWE. Sin embargo, existen otras aproximaciones orientadas a datos que representan el modelo mediante diagramas entidad-relacion (OOHDM, WebML). Sin embargo, a pesar de que esta notación no es la misma, los conceptos son perfectamente mapeables de una aproximación a otra.

A partir de la vista de dominio, se establecen otras vistas funcionales, como la vista de navegación y la vista de proceso. La vista de navegación, se puede considerar junto a la vista de dominio, como un común denominador de las metodologías de diseño Web. Esta vista proporciona la posibilidad de especificar los enlaces navegacionales que existen entre los diferentes nodos o elementos de interfaz de usuario. Dicha vista aparece como particular del dominio de la aplicación Web, debido a que ha sido en estas aplicaciones donde surge la necesidad de especificar claramente cual es la navegación que existe entre las diferentes páginas Web. Otras metodologías llegadas a este punto se han puesto de acuerdo semánticamente en la mayoría de los casos en los conceptos recogidos, sin embargo, han diferenciado sus aproximaciones siguiendo notaciones diferentes. En algunos casos dichas notaciones son propietarias como OOH, WebML, OOHDM, y en otras se han basado en un estándar UML como UWE.

Junto a la vista de navegación, se tiene la vista de proceso que ha sido introducida por diversas aproximaciones Web como un añadido a los modelos de dominio y navegación ya definidos. Y además, se han seguido dos líneas diferentes en lo que el modelo de proceso ha representado. Mientras por un lado, en WebML y OOHDM los modelos de proceso sirven para definir flujos de trabajos dentro de aplicaciones Web colaborativas. Por otro lado, en UWE y OOH, el modelo de proceso tiene como objetivo principalmente conectar la lógica de negocio con los modelos de dominio y navegación. Una descripción detallada de estos métodos puede verse en Meliá & Gomez [75].

La última vista que queda por definir dentro de la perspectiva funcional, es la vista de presentación que proporciona los elementos que permiten describir la interfaz de usuario. Para este modelo tampoco hay un consenso a la hora de elegir cuáles son los modelos para representar dicha vista. Por ejemplo, OOH lo denomina Diagrama de Presentación Abstracta (DPA) y es obtenido por un proceso de generación a partir del modelo de navegación, y representa cada instancia de página Web de forma abstracta y las relaciones entre dichas páginas. Por otro lado, UWE en su modelo de presentación representa mediante una notación de composición del diagrama de clases, los diferentes conceptos que componen la interfaz.

Por último y no por ello menos importante, se describe la perspectiva de arquitectura que constituye un punto de vista complementario a las aproximaciones planteadas hasta la fecha. Solo aproximaciones como Meinecke et al. [70] se han planteado su representación en las aplicaciones Web. Como se ha especificado en la sección 3.2, la arquitectura describe el sistema software mediante los subsistemas, componentes y las relaciones entre ellos. Como muestra la Figura. 4, el modelo de vistas de WebSA hace una división en 2 vistas diferentes dentro de la perspectiva de arquitectura. Por un lado, la vista estática en la que se establecen cuáles son los diferentes subsistemas y componentes que constituyen nuestro sistema y sus relaciones. Por otro lado, se tiene la vista dinámica, establecida una vez los componentes estáticos han sido definidos y mediante la cual se establecen cuáles son

las secuencias de llamadas realizadas entre los componentes y comportamiento interno que tiene cada uno de ellos. A partir de aquí, WebSA propone para cada una de las vistas un conjunto de modelos situados en diferentes niveles de abstracción. Dentro de la vista estática, WebSA ha definido los siguientes modelos: el modelo de subsistema, es un estilo arquitectónico que descompone el sistema mediante un conjunto de subsistemas cada uno situado en una capa diferente (ver capítulo 5), el modelo de configuración también es un estilo que representa el sistema mediante un conjunto de componentes y conectores localizados en el dominio de las aplicaciones Web (ver capítulo 6) y el modelo de integración que representa el sistema mediante los componentes Web donde se ha introducido la información que proviene de la perspectiva funcional (ver capítulo 7). Por otro lado, se tienen los modelos definidos en la vista dinámica, que consisten en el modelo de configuración dinámica, y el modelo de integración dinámica, ambos modelos consisten en diagramas de secuencia que muestran la interacción existente entre los componentes definidos en los modelos definidos en la vista estática. Estos modelos han quedado fuera del ámbito de la presente tesis.

Por último, es importante resaltar la dependencia existente entre las perspectiva funcional y de arquitectura. A priori, la inserción de elementos funcionales dentro de la vista de arquitectura es muy habitual, ya que los componentes definidos en esta vista, se completan en la fase de diseño de aspectos funcionales como son entidades de dominio, atributos, operaciones, etc. que son proporcionados por la parte funcional. Sin embargo, es más difícil encontrarnos con el caso en el que la vista funcional se vea influenciada por la definición arquitectónica de la aplicación. Un ejemplo, podría ser aquella funcionalidad que por motivos estratégicos es eliminada del sistema, ya que va a ser proporcionada por un sistema legado. En esa ocasión, podría suponer una modificación de la parte funcional forzada por la parte de arquitectura.

Una vez presentadas las perspectivas que son relevantes para WebSA, se trata la integración propuesta por WebSA para unir la vista funcional y la vista de arquitectura.

3.4.2 Arquitectura Especifica para las Aplicaciones Web

Debido al cada vez mayor protagonismo de Internet y de los medios hipermediales, se acentúa la necesidad de desarrollar aplicaciones Web que ofrezcan un mayor número servicios y de mayor calidad o satisfacción para el cliente final. Por lo tanto, existe la necesidad de desarrollar aplicaciones Web cada vez más complejas que deben cumplir con una serie de requisitos tanto funcionales como no funcionales según Boehm [14] son el rendimiento, la usabilidad, la escalabilidad, el mantenimiento, etc. para poder satisfacer a unos usuarios cada vez más exigentes. Por otro lado, existe una gran variedad de aplicaciones Web, que van desde aplicaciones en Intranet sencillas que requieren poca escalabilidad y rendimiento, hasta las aplicaciones Web de comercio electrónico con un alto grado de escalabilidad, mantenimiento, etc. Esto hace muy conveniente que su desarrollo capture los aspectos arquitectónicos y los incorpore en el resto de las fases del ciclo de vida.

Sin embargo, salvo excepciones como Conallen [24], Hassan & Holt [42], OOHDM-Java [51] las metodologías se preocupan principalmente en capturar los aspectos funcionales de una aplicación y olvidan los requisitos de calidad o no

funcionales. Como indica Bass et al. [10], los aspectos no funcionales son expresados mediante AS y son capaces de completar las necesidades que el usuario requiere en dicha aplicación. Poniendo un símil con otro tipo de obras de ingeniería, sería impensable que ningún ingeniero civil abordara la construcción de un puente sin haber realizado previamente un estudio arquitectónico. Sin embargo, hoy en día, todavía hay muchas organizaciones que abordan la realización de sus aplicaciones Web sin tener en consideración su arquitectura.

Como indica Booch [15], a la hora de desarrollar una aplicación Web debido a su especial naturaleza, se han de tener en consideración ciertas características particulares para que esta tenga éxito:

1. Integración con sistemas legados
2. Una evolución continua
3. Consideración de los picos de tráfico

Además de estos tres puntos críticos a considerar, existen otros dos aspectos como son la seguridad y la presentación que tienen especial relevancia dentro de las aplicaciones Web.

Construir aplicaciones Web como por ejemplo las aplicaciones de comercio electrónico es una tarea bastante compleja desde que se necesitan unos elevados conocimientos técnicos que van desde los propios de la interfaz Web, de su parte servidora, hasta llegar a las tecnologías de bases de datos y de integración con sistemas legados. Además, requiere una adecuada coordinación entre los componentes del equipo de desarrollo debido a la necesidad de especialización entre cada una de las partes que componen una aplicación Web. En este sentido, considerar la AS en el desarrollo de la aplicación Web, permite la comunicación entre los miembros de desarrollo, y sirve para dirigir el desarrollo y determinar la capacidad para evolucionar el crecimiento del sistema.

Como ha sido indicado previamente, WebSA ha establecido tres modelos de AS, los cuáles se basan en 2 estilos arquitectónicos destinados a la representación de la arquitectura Web en diferentes niveles de abstracción. Para realizar la definición de estos estilos de arquitectura, WebSA ha seguido la filosofía establecida por una arquitectura específica de dominio (DSSA). Existen diferentes definiciones de DSSA, este trabajo se ha centrado en la siguiente: '*DSSA es un ensamblado de componentes software, especializados para un particular tipo de tarea (dominio), generalizados para un uso efectivo a través del dominio y compuesto por una estructura estandarizada (topología) efectiva para conseguir el éxito en la construcción de aplicaciones*". (Hayes & Roth [44])

Siguiendo esta filosofía WebSA se define como una DSSA enfocada en el dominio bien conocido de las aplicaciones Web. En este dominio se establecen dos topologías de componentes generalizados y abstraídos de la familia de aplicaciones Web, lo que permitirá representar la AS de una forma sencilla y efectiva. Además, centrarse en un dominio en concreto, ayudará a restringir tanto el espacio del problema como el espacio de la solución, lo que conduce en muchos casos a que el desarrollo de los diferentes sistemas software sea bastante similar. Permitiendo en muchos casos, diferentes niveles de reutilización, tanto a nivel de análisis mediante patrones de arquitectura, como a nivel de diseño mediante patrones de diseño. Como menciona [83], el desarrollo de una arquitectura DSSA se basa en el estudio de las propiedades comunes y diferencias entre las aplicaciones de un dominio.

Las propiedades más importantes que debe perseguir una arquitectura DSSA según [45] son:

- Trabajar en un espacio del problema y de la solución restringidos.
- Genérica, para poder adaptar el DSSA a un desarrollo en concreto.
- Un nivel de abstracción adecuado para abarcar todo el dominio.
- Elementos de reutilización dentro del proceso de desarrollo que son típicos de cada dominio.

Entre las propiedades anteriores destaca que al trabajar en un espacio de solución restringido facilita la tarea de identificación de la funcionalidad de ciertos componentes que en otras ocasiones sería imposible. Gracias a ello, se ha podido definir topologías de subsistemas Web dentro del modelo de subsistemas y de componentes Web dentro del modelo de configuración.

Por otro lado, es importante considerar que el nivel de abstracción debe ser el adecuado para abarcar todo el dominio. En este sentido se han establecido los modelos de arquitectura a diferentes niveles de abstracción, cada uno de los cuáles proporciona una vista diferente del dominio de la aplicación. Sin embargo, hay considerar que una arquitectura DSSA debe ser tratada de una forma evolutiva, es decir, siempre se han de buscar mejores abstracciones para todo el proceso de desarrollo. Esto se consigue trabajando con la arquitectura, ganado más experiencia y validando la existente siguiendo así un proceso de refinamiento.

3.4.3 Integración de WebSA con las Propuestas de Diseño Funcional

Como ya ha quedado reflejado en el modelo de vistas y en el capítulo 2, existen múltiples propuestas dentro de la ingeniería Web, que especifican todas o parte de las diferentes vistas que constituyen la perspectiva funcional. Por lo tanto, desde nuestra aproximación carecía de sentido presentar nuevos modelos para la parte funcional, pudiéndose valer de aquellos modelos que ya han sido definidos por otras propuestas. Por otro lado, ha quedado reflejado una carencia en la mayoría de las aproximaciones Web, a la hora de representar la perspectiva de arquitectura. Y por lo tanto, no se dispone de modelos de arquitectura a parte de los que proporciona WebSA.

A partir de aquí, nuestro siguiente objetivo es integrar WebSA con aquellas aproximaciones que estén aceptadas por la comunidad y que presenten una serie de características adecuadas y cumplan con los requisitos para realizar la integración. Así, se consigue tener una descripción completa de la aproximación Web y establecer un proceso que permita obtener la integración de ambas perspectivas en el código final.

Como se introduce en la sección 3.2 de Ingeniería dirigida por modelos, WebSA define los artefactos de su proceso de desarrollo basándose en MDA. Concretamente, se especifica un conjunto de modelos de arquitectura siguiendo el estándar UML. Por otro lado, se utiliza un conjunto de transformaciones para integrar la funcionalidad con la arquitectura, dichas transformaciones están definidas mediante UPT, que se basa en los metamodelos MOF para definirlas. Por lo tanto, siguiendo esta premisa la única restricción que necesita poder integrar una aproximación funcional con WebSA es la siguiente: "Los modelos funcionales han de estar formalizados mediante un metamodelo MOF".

No es un requisito excesivamente difícil, ya que en la mayoría de los casos es fácil obtener el metamodelo MOF de los diferentes modelos de las aproximaciones. Sin embargo, es una tarea que ha sido realizada únicamente por cuatro aproximaciones: OO-H, UWE, WebML y W2000.

Por lo tanto, nuestro trabajo se centra en OO-H y UWE como metodologías principales que van a proveer la funcionalidad de la aplicación Web. A continuación se describen cuáles son los modelos que son considerados por WebSA de cada aproximación.

De la aproximación OO-H, se consultan los elementos de los siguientes modelos:

- **Modelo de Dominio**: representa las clases entidad, sus atributos, operaciones y las relaciones entre las clases de herencia y de asociación. Este modelo está explicado con detalle en la sección 4.5.2.
- **Modelo de Navegación**: representa la navegación entre los diferentes nodos navegacionales. Tiene como peculiaridad que se trata de un modelo con una notación no UML, pero que posee el metamodelo en MOF. Este modelo está explicado con detalle en la sección 4.5.3.

De UWE se obtiene información de los siguientes modelos:

- **Modelo de Dominio:** Sigue la misma notación que el representado por OO-H, es un modelo de clases UML en el cual se representan las entidades sin ningún tipo de detalle de implementación.
- **Modelo de Navegación:** Define el modelo de navegación (navigation model) mediante un perfil del diagrama de clases, en el cual define los estereotipos para navigation class (clase navegación), link (enlace), etc. A la hora de representar su metamodelo, extiende el metamodelo de UML de forma conservadora, es decir, extendiendo las metaclases de UML y añadiéndole las propiedades a las clases heredadas. Ver más detalles en la sección 4.5.3.

3.5 Conclusiones

El presente capítulo proporciona una visión general de WebSA mostrando cuáles son las tres disciplinas o áreas de conocimiento sobre las que se fundamenta:

- El desarrollo dirigido por modelos que proporciona la base fundamental sobre la cual se definen los artefactos más importantes del desarrollo, es decir, los modelos que permiten representar el sistema, las transformaciones que permiten el establecimiento de los mapeos entre los modelos y el código para obtener la implementación final de la aplicación.
- La Arquitectura del software, en ella se basa una de las principales aportaciones de nuestra aproximación, es decir, el modelado de la arquitectura en el dominio específico de las aplicaciones Web. Dentro de la arquitectura también se han tratado los patrones de arquitectura, su definición desde el análisis, permite vincular los requisitos no funcionales con la arquitectura propuesta. Además, al ser reutilizables en diferentes aplicaciones permiten acelerar el desarrollo.
- Las Ingeniería Web. WebSA presenta cuáles son las diferentes vistas que considera dentro de una aplicación Web a partir de los trabajos existentes en

Ingeniería Web. Esto permite establecer cual es la integración con diferentes propuestas existentes como OO-H, UWE, WebML, etc.

Volver al índice/Tornar a l'índex

Volver al índice/Tornar a l'índex

PARTE II

Proceso y Modelado Dirigido por la Arquitectura del Software para Web

Esta parte se centra en la definición del proceso de desarrollo Web dirigido por los modelos de Arquitectura del software. Primero se introduce el proceso de desarrollo y los artefactos que lo componen, es decir, modelos y transformaciones. A continuación, los siguientes capítulos profundizan en los modelos definidos por WebSA para representar la Arquitectura del Software. Cada uno de los modelos ha sido formalizado mediante la definición de sus elementos, su correspondiente metamodelo MOF y una notación basada en un perfil UML 2.0.

CAPÍTULO 4

El Proceso de Desarrollo de WebSA

"*La informática es la ciencia de la abstracción – se crea el modelo correcto para un problema y se inventa las técnicas mecanizadas apropiadas para resolverlo*".

[Aho and Ullman]

4.1 Motivación

Grady Booch [15] reconoce que la propia singularidad de las aplicaciones Web hace imprescindible un tratamiento diferente al resto de aplicaciones software. Siguiendo esta premisa WebSA define un proceso de desarrollo específico para las aplicaciones Web que incluye (1) artefactos específicos para su desarrollo, (2) introduce mecanismos para acelerar la puesta en el mercado de las aplicaciones Web y (3) facilidades para la coordinación del trabajo de un grupo interdisciplinario.

Estos modelos complementan el conjunto de modelos funcionales provenientes de reconocidas metodologías de la ingeniería Web como por ejemplo son OO-H [19] y UWE [59], y proporcionan al proceso información específica sobre la funcionalidad de la aplicación Web. Esta distinción entre modelos de arquitectura y modelos de funcionalidad proporciona una separación de roles que mejora la eficiencia del proceso de desarrollo: mientras que los expertos en el dominio del problema se dedican a la funcionalidad, los expertos en la arquitectura Web definen cuáles son los componentes necesarios para el funcionamiento y posterior mantenimiento de la aplicación. Además, uno de los objetivos más importantes que pretende alcanzar WebSA con la incorporación de la arquitectura es cubrir el vacío que existe en la actualidad entre los modelos funcionales definidos por las metodologías Web y el diseño final que obtienen para dicha aplicación. Las metodologías tradicionales no especifican si la aplicación se representa mediante páginas de servidor o mediante servicios Web con interfaz, si se utiliza un servidor distribuido o no, o si la base de datos es remota o no, etc. Por ello, WebSA define un proceso centrado en la Arquitectura del Software y considera a los componentes de la arquitectura como ciudadanos de primera clase desde la fase de análisis hasta la fase de implementación.

Como indica Greenfield [41], la premisa de que las aplicaciones Web deben estar en un reducido tiempo en el mercado y requieren de una mejora de la calidad conduce a una mayor automatización del proceso de desarrollo. En este sentido, Bezivin [12] señala que las actuales metodologías de la Ingeniería del Software basadas en el paradigma orientado a objetos han alcanzado un elevado grado de agotamiento y es el paradigma dirigido por modelos (MDD) quien se presenta como su sucesor. Por ello, WebSA propone un proceso que intenta establecer una automatización tomando como punto de partida los modelos y como herramienta de mapeo las transformaciones. De este modo es posible definir una trazabilidad precisa desde la definición de los modelos de análisis, pasando por el diseño y llegar a la implementación final.

En este sentido, dentro de la ingeniería dirigida por modelos (MDE) [55] y su representación estándar MDA [89] se ha definido el llamado MDA development process [57] donde el modelo es el artefacto más importante dentro del desarrollo no solamente porque sirve para representar el sistema sino porque se utiliza para formalizar los propios modelos (metamodelos) y para definir las transformaciones entre los modelos hasta su implementación. Como indica Kleppe [56], el desarrollo dirigido por modelos es implementado por la ejecución de una red compleja de transformaciones de modelos. Por ello, WebSA se propone como una instancia del proceso de desarrollo de MDA que incorpora un conjunto transformaciones que realiza cada uno de los diferentes saltos de abstracción en cada fase de desarrollo. La primera transformación convierte los modelos de análisis en un modelo de diseño y la segunda transformación convierte el modelo de diseño en implementación.

4.2 Los Procesos de Desarrollo del Software Actuales

Como señala Kepple [57], *MDA no requiere del uso de ningún proceso concreto para el desarrollo del software*. Sin embargo, los distintos tipos de modelos (CIM, PIM, PSM, etc.) y de transformaciones (modelo-a-modelo y modelo-a-texto) han de ser considerados como parte fundamental en cualquiera de las metodologías de proceso de desarrollo utilizada. A continuación se revisan los procesos de desarrollo más relevantes actualmente y se identificará aquel que se considere más apropiado para WebSA.

4.2.1 Método Ágile

El propósito fundamental de cualquier método de desarrollo es que el software entregado sea satisfactorio para el cliente final. Sin embargo, se ha de tener presente que los requisitos del cliente cambian continuamente, y por lo tanto, el software entregado debe cambiar acorde a estos requisitos. En este sentido, el aspecto central del proceso de desarrollo Ágile [68] es que el proyecto software tenga la capacidad de acomodarse a los cambios de forma flexible e inmediata. En este sentido, WebSA proporciona la característica de una adaptación rápida al cambio desde que el cambio en un modelo supone el cambio en el software. Sin embargo, existen otras características en un método de desarrollo ágile según Miller [80] como son: (1) un ciclo de vida corto e iterativo que permita una rápida corrección y verificación, (2) el tiempo de una iteración vaya de una a seis semanas, (3) estilo de trabajo colaborativo y comunicativo, (4) orientado a las personas en lugar del proceso y a las herramientas.

En este último aspecto, WebSA no se adapta de forma satisfactoria a Agile debido a que como en cualquier proceso basado en MDA es importante el uso adecuado de herramientas que representen los artefactos (modelos y transformaciones) que conduzcan a la implementación.

4.2.2 Extreme Programming

Extreme Programing (XP) [11] es una aproximación muy popular por su manera de trabajar. Define un proceso de desarrollo muy reducido, más centrado en las personas que en las herramientas. XP establece un método donde el código se escribe de forma incremental, manteniendo así el sistema en desarrollo durante todo el tiempo. Cada nuevo requerimiento debe estar acompañado por un caso de prueba que debe probar el software realizado hasta ese momento. Cuando se añada nueva funcionalidad, todos los casos de prueba definidos anteriormente deben probarse y añadirse los casos nuevos, asegurando así la correcta funcionalidad. En este sentido, la principal diferencia con el método Ágile es que XP tiene una visión muy centrada básicamente en el código. Este trabajo a tan bajo nivel provoca que los desarrolladores de implementación y documentación se vean desbordados. Por su lado, WebSA ofrece la posibilidad de trabajar sobre modelos que son transformados automáticamente en código, haciendo que los desarrolladores eleven el nivel de abstracción y reduzcan su trabajo. En este sentido, en lugar de Extreme Programming podría hablarse de Extreme Modeling.

4.2.3 Unified Software Development Process (UP)

El Unified Software Development Process (UP) [50] es un estándar desarrollado por los autores de UML que consiste en un proceso de desarrollo definido de forma genérica y que se especializa o instancia para una organización, proyecto o dominio concretos. Especializar UP supone fijar los modelos que utilizan, definir una estrategia de ciclo de vida concreta, definir las actividades, los tipos de trabajadores, etc. Ejemplos de instancias conocidas de UP son la comercial Racional Unified Process (RUP) [47] y Enterprise Unified Process (EUP) [102]. UP además se fundamenta en tres ideas básicas que identifican su aproximación:

- Dirigido por Casos de Uso: es decir, los casos de uso son utilizados para especificar los requisitos del sistema y a partir de ellos se dirige el análisis, diseño, implementación y test.
- Centrado en Arquitectura: la arquitectura del software juega un papel fundamental dentro de UP. La arquitectura permite realizar una representación de los aspectos estáticos y dinámicos más relevantes del sistema, mejorando así la comunicación entre los actores del sistema y organizando mejor el desarrollo del software.
- Iterativo e Incremental: El desarrollo de productos software está divido en pequeñas iteraciones que van produciendo un incremento o evolución del producto. Además, los modelos una vez definidos no permanecen intactos sino que evolucionan incorporándose nuevos requisitos que provocan cambios.

Además, el proceso de desarrollo de UP está estructurado en cuatro fases (Inicio, Elaboración, Construcción y Transición) y cinco flujos de trabajo (workflows) (Requisitos, Análisis, Diseño, Implementación y Prueba). Durante el desarrollo del

proyecto en cada una de las fases se producirán iteraciones en la que se abordarán los diferentes flujos de trabajo. Así, según la fase en la que se encuentre el proceso se realizará más o menos trabajo en cada tipo de flujo de trabajo. Por ejemplo, en la fase de Inicio se trabaja más sobre el flujo de trabajo de Requisitos, y por el contrario en la fase de Transición se trabaja más sobre el flujo de trabajo Prueba.

Intentado adecuar WebSA al estándar UP, se han encontrado puntos de convergencia a los cuáles WebSA se ajusta perfectamente, como es la relevancia que adquiere la arquitectura del software dentro de su desarrollo y el considerar el proceso como iterativo e incremental. Sin embargo, existen otros aspectos en los que no encaja con la misma naturalidad. WebSA establece el papel fundamental de la arquitectura dentro del proceso y para ello es necesario capturar en la fase de requisitos los requisitos no funcionales. Una de las claves de UP es que esta dirigido por los casos de uso que únicamente permiten capturar los aspectos funcionales. El proceso de WebSA necesita completarse con los requisitos no funcionales para establecer un mapeo adecuado con los modelos de arquitectura. Seguidamente, se procede a mostrar el proceso de desarrollo de WebSA.

4.3 Proceso de Desarrollo de WebSA

El Proceso de Desarrollo de WebSA (Meliá & Gómez [73], Meliá & Cachero [71]) se define como: *"una instancia del proceso estándar UP adecuada al desarrollo Web dirigido por modelos"*. Por lo tanto, para definirlo adecuadamente conviene realizar una descripción que trate cada uno de los elementos genéricos del proceso (flujos de trabajo y artefactos, actores y fases) desde la perspectiva de WebSA. A continuación, se describen las características principales que hacen único al proceso de desarrollo de WebSA y se realiza una revisión de los flujos de trabajo que constituyen el proceso junto con los diferentes artefactos (modelos y transformaciones) que se definen en cada flujo de trabajo. Posteriormente se muestran quiénes son los actores o tipos de desarrolladores encargados de cada una de las tareas. Por último, se realiza una descripción de cómo se distribuye el trabajo de cada flujo de trabajo en cada una de las fases del proceso.

4.3.1 Características Principales

Seguidamente se especifican las características que son consideradas de mayor relevancia dentro del proceso de desarrollo de WebSA:

1. **Específico para el dominio de las aplicaciones Web**. En el proceso de desarrollo WebSA todos los artefactos que intervienen (desde los modelos funcionales a los modelos de arquitectura) son específicos para el modelado de las aplicaciones Web. Esto dota al proceso de una mayor precisión a la hora de definir el problema, y reduce tanto espacio del problema (especificaciones más sencillas) como el espacio de la solución. Además, permite aumentar la calidad del producto producido y en muchas ocasiones mejorar la reutilización mediante componentes Web específicos definidos previamente.
2. **Centrado en la Arquitectura del Software**. Como ya se ha mencionado en el capítulo 3, WebSA define un proceso de los llamados "centrados en

arquitectura", lo que implica considerar a los elementos definidos en la arquitectura (como son los componentes, conectores y subsistemas) como fundamentales a la hora de modelar y evolucionar el sistema. Esto permite representar la arquitectura adecuada desde una fase temprana, así como poder aplicar los patrones adecuados a cada una de las fases del desarrollo, desde los patrones de arquitectura más genéricos, pasando por los patrones de diseño, hasta patrones de código. Además, el proceso WebSA define claramente dónde se ubica cada uno de los aspectos funcionales específicos de las aplicaciones Web dentro de la propia arquitectura.

3. **Automatizable desde el análisis hasta la implementación.** El proceso WebSA se centra principalmente en los modelos funcionales y de arquitectura definidos en la fase de análisis. A partir de estos modelos, se ejecutan las transformaciones necesarias para obtener los modelos de diseño y posteriormente la implementación. Así, la intervención humana se reduce únicamente a la fase de análisis y a los momentos en que se especifica una transformación específica para un sistema en concreto. De este modo se reduce el coste de modelado y tiempo de desarrollo, y por otro lado, se mejora la calidad ya que se reduce sustancialmente el número de potenciales errores que se pueden producir por la intervención humana.
4. **Dirigido por modelos**. WebSA se sitúa dentro de un nuevo paradigma llamado MDE que establece que el modelo es el artefacto central dentro del proceso de desarrollo. El modelo permite definir el sistema, formaliza el propio modelo mediante su metamodelo y facilita la comunicación entre diferentes analistas y modeladores. Es además un elemento reutilizable entre diferentes sistemas, e incluso las propias transformaciones de modelos son modelos que establecen una trazabilidad desde el modelo hasta su implementación.
5. **Estandarizado y Genérico**. El proceso de desarrollo WebSA se basa en los estándares proporcionados por MDA [57] y permite la integración de las diferentes metodologías mediante el uso de las transformaciones entre los diferentes modelos. De este modo los modelos WebSA pueden ser definidos en cualquier herramienta que soporte UML 2.0. Por otro lado, el proceso WebSA permite integrar la arquitectura con la información funcional obtenida mediante cualquier metodología cuyo metamodelo esté especificado en MOF. También utiliza MOF para definir las transformaciones UPT (Meliá & Gómez [76]) y MOFScript [94], todo lo cual contribuye a su genericidad.

Seguidamente se presentan los diferentes flujos de trabajo y artefactos que se definen dentro del proceso de desarrollo de WebSA.

4.3.2 Flujos de Trabajo del Proceso de WebSA

El proceso de WebSA es una especialización del proceso de desarrollo MDA en el que los artefactos de salida de cada flujo de trabajo son modelos que representan la especificación de una aplicación Web a diferentes niveles de abstracción y desde diferentes perspectivas. Como se aprecia en la Figura. 5, los flujos de trabajo de

requisitos y análisis están divididos en dos perspectivas, la perspectiva funcional Web y la perspectiva arquitectónica Web. Por un lado, en la perspectiva funcional Web inicialmente se definen los casos de uso que recogen los requisitos funcionales necesarios para la definición de los modelos de análisis especificados por diferentes aproximaciones de Ingeniería Web como OOH [19], UWE [59], etc. Simultáneamente, en la perspectiva arquitectónica Web son especificados los requisitos no funcionales por parte del cliente, que servirán para la definición de los modelos de arquitectura definidos en la parte de análisis. Estos modelos son estilos arquitectónicos que permiten especificar a grandes rasgos la arquitectura de la aplicación. Se trata del modelo de subsistemas, que define un estilo de capas, basado en el trabajo de [17] y el modelo de configuración, que define un estilo expresado mediante componentes. El modelo de casos de uso utilizado en la fase de requisitos se puede considerar como un modelo CIM dentro de MDA, mientras que los modelos definidos en la fase de análisis son modelos PIM, siendo independientes de la plataforma en la que se implementa la solución. Otro aspecto a destacar es que el mapeo de los requisitos funcionales a los modelos de análisis funcional se muestra mediante una flecha punteada indicando que se realiza de forma automática o manual dependiendo de la propuesta utilizada. Por otra parte, el paso entre requisitos no funcionales y arquitectura se muestra con una flecha discontinua que indica que se realiza de forma manual. Esto se debe a que los artefactos definidos en los requisitos no funcionales no ofrecen la posibilidad de definir un mapeo completo a los modelos de análisis arquitectónico.

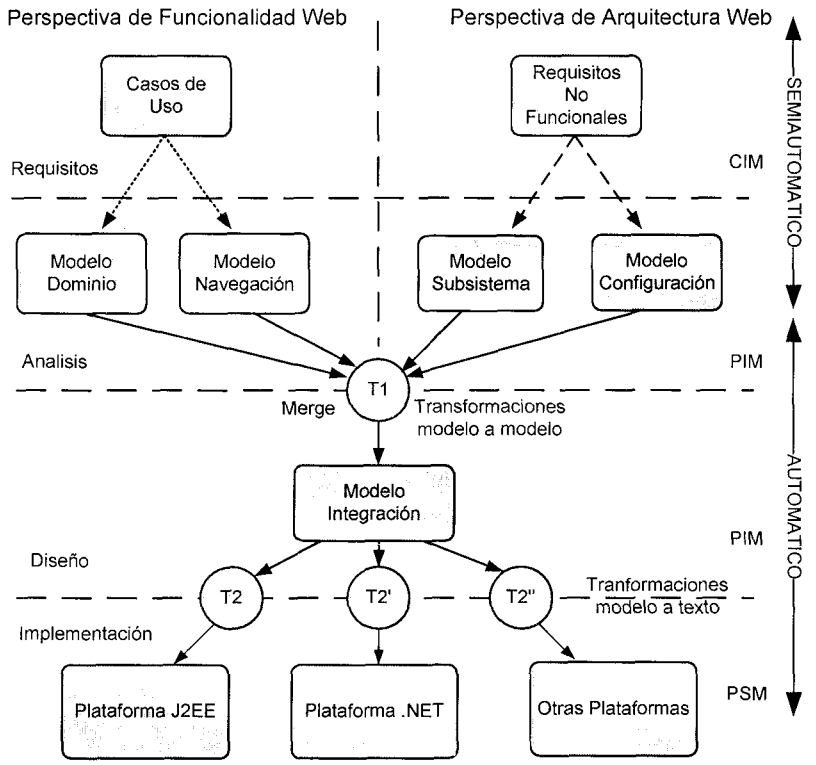

Figura. 5. Flujos de Trabajo y Artefactos del Proceso de Desarrollo de WebSA

Una vez especificado los modelos de análisis comienza la parte automatizada del proceso WebSA. Se establece una transformación PIM a PIM que tiene como origen los modelos de análisis independientes de plataforma y como destino el modelo de integración. Esta transformación se basa en los metamodelos MOF de las diferentes aproximaciones para obtener su información, de manera que consigue realizar una fusión de la información proveniente de los modelos de la perspectiva funcional con los modelos de la perspectiva de arquitectura (subsistemas y configuración). Así, se obtiene un modelo de arquitectura que incorpora las características de ambas perspectivas que se denomina modelo de integración. El modelo de integración se encuentra ubicado en el flujo de trabajo de diseño independiente de plataforma.

Como se aprecia en la Figura. 5, a partir del modelo de integración surgen diferentes transformaciones T2, T2' y T2''. Cada una de ellas, es una transformación PIM a PSM, en las cuáles se introducen los aspectos propios de una determinada plataforma. Cada transformación es una composición de transformaciones que obtiene la información sobre los componentes de arquitectura independientes de plataforma y los transforma en los componentes propios de una plataforma en particular. Obteniendo así, en la fase de implementación el código de las plataformas como J2EE, .NET, etc.

A primera vista, los diferentes flujos de trabajo que contiene el proceso de desarrollo de WebSA son muy similares a los definidos en los procesos de desarrollo tradicionales. Sin embargo, la diferencia radica en los modelos generados desde el análisis hasta la implementación del proceso de WebSA. Son modelos formales entendibles por las computadoras y son la entrada y salida de las transformaciones realizadas entre cada flujo de trabajo, en lugar de hacerse de forma manual como se realiza en el proceso tradicional.

A continuación, se presentan los diferentes actores o roles de los desarrolladores necesarios para llevar a cabo el proceso de desarrollo de WebSA.

4.3.3 Actores o Roles del Proceso de WebSA

Para definir adecuadamente un proceso de desarrollo es necesario identificar quienes son los diferentes tipos de actores o roles (stakeholders) que necesarios para realizar las distintas tareas. En el proceso WebSA se definen los siguientes roles:

- **Expertos de Dominio Web:** Personas conocedoras del dominio de la aplicación Web y encargadas de a partir de los requisitos funcionales definidos en los casos de uso, realizar los modelos funcionales definidos en el análisis. WebSA permite definir los modelos funcionales utilizando otras metodologías Web. Por lo tanto, los expertos de dominio tienen la posibilidad de poder elegir la metodología Web en la que se encuentren más cómodos.
- **Arquitectos Web:** Personas encargadas de la definición de los modelos de arquitectura definidos en la fase de análisis a partir de los requisitos no funcionales. Se caracterizan por tener una visión global de la arquitectura de una aplicación Web y de conocer cuáles son los diferentes elementos que aparecen en ella.
- **Expertos en la Transformación T1:** Persona encargada de definir transformaciones que extienden T1 introduciendo aspectos de arquitectura que sean específicas para el dominio de aplicación. Para ello, requiere del conocimiento del lenguaje de transformaciones UPT, de la arquitectura Web y de la funcionalidad Web (dominio y navegación).
- **Expertos en la Transformación T2:** Persona encargada de introducir transformaciones modelo a texto que extienden la transformación T2. Para ello, deben ser conocedores del modelo de integración que es el origen de la transformación, así como de las transformaciones de modelo a texto definidas en MOFScript y de la plataforma concreta .NET o J2EE que es el destino de la transformación.
- **Diseñadores de Interfaz Web:** Personas que poseen un perfil más cercano al diseño gráfico que a la programación. Se encargan de realizar el diseño de las páginas Web, trabajando únicamente sobre HTML y CSS y respectando la programación que las páginas de servidor puedan contener. En cualquier caso, las tareas que realizan quedan fuera del interés de esta propuesta.
- **Programadores de Lógica de Negocio:** Personas que realiza la tarea de completar aquellos métodos de los componentes de lógica de negocio que no han podido ser codificados desde los modelos funcionales de WebSA, y que

únicamente deben completar ciertos métodos de los componentes de lógica de negocio.

Como puede apreciarse, las personas involucradas en el proceso de desarrollo WebSA requieren tener un conocimiento elevado en el modelado Web y las transformaciones, y solamente los expertos en la transformación T2 necesitan conocer la implementación en las diferentes plataformas. Esto pone de relieve que el proceso de desarrollo WebSA eleva el nivel de abstracción y permite trabajar cada vez menos sobre la implementación y más sobre el modelado.

4.3.4 Fases del Proceso de WebSA

Como instancia del proceso de desarrollo de UP, el proceso WebSA define 4 fases principales. En la Figura. 6 se muestran las diferentes fases, cual es el trabajo realizado sobre cada uno de los flujos de trabajo y los artefactos que se van obteniendo en cada uno de los flujos de trabajo.

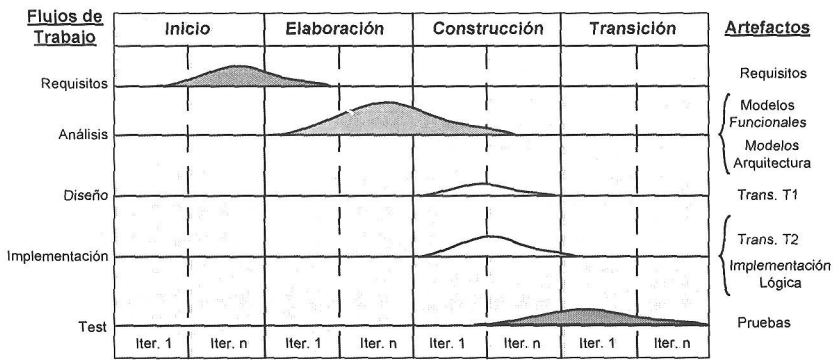

Figura. 6. Fases del Proceso de Desarrollo de WebSA

A continuación se describen cada una de las fases del proceso de desarrollo de WebSA:

- **Inicio**: El principal objetivo de esta fase es alcanzar el acuerdo entre todas las personas que participan, es decir entre los clientes y los analistas (arquitectos y expertos en dominio). Se realiza la definición de los requisitos funcionales y no funcionales. La principal motivación de inicio es fijar los objetivos principales del proyecto y evaluar su viabilidad. En esta fase WebSA sigue con fidelidad las premisas establecidas por UP. Las iteraciones que se producen en esta fase siempre son motivadas por los cambios de requisitos que se producen por parte del cliente.
- **Elaboración**: El principal objetivo de esta fase es establecer la base arquitectónica y funcional que permitirá obtener posteriormente el diseño y la implementación. En esta fase termina la definición de los requisitos tanto funcionales (mediante casos de uso) como no funcionales (mediante especificaciones textuales). A partir de ellos, los analistas establecen a grandes rasgos la arquitectura de la aplicación mediante los dos estilos arquitectónicos que proporciona WebSA (modelo de subsistema y

configuración). Concurrentemente, se definen los modelos funcionales específicos de las aproximaciones Web, es decir, dominio y navegación. Dentro de WebSA es la fase más importante y como se aprecia en la Figura. 6 es la fase que mayor esfuerzo requiere, puesto que la mayoría de los modelos de análisis han de elaborarse manualmente y es a partir de ellos comienza la automatización del proceso hasta la implementación. Su realización requiere de diversas iteraciones en las que se irán mejorando los modelos de análisis en función del diseño o implementación obtenidos.

- **Construcción**: Basándose en la arquitectura y funcionalidad definida en la Elaboración, en la fase de Construcción se realiza el desarrollo del diseño y la implementación del sistema. Debido a que WebSA es un proceso MDA y se basa en la automatización proporcionada por las transformaciones, tanto el modelo de diseño como la implementación son obtenidos de forma automática. Aquí reside la principal diferencia con el proceso definido por UP donde diseño e implementación se realizan de forma manual, y por lo tanto, la construcción es la fase de mayor coste. Sin embargo, en WebSA la construcción exige únicamente un trabajo de refinamiento sobre las transformaciones tanto en T1 como en T2, en los cuáles se recogen aquellos casos especiales que son específicos del sistema en desarrollo. La implementación requiere tan solo de la introducción de la lógica de negocio que no ha podido especificarse mediante el modelo de dominio, así como de la especificación del diseño gráfico de las páginas Web por parte del diseñador de interfaz Web. En cualquier caso, se reduce enormemente el coste de elaboración de los flujos de trabajo de diseño e implementación. Finalmente, hay que destacar que en general el proceso WebSA asume varias iteraciones en cada una de las cuales se producen refinamientos sucesivos del producto obtenido hasta que se considera que la aplicación está lista para realizar el despliegue.
- **Transición**: El foco de la fase de transición es asegurar que la aplicación está disponible para los usuarios finales. Para ello, se realizan las pruebas necesarias para la preparación del producto en versión definitiva (release). Llegados a este punto el cliente suele sugerir diversos ajustes que requieren el refinamiento del producto, en aspectos como la configuración, instalación, presentación, usabilidad, etc. En este momento se decide entonces si los objetivos del proyecto han sido alcanzados, o por el contrario se necesitan más iteraciones.

Una vez definido el proceso a grandes rasgos, se procede a recorrer cada uno de los flujos de trabajo y sus correspondientes artefactos.

4.4 Los Requisitos

A pesar de no existir un estándar dentro de la ingeniería Web para soportar la fase de requisitos, las mejores prácticas de desarrollo de aplicaciones Web proporcionan un conjunto de técnicas. Un reciente estudio comparativo sobre la ingeniería de

requisitos para el desarrollo de aplicaciones Web mostró que el modelado de casos de uso es la técnica más popular para la especificación de los requisitos funcionales mientras que las entrevistas es la técnica más utilizada para la captura de estos requisitos. En este sentido, la práctica común coincide en la definición del flujo de trabajo de requisitos del estándar UP. Sin embargo, WebSA recomienda la separación del flujo de trabajo de requisitos en dos perspectivas diferentes, la perspectiva funcional Web y la perspectiva de arquitectura Web (ver Figura. 5). Al añadir la captura y especificación de los requisitos no funcionales se persigue mejorar la satisfacción del cliente al incluir desde el inicio del desarrollo la consideración específica de una arquitectura del software acorde con sus necesidades.

A continuación, se describe como se ha especificado los requisitos en el proceso de WebSA.

4.4.1 Requisitos Funcionales

WebSA se basa en dos metodologías Web existentes como OO-H y UWE para definir su parte funcional, y por tanto sigue sus indicaciones a la hora de definir los requisitos funcionales. En este caso, ambas aproximaciones utilizan el diagrama de Casos de Uso definido por UML [95] como modelo para expresar los requisitos funcionales.

Los principales elementos de modelado de casos de uso definidos por la especificación UML son los actores representados mediante un hombre y los casos de uso representados mediante un óvalo. Los actores pueden estar relacionados con los casos de uso mediante asociaciones, que indican que existe una comunicación entre una instancia del caso de uso y el usuario que ostenta el rol representado por el actor. Los casos de uso pueden estar relacionados con otros casos de uso mediante relaciones de tipo inclusión (<<include>>), extensión (<<extend>>) y generalización. La Figura. 7 muestra un ejemplo donde se representan cada uno de los elementos de un Caso de Uso con la notación de UML.

Con el fin de construir el modelo de casos de uso de la aplicación Web, OO-H sugiere los siguientes pasos definidos por [103]:

- Identificar actores
- Para cada actor, identificar las actividades que éste realiza en el sistema
- Agrupar las actividades en casos de uso
- Establecer las asociaciones entre casos de uso y actores
- Establecer las generalizaciones entre actores necesarias para simplificar el diagrama
- Establecer las relaciones de tipo <<include>> y <<extend>> entre casos de uso.

Aunque la especificación capturada mediante el diagrama de Casos de Uso se puede enriquecer con diversos mecanismos (escenarios, diagramas de actividad, diagramas de colaboración o secuencia, etc.), WebSA no realiza ninguna recomendación respecto a su uso y en caso de producirse, estos diagramas sólo son tenidos en cuenta como documentación del sistema y no influyen en el resto de fases del proceso.

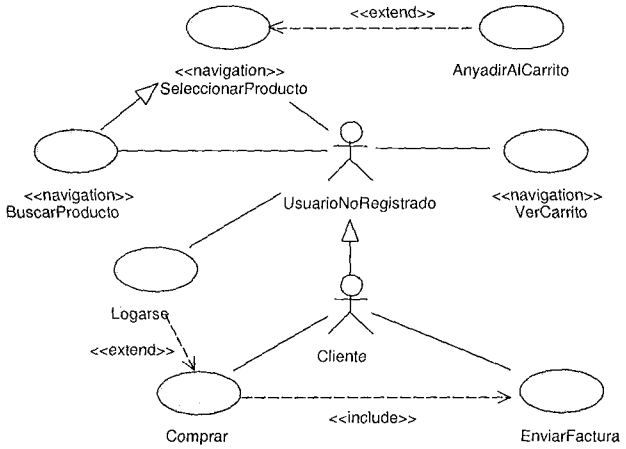

Figura. 7. Caso de Uso de una Tienda Virtual

La Figura. 7 muestra un ejemplo de diagrama de Caso de Uso definido en el trabajo [60], que representa la interacción de los usuarios de una tienda virtual.

Una vez definido los casos de uso, algunas aproximaciones proponen la automatización del mapeo desde el modelado de casos de uso hasta la fase de análisis funcional (NDT [29] para UWE [29] y Valderas [122] para OOWS). El resto de aproximaciones realizan el mapeo de una forma manual.

4.4.2 Requisitos No Funcionales

WebSA para abordar la especificación de los requisitos de calidad y negocio del sistema, se decanta por la enumeración de forma textual de cada uno de ellos. Cada requisito de calidad debe ser considerado tanto por el estímulo como por la respuesta deseada. Un requisito de calidad especificado de la forma: "*El sistema tendrá que ser modificable*" carece de sentido, desde el punto de vista de que todos los sistemas tienen partes fácilmente modificables y contiene otras difíciles de modificar. Un requisito de modificabilidad deberá definirse caracterizando el conjunto de modificaciones, como p.e. "*El sistema de Comercio electrónico será modificable respecto a cambios en su navegación*". Esto supondrá que en el flujo de trabajo de análisis se tomen las medidas adecuadas en la arquitectura para que los cambios de navegación sean lo menos traumático posible. Por ejemplo, utilizar un almacén que contenga la navegación en un fichero aparte, permite no tener que modificar el código para cambiar la navegación.

La diferencia entre los requisitos de negocio y de calidad no está claramente definida. Hay muchos objetivos de negocio para la arquitectura que pueden transformarse en requisitos abstractos de calidad. Por ejemplo, el objetivo "*La arquitectura soporte una determinada línea de productos*" se podrá transformar en una colección de requisitos de extensibilidad. WebSA considera conjuntamente los requisitos de calidad y negocio ya que el origen de los requisitos no es importante para el proceso.

La relación entre los requisitos no funcionales y la arquitectura se realiza a través de los patrones. Las fuerzas que llevan a la aplicación de un determinado patrón de arquitectura, son los propios requisitos no funcionales capturados en esta fase. Esto ocurre, tanto en el modelo de subsistemas mediante los patrones de distribución como en el modelo de configuración mediante los patrones de arquitectura. Por ejemplo, si a la hora de proponer una interfaz de usuario el cliente desea que se cumplan requisitos no funcionales como una fácil distribución y seguridad se propone una interfaz de usuario fina aplicando el patrón de distribución llamado Presentación distribuida (ver sección 5.5.1.2). Sin embargo, si es la usabilidad y el rendimiento los requisitos que más importan al cliente a la hora de definir la interfaz, se elegirá una interfaz grueso mediante la aplicación del patrón de distribución Interfaz de Usuario Remoto (ver sección 5.5.1.1).

4.5 El Análisis

El flujo de trabajo de análisis sigue el mismo objetivo que el definido genéricamente por UP, es decir, se centra en la realización de una representación del sistema a alto nivel de abstracción y partiendo de los requisitos funcionales y no funcionales que se han recogido en la fase de inicio. Sin embargo, en el proceso de WebSA este flujo de trabajo toma especial relevancia, puesto que es el que mayor esfuerzo requiere y el que mayor impacto tiene sobre el resto de las fases, debido a que a partir del análisis se establece un proceso basado en transformaciones que conduce al diseño y a la implementación.

Al igual que el flujo de trabajo de requisitos, el flujo de trabajo de análisis está divido en las perspectivas funcional y de arquitectura (ver Figura. 5). Esta característica diferencia a WebSA del resto de aproximaciones Web como UWE [59], OOH [19], WebML [20], etc. que se centran únicamente en la parte funcional.

Tanto la parte funcional como la de arquitectura se pueden definir de forma independiente. Esto permite que los actores encargados de cada perspectiva (los expertos de dominio y los arquitectos Web) puedan trabajar de forma concurrente, ahorrando tiempo en la definición del análisis. Además, se puede realizar una reutilización de cada uno de los modelos definidos en cada una de las vistas. Por un lado, es posible reutilizar la misma vista de arquitectura para diferentes aplicaciones. Por otro, es posible aplicar el mismo problema o vista funcional para diferentes arquitecturas.

Los modelos de análisis se representan en un alto nivel de abstracción, lo que implica una reducción de número de modelos y elementos necesarios para representar el sistema. Esto permite representar el sistema con un menor coste de modelado respecto a los flujos de trabajos de diseño e implementación.

4.5.1 Análisis Funcional Web

El análisis funcional Web es una actividad centrada en el dominio del problema que captura los aspectos funcionales que se consideran relevantes para la aplicación Web.

En el proceso de desarrollo de WebSA, el análisis funcional es notacionalmente genérico, es decir, admite el uso de diferentes aproximaciones para representar los modelos funcionales. Esto permite que los expertos en el dominio del problema,

puedan utilizar la aproximación Web que le sea más familiar y fácil de usar. Además es posible combinar, reutilizar e intercambiar modelos expresados en distintas notaciones dentro del proceso de desarrollo de una aplicación determinada.

Actualmente, WebSA se ha centrado principalmente en dos aproximaciones: OOH y UWE. Esto no descarta, que en un futuro inmediato, se puedan incluir dentro del análisis funcional de WebSA otras aproximaciones. Para que WebSA pueda utilizar los modelos definidos por una aproximación, estos deben estar formalizados en un metamodelo MOF. Solamente así el proceso WebSA puede establecer las transformaciones necesarias para integrar la funcionalidad con la arquitectura.

A continuación, se detallan los modelos funcionales utilizados por el proceso WebSA: el Modelo de Dominio y el Modelo de Navegación. Sobre cada uno de ellos se presentan sus elementos, su metamodelo MOF, el método de aplicación y un ejemplo del modelo. Así se consigue que la tesis sea más autocontenida y no se tenga la necesidad de consultar los modelos en diferentes fuentes.

4.5.2 Modelo de Dominio

También conocido como modelo de contenido en UWE o modelo estructural en OO-H. El modelo de dominio (MD) surge como resultado de un análisis de dominio que captura los objetos y relaciones que colaboran en dar respuesta a los requisitos de la aplicación. Por lo tanto, el modelo de dominio se centra principalmente en las entidades del dominio de la aplicación Web, las cuáles están libres de cualquier aspecto técnico o detalles de implementación, representando así un diagrama de clases ideal.

El MD es de gran importancia ya que sobre él se toman las primeras elecciones sobre que conceptos del dominio capturar o no, lo que repercutirá en el modelo de navegación.

4.5.2.1 Elementos del modelo

Los elementos de modelado principales presentes en el MD son las clases (con sus atributos y operaciones) y sus relaciones (asociación, agregación, composición, generalización). La semántica y representación gráfica de estos elementos se basa en la presentada en el diagrama de clases de UML [95].

4.5.2.2 Metamodelo de dominio

En el metamodelo de MD es un subconjunto del metamodelo que UML establece para el diagrama de clases. A continuación, en la Figura. 8 se muestra el metamodelo MOF del MD definido tanto para WebSA, y que es común tanto para OO-H como para UWE. Este metamodelo, es autocontenido, es decir, define todos los elementos necesarios para definir el MD sin depender de otros.

El elemento central del metamodelo es la clase *Clase* que está relacionada por una relación de composición con las propiedades que contiene, es decir, *Atributo* y *Operacion*. Cada una de las operaciones puede contener a su vez un conjunto de elementos *Argumento*. Por último indicar las relaciones de asociación, en todos sus tipos, que están representas por la clase *Asociacion*. Y la herencia que está representada mediante la relación *Superclase*.

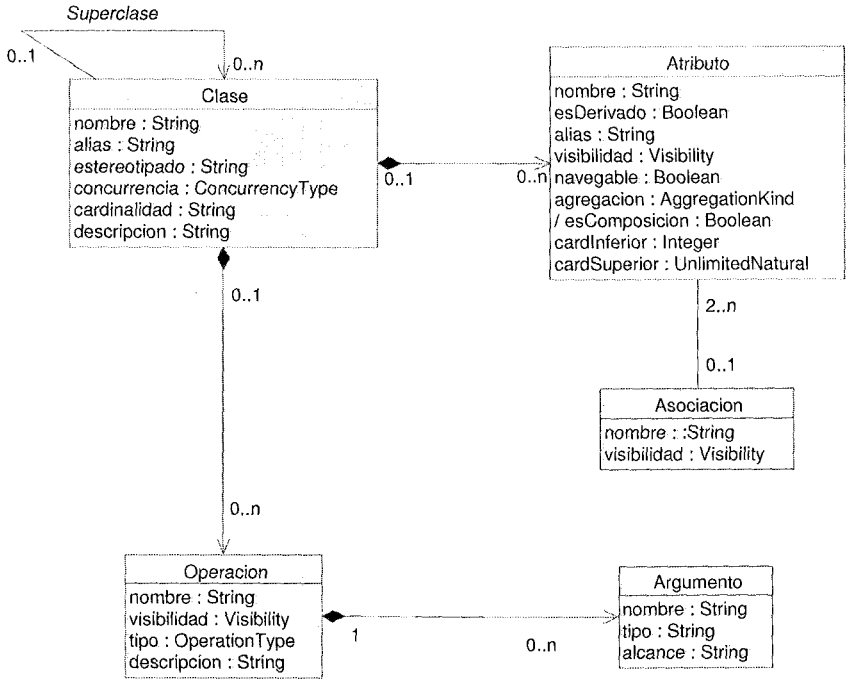

Figura. 8. Metamodelo de Modelo de Dominio.

Una vez formalizado los elementos y las relaciones que constituyen MD, se procede a mostrar un ejemplo de MD para una aplicación de comercio electrónico.

4.5.2.3 Ejemplo de Modelo de Dominio (Petstore)

La Figura. 9 muestra un MD que contiene las entidades del dominio del problema más importantes de la conocida aplicación de comercio electrónico Petstore [115]. La clase *Cliente* contiene un conjunto de atributos que representan los datos personales (*idCliente, email, nombre, dirección, teléfono*, etc.). También, la aplicación contiene las preferencias del cliente en la clase *Perfil* como categoría favorita, lenguaje, etc. Para poder facilitar la accesibilidad de los diferentes productos, cada *Producto* es clasificado en una *Categoría* (por ejemplo, un loro corresponde a la categoría de pájaro). Así, el *Cliente* puede seleccionar un producto particular a partir de una lista de categorías. En ese momento, la aplicación muestra la información detallada sobre el producto seleccionado. La clase *Producto* contiene la descripción y la imagen. Cuando existen varias variantes del mismo *Producto*, cada variante es mostrada en un *Artículo* separado. Por ejemplo, cuando se mostraran los detalles sobre un periquito africano, un artículo podría ser un gran periquito africano macho, o un pequeño periquito africano hembra, los cuáles tienen diferente coste por unidad, proveedor, stock, etc. Cuando el *Cliente* decide comprar un *Artículo* particular invoca a la operación *anyadirCar* sobre *Artículo* para añadirlo al carrito de la compra. Finalmente, el *Cliente* puede elegir pedir los artículos que contiene el carrito de la

compra en cualquier momento. Entonces es ejecutada la operación de *comprar* desde la clase Carrito.

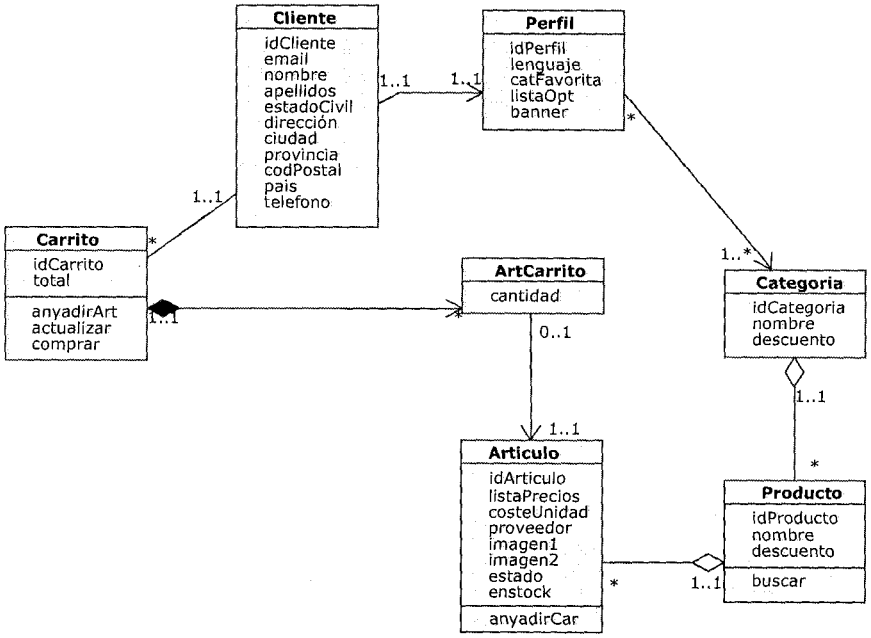

Figura. 9. Modelo de Dominio de Petstore

Una vez definido el modelo de dominio, el siguiente paso en el análisis funcional consiste en especificar el modelo de navegación.

4.5.3 Modelo de Navegación (MN)

A pesar de que el concepto de navegación es común para cualquier tipo de modelo de interfaz ha sido dentro del desarrollo de aplicaciones Web donde con mayor claridad ha surgido la necesidad de especificar cuáles son los saltos navegacionales que existen entre diferentes páginas de la interfaz de la aplicación Web.

El modelo de navegación (MN) permite la representación de los caminos semánticamente relevantes a través del espacio de información. No sólo documenta la aplicación Web, sino que ayuda a controlar su complejidad y a evaluar su adecuación para responder a requisitos funcionales de usuario.

Esto ha provocado que todos los métodos de modelado Web hayan incluido un MN en su proceso de desarrollo. Por ello, el modelo de navegación dentro de WebSA es considerado como un artefacto imprescindible dentro de la perspectiva funcional de una aplicación Web. En el modelo de vistas de WebSA (ver la Figura. 4), se puede apreciar, cómo el MN es dependiente del MD. Esto es debido a que los enlaces de navegación se establecen entre los conceptos que se han capturado dentro del modelo

de dominio. La dependencia indica que solamente puede definirse el MN si se ha definido previamente el MD.

El MN es también independiente de los aspectos de plataforma, con lo que se considera un modelo PIM dentro de MDA. Existen diferentes métodos que definen la navegación en los flujos de trabajo de análisis y de diseño, dependiendo del nivel de abstracción. WebSA ha elegido para definir MN el modelo de navegación de UWE y el modelo de acceso navegacional (DAN) definido por OOH. Sin embargo, en el futuro está previsto incorporar otras aproximaciones como (WebML [20], W2000[8]) para definir los conceptos de navegación.

4.5.3.1 Elementos del modelo de navegación

Para definir la navegación WebSA permite utilizar dos artefactos diferentes: el DAN de OOH o el modelo de navegación de UWE. A continuación se presentan los elementos principales de cada una de las diferentes propuestas.

Destacan como elementos principales del modelo de navegación de OOH los siguientes elementos:

- **Clase Navegacional** (CN): son vistas parciales sobre clases de dominio, y reflejan el grado de interés que cada uno de sus atributos/operaciones tiene para cada tipo de usuario.
- **Destino Navegacional** (DN): agrupa los elementos del modelo que colaboran en la cobertura de los distintos requisitos navegacionales del usuario.
- **Enlace Navegacional** (EN): En este caso, OOH, distingue entre 3 tipos principales de enlace, (1) Enlace de requisito: Marca el inicio de uno o más caminos de navegación, destinados a dar respuesta a los requisitos del usuario, (2) Enlace de travesía: establece el enlace entre dos elementos navegacionales, ya sea dos CN, colecciones o DN (3) Enlace de servicio: define los posibles caminos de navegación que un usuario puede requerir para proporcionar/recibir la información necesaria durante su interacción con cualquier servicio de entre los definidos en el modelo de dominio.
- **Colección**: Estructura de agrupación de enlaces navegacionales. El ejemplo más típico es el menú.

En el modelo de navegación de UWE, como elementos más importantes:

- **Nodo Navegacional** (NN): clase abstracta que representa a cualquier tipo de nodo dentro de un modelo navegacional. Puede tratarse de una clase navegacional, una visita guiada, un índice o una consulta.
- **Clase Navegacional** (CN): es una clase del dominio cuyas instancias son visitadas por el usuario durante la navegación. También contendrá los atributos navegacionales que sean de interés mostrar para el usuario.
- **Enlace Navegacional** (EN): Establece una navegación entre dos nodos navegacionales.
- **Clase Proceso** (CP): Se trata de una clase del dominio, cuyas instancias participan en la ejecución de un proceso ejecutado por la lógica de negocio del sistema.
- **Enlace de Proceso** (EP): Indican los puntos de inicio de proceso dentro de la estructura de navegación.

Como indica [19], los modelos de navegación a pesar de presentar diferencias y pequeños matices semánticos, contienen conceptos que se repiten en casi todos como son la clase navegacional y el enlace navegacional.

Seguidamente se formalizan los elementos y relaciones de los artefactos utilizados como MN en WebSA mediante sus metamodelos.

4.5.3.2 Metamodelo del modelo de navegación

La Figura. 10 muestra el metamodelo de navegación de OOH. En él se representan los elementos de MN y además se establecen las posibles relaciones y configuraciones válidas entre cada uno de los elementos.

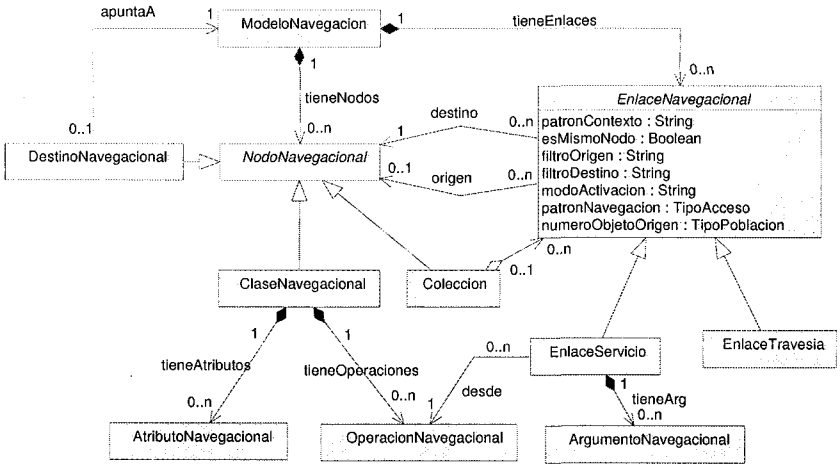

Figura. 10. Metamodelo del Modelo de Navegación de OOH.

En el metamodelo de OOH se han introducido dos elementos abstractos como son *NodoNavegacional* y *EnlaceNavegacional* conceptos que no son utilizados en el propio modelo, pero que sirven para abstraer conceptos comunes en las diferentes especializaciones. Por ejemplo, en el *EnlaceNavegacional* se establecen un conjunto de atributos como son el patrón de contexto, los filtros origen y destino, etc., los cuáles son aplicados a todos los tipos de enlace. Para más información consultar [19].

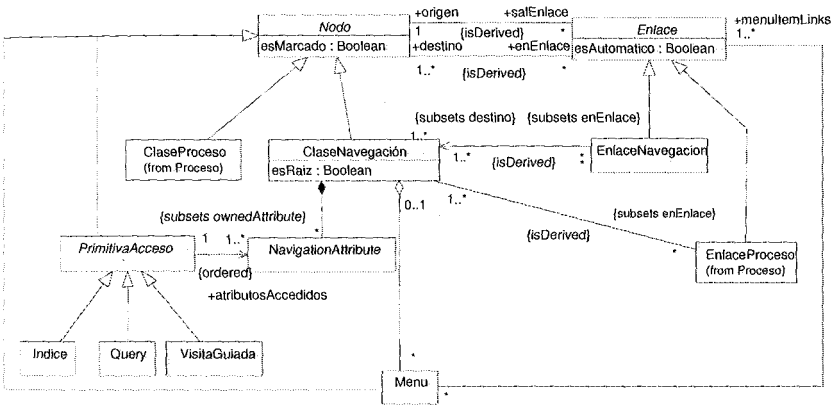

Figura. 11. Metamodelo del Modelo de Navegación de UWE.

UWE [62] propone definir su metamodelo extendiendo de forma conservadora el metamodelo definido por UML. Para poder ser utilizado desde WebSA y definir las transformaciones que permitan obtener la información directamente sobre las metaclases, se han extraído los elementos en un metamodelo MOF (ver Figura. 11).

4.5.3.3 Ejemplo de Modelo de Navegación (Petstore)

La Figura. 12 muestra el destino navegacional principal del modelo de navegación de Petstore en OO-H. Situándose en la izquierda de la figura se aprecia el punto de entrada de navegación, que consiste en un enlace de requisito que apunta a un triángulo invertido, que es el icono que OO-H utiliza para representar una Colección. Esta colección constituye el conjunto de enlaces del menú principal. El *Menú* da acceso a la clase navegacional *Categoría* por medio de un enlace navegacional (representado mediante una flecha). OO-H define distintos tipos de flechas: en concreto, el que la cabeza de la flecha esté rellena indica que se navega a una página diferente. El que la línea sea continua, por otro lado, indica que es un enlace manual, es decir, activado por el usuario. En la clase *Categoría* se muestra el atributo nombre, y además el conjunto de *Productos* asociados por medio de un nuevo enlace navegacional que tiene un círculo en el origen. Este círculo indica que es un enlace definido sobre una relación de rol subyacente en el MD, es decir, que sólo se van a ver los productos asociados a la categoría que el usuario haya elegido previamente. Ese enlace muestra la información en la misma página (cabeza blanca) y es automático (línea discontinua). Seleccionado un *Producto*, se navega a otra página específica del producto concreto, donde se muestran más detalles del mismo, así como el conjunto de *Artículos* asociados a ese producto. De la misma manera, en esta página se puede seleccionar cualquier *Artículo* para verlo con más detalle. Desde ambas páginas se puede activar la operación *anyadirACarrito* mediante la activación de un enlace de servicio. Este enlace añade el *Artículo* al *Carrito* y lo muestra en la página del *Carrito*, desde la cual se ve además el monto total acumulado de la compra. Por último, el modelo muestra un enlace a otro destino navegacional llamado *Compra* que contiene el resto de la navegación para realizar la compra.

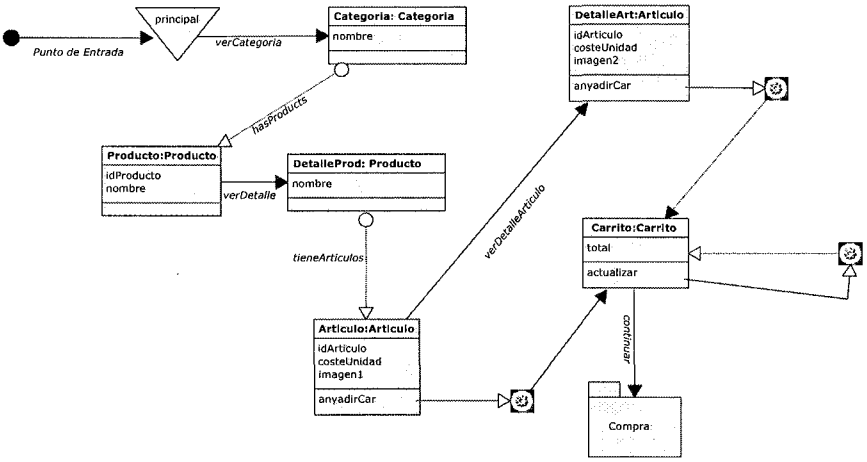

Figura. 12. Modelo de Navegación de OOH para Petstore.

La Figura. 13 muestra como el modelo de navegación de UWE introduce nuevos iconos que son estereotipos del modelo de clases de UML. El rectángulo representa una clase navegacional, el cuadrado con líneas representa un índice y el icono con forma de flecha representa una clase de proceso (Ver descripción en [59]). Este modelo describe los mismos pasos de navegación que el modelo de OOH, pero utilizando sus propios conceptos. Tal vez la principal diferencia semántica, es que en el modelo de navegación de UWE no se establecen cuales son los atributos que va a mostrar cada clase navegacional, puesto que este aspecto es tratado en el modelo de presentación. Sin embargo, en UWE si se especifica la relación entre el modelo de navegación y el modelo de proceso mediante la introducción en el modelo de navegación de los conceptos ClaseProceso y EnlaceProceso.

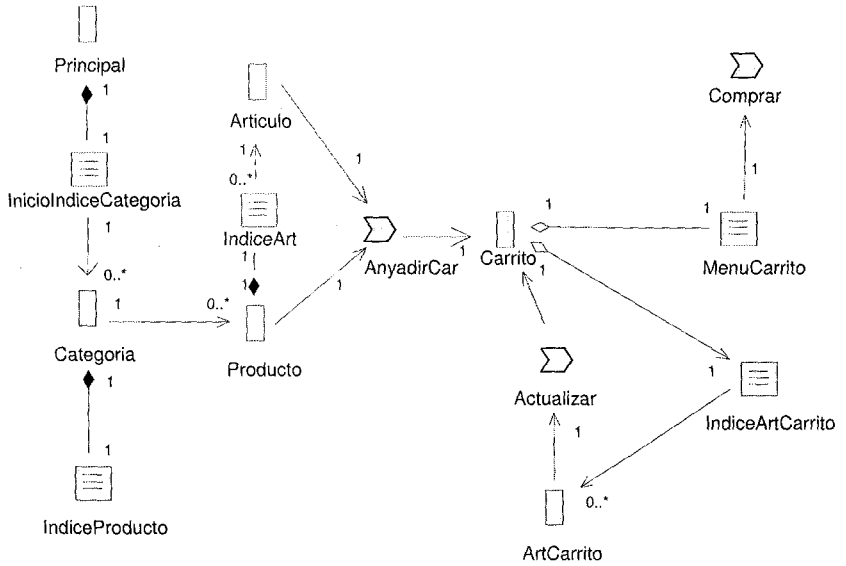

Figura. 13. Modelo de Navegación de UWE para Petstore.

4.5.4 Análisis de Arquitectura Web

Como instancia de UP, el proceso WebSA define la arquitectura en la fase de elaboración lo que permite establecer una base para los posteriores flujos de trabajo de diseño e implementación. Para ello, WebSA propone la introducción de una nueva perspectiva dentro del mismo flujo de trabajo de análisis, la llamada el análisis de arquitectura Web (ver Figura. 5). El análisis de arquitectura Web intenta expresar a grandes rasgos la arquitectura de la aplicación Web basándose en los requisitos no funcionales que han sido propuestos por el cliente en el flujo de trabajo de requisitos. Este análisis se establece mediante modelos que representan diferentes estilos arquitectónicos independientes del dominio del problema. Eso permite que el arquitecto Web introduzca los modelos sin la necesidad de tener en cuenta los aspectos funcionales, y realizar su trabajo de manera concurrente al realizado por los expertos en el dominio en el problema, lo que acelera el proceso de desarrollo.

La arquitectura definida en este nivel, se expresa a partir de un esfuerzo de abstracción, en el cual se han identificado un conjunto de componentes que se considera idóneo para representar la estructura y el comportamiento de una aplicación Web en una fase temprana del desarrollo.

A la hora de elegir una notación para los dos modelos de arquitectura de la fase de análisis, WebSA se ha basado en estándar UML 2.0 [95]. Sin embargo, estos estilos arquitectónicos se encuentran centrados en el dominio de la arquitectura de las aplicaciones Web, y por ello, han sido definidos mediante perfiles sobre los modelos UML, de manera que ha sido posible introducir nuevos elementos para Web, y además restringir posibles relaciones entre ellos.

Cada uno de los modelos de arquitectura se ha definido a partir de un metamodelo MOF que permite establecer las reglas de representación. Estos modelos de

arquitectura son el origen para establecer las transformaciones hasta el flujo de trabajo de diseño.

A continuación se muestra una breve explicación que pone en contexto al lector sobre los dos modelos de análisis y los dos de arquitectura: el modelo de subsistemas y el modelo de configuración. Ambos serán explicados con detalle en los capítulos 5 y 6 respectivamente.

4.5.5 Modelo de Subsistemas

Dentro del proceso de desarrollo de WebSA, el Modelo de Subsistemas (MS) es el primer modelo que se define en el análisis arquitectónico puesto que es el modelo de arquitectura que se encuentra a un mayor nivel de abstracción. En el MS se realiza una distribución o división inicial de la aplicación basada en patrones de distribución cuyas fuerzas de aplicación son los requisitos no funcionales como el rendimiento, la escalabilidad, seguridad, etc.

MS representa un estilo de arquitectura basado principalmente en el estilo *layers architecture* definido por Buchmann et al. [17]. MS propone una representación de la arquitectura de la aplicación Web basada en el concepto de subsistema Web. Cada subsistema Web representa a un conjunto de componentes definidos en una determinada capa lógica de una aplicación Web. De esta manera, un subsistema Web de una determinada capa solo puede relacionarse con otro subsistema de la misma capa o de la inmediatamente inferior.

Este modelo por su importancia y por tratarse de una de las contribuciones más importantes de este trabajo, es descrito con profundidad en el capítulo 5.

4.5.6 Modelo de Configuración

El Modelo de Configuración (MC) establece la configuración de los componentes Web, permitiendo indicar qué patrones de arquitectura se desean aplicar y cuál es la interacción con los usuarios y los sistema legados.

Para ello, el MC define un nuevo estilo arquitectónico basado en el componente Web como elemento central. Este estilo permiten representar la arquitectura de la aplicación Web, de forma genérica para cualquier tipo de dominio del problema. De este modo, es posible reutilizar un mismo MC para diferentes dominios del problema y diferentes MC pueden ser aplicados al mismo dominio.

Por otro lado, el modelo de configuración ha sido definido siguiendo la premisa de estandarización MDA. Su definición se ha basado en un perfil sobre un modelo estándar de arquitectura como es el modelo de estructura compuesta definido por UML 2.0. Así, el perfil de MC permite que los modelos de arquitectura puedan ser definidos en cualquier herramienta con soporte UML 2.0.

Este modelo por su importancia y por ser también una contribución importante de este trabajo es descrito con profundidad en el capítulo 6.

4.6 El Diseño

Dentro del proceso de desarrollo de WebSA, el flujo de trabajo de diseño proporciona una visión adecuada de la solución del problema al mismo tiempo que sigue

presentando un cierto grado de abstracción con respecto a ciertos aspectos que están ligados exclusivamente a la implementación. En este sentido, el diseño se representa a través de un artefacto denominado modelo de integración que representa la aplicación Web de forma independiente de la plataforma destino.

Como se aprecia en la Figura. 5 que representa los flujos de trabajo y los artefactos de WebSA, el modelo de integración es obtenido a través de la transformación T1 que realiza una fusión de los aspectos funcionales y arquitectónicos definidos por los artefactos del análisis. La salida del flujo de trabajo de diseño es el modelo de integración que servirá de entrada para la transformación T2 que lo convertirá en implementación.

A continuación, se realiza una breve descripción de los artefactos que participan en el diseño, la transformación T1 y el modelo de integración.

4.6.1 La Transformación T1

La transformación T1 se considera un artefacto que reside a mitad de camino entre el análisis y el diseño. Esto se debe a que dicha transformación establece el cambio del nivel de abstracción dentro del proceso entre los modelos origen definidos en el flujo de trabajo de análisis y el modelo destino definido en el flujo de trabajo de diseño.

T1 propone la fusión de los modelos funcionales definidos dentro de las propuestas de la ingeniería Web con los modelos de arquitectura definidos por WebSA. La transformación T1 está dirigida por los modelos de arquitectura que en función de los tipos de componentes, establecen en qué lugar se han de situar la funcionalidad. Como resultado de dicha fusión se obtiene el modelo de integración (MI). Tanto el origen como el destino son independientes de plataforma. Así, siguiendo la terminología definida por MDA, T1 es una transformación PIM a PIM.

Respecto al proceso WebSA, T1 se establece de forma genérica basándose en los metamodelos, permitiendo así su aplicación sobre cualquiera de sus instancias, es decir, los modelos. Sin embargo, esto no siempre cubre todas las necesidades y en ocasiones la relación entre arquitectura y funcionalidad no es homogénea. Determinados elementos funcionales no poseen la misma representación arquitectónica que otros elementos dentro de un mismo dominio. Para ello, es necesaria la definición de transformaciones que son específicas de un dominio en concreto y que extenderán las transformaciones que forman el núcleo de T1. Aquí reside el trabajo que se debe realizar en el flujo de trabajo de diseño, que como se aprecia en la Figura. 6 es un trabajo poco costoso, pero que requiere de gran conocimiento y experiencia por parte del actor llamado experto en T1.

La definición de la transformación T1 se realiza mediante el lenguaje de transformaciones declarativo UPT, definido en [76] y basado en la definición de un perfil UML para la definición de las transformaciones.

La transformación T1 se describe en profundidad en el capítulo 8.

4.6.2 Modelo de Integración

El Modelo de Integración (MI) es el artefacto dentro del proceso de WebSA que permite la representación del diseño. MI define la estructura completa de la aplicación Web mediante una definición de los componentes, su funcionalidad y las relaciones que constituyen la aplicación Web en todas sus capas. Para ello, los componentes

definidos incorporan tanto las características arquitectónicas como las funcionales. La transformación T1 establece el número de componentes en MI que es obtenido a partir de los estilos MS y MC, y además sitúa la información proveniente de la funcionalidad (atributos y operaciones) dentro de los componentes en función de su tipo.

Dentro del proceso de WebSA, MI debe ser revisado por un experto en T1, para comprobar si representa correctamente la estructura principal de la aplicación o es necesario introducir una nueva transformación que sobrescriba o extienda las existentes en T1.

Como parte de la aportación del presente trabajo, el capítulo 7 describe MI en profundidad.

4.7 La Implementación

El flujo de trabajo de implementación dentro del proceso WebSA supone la codificación en una determinada plataforma de la aplicación Web que ha sido especificada en el diseño mediante MI. Siguiendo el proceso automatizado que promulga el desarrollo dirigido por modelos, el paso del flujo de trabajo de diseño a implementación se realiza de forma automatizada mediante la transformación T2. T2 establece la integración de los aspectos dependientes de la plataforma sobre el modelo de integración (MI) obteniéndose el código para las plataformas J2EE y .NET.

Sin embargo, la obtención de la implementación no es totalmente completa. El código que se obtiene a nivel de lógica de negocio se centra en las operaciones CRUD (Create, Read, Update and Delete), es decir, alta, baja, modificación y consulta. Estas tareas son obtenidas a partir de la información proporcionada por los modelos funcionales de dominio y navegación. Sin embargo, al no especificarse la coreografía y el comportamiento interno de los componentes, una lógica de negocio más completa debe ser introducida de forma manual por el programador. En un futuro, no se descarta incorporar los modelos necesarios para obtener el código completo de la aplicación.

A pesar de ello, la implementación obtenida contiene los patrones arquitectónicos y la funcionalidad que se estableció en los flujos de trabajo de análisis y se refinó en el diseño, y el coste del flujo de trabajo de implementación se reduce a implementar algunos métodos de lógica de negocio y a extender la transformaciones T2 en aquellos casos en que es necesario refinar algunos aspectos tecnológicos específicos del sistema implementado.

4.7.1 La Transformación T2

La transformación T2 define como un conjunto de reglas modelo a texto definidas en la propuesta del estándar de la OMG MOFScript [94]. T2 realiza un recorrido de MI, y durante ese recorrido por cada uno de los componentes especificados en MI se obtiene el código correspondiente de las diferentes plataformas (en este momento J2EE y .NET).

La transformación T2 es dependiente de la plataforma destino. Así, el proceso de WebSA define diferentes transformaciones T2 (T2', T2'', etc.) para cada una de las plataformas destino. Sin embargo, hay un conjunto de reglas de transformación que

constituyen el núcleo de T2 que se pretende que sean comunes para cualquiera de las plataformas.

Según especifica el documento MDA guide [87], T2 es una transformación directa a código desde un modelo PIM, en este caso llamado MI. El paso de PIM a un PSM que representa la implementación no es un paso productivo ya que el PSM no está aportando información alguna que no pueda encontrarse en el propio código. Por ello, se ha considerado más interesante la utilización de un estándar como MOFScript que permite el recorrido del metamodelo MOF de MI, y el establecimiento de un conjunto de plantillas que especifican cual es su implementación.

4.8 Las Pruebas

El objetivo de este flujo de trabajo es verificar que la implementación funciona como se ha especificado en los requisitos. Concretamente, el propósito de las pruebas es:

- Planificar las pruebas requeridas.
- Diseñar e implementar las pruebas creando los casos de prueba
- Realizar las pruebas y analizar los resultados de cada una de ellas.

Para este flujo de trabajo no se crea ningún artefacto nuevo, y por tanto únicamente los modelos de análisis y las transformaciones de los flujos de trabajo anteriores pueden ser modificados en función de las acciones correctivas que sean tomadas.

Gracias a que el proceso de WebSA automatiza los pasos que van desde el análisis hasta la implementación, es posible situar los errores introducidos por la acción humana únicamente en los siguientes artefactos:

- Los requisitos que no hayan sido correctamente capturados por el analista.
- Los modelos de análisis no representen de forma adecuada los requisitos.
- Las transformaciones específicas de T1 no hayan sido correctamente introducidas.
- Las transformaciones específicas de T2 no hayan sido correctamente introducidas.
- El código de lógica de negocio introducido por el programador no sea correcto.

Como se especifica en la Figura. 6, el esfuerzo de la fase de transición del proceso de WebSA se centra en el flujo de trabajo Prueba. Este esfuerzo es bastante menor comparativamente al que se realiza en el proceso UP tradicional, gracias a que la parte del proceso que se realiza de forma automatizada reduce la acción humana asegurando así un menor número de errores.

4.9 Conclusiones

El presente capítulo ha presentado el proceso de desarrollo de WebSA definido como una instancia del proceso unificado de desarrollo del software (UP) pero caracterizado por ser un desarrollo dirigido por modelos y por su especificidad en la familia de las aplicaciones Web.

El proceso de WebSA, a diferencia de los procesos definidos hasta la fecha por las aplicaciones Web, se caracteriza por estar centrado en la arquitectura, para lo cual ha

Volver al índice/Tornar a l'índex

introducido la perspectiva de arquitectura en los flujos de trabajo de requisitos y análisis, así como las transformaciones para la obtención del diseño e implementación, transformaciones que están dirigidas por los componentes de arquitectura.

Su carácter genérico permite utilizar diversas aproximaciones Web para la definición de la parte funcional, lo que lo hace especialmente atractivo dentro de la comunidad de la Ingeniería Web, que carece de unos modelos estándar su representación.

Es importante destacar cómo el proceso WebSA, al utilizar el estándar MDA, pretende conseguir una reducción de coste de desarrollo, así como reducir los errores introducidos por la acción humana, con lo cual se reduce también el peso del flujo de trabajo de pruebas.

Todos artefactos definidos por WebSA que han sido introducidos en el proceso serán desarrollados en los siguientes capítulos, desde los modelos de arquitectura hasta las transformaciones T1 y T2. Por último, se presentará la herramienta llamada WebTE, que posibilita la obtención de los modelos y la ejecución de las transformaciones.

El siguiente capítulo comiénzala parte de la tesis dedicada a la descripción de los artefactos. Dentro de esta parte, el primer modelo de arquitectura del flujo de trabajo de análisis que se va a presentar es el Modelo de Subsistemas.

CAPÍTULO 5

Modelo de Subsistemas

5.1 Motivación

El análisis arquitectónico comienza con el modelado de los subsistemas los cuáles son unidades arquitectónicas localizadas en el nivel de abstracción más alto en este nivel. El modelado de subsistemas está basado en la arquitectura de capas propuesta por autores como Buchmann et al. [17] y Garlan & Shaw [38], que definen el estilo en capas como una organización jerárquica donde cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Instrumentan así una vieja idea de organización por capas que se remonta a las concepciones formuladas por Edsger Dijkstra en la década de 1960, largamente explotada en los años subsiguientes.

En la práctica, las capas son entidades complejas y están compuestas por varios subsistemas. Esto se debe a que debido a la complejidad de los sistemas, las capas se encuentran poco cohesionadas, es decir, realizan actividades independientes, llevando en este caso a una separación en las capas en grandes componentes que se encuentran cohesionados denominados subsistemas.

Con la división de la aplicación en diferentes subsistemas se consiguen mejoras como: (1) Minimizar el impacto sobre otras partes de la aplicación, reduciendo el trabajo de depuración, y facilitando el mantenimiento y mejora la flexibilidad de toda la aplicación. (2) La separación de tareas entre componentes (por ejemplo, la separación de la interfaz de usuario de lógica de negocio) incrementa la flexibilidad, el mantenimiento y la escalabilidad. (3) Mejora de la velocidad de desarrollo de la aplicación, ya que permite que diferentes equipos de desarrolladores trabajen concurrentemente en diferentes subsistemas, utilizando entre ellos interfaces bien definidos. (4) Mejora la seguridad ya que el hecho de dividirlo en diferentes capas, y establecer políticas de seguridad en cada capa, hace más difícil acceder a las capas inferiores.

Sin embargo, dicha división en subsistemas puede empeorar aspectos como: (1) cruzar demasiadas capas tiene el efecto adverso de empeorar el rendimiento. (2) Para asegurar un buen rendimiento y fiabilidad, la aplicación debe ser depurada. Definir

diferentes capas dificulta la depuración de determinadas partes. Por ello, se debe encontrar una solución de compromiso donde en función de los requisitos no funcionales se tome la decisión de dividir o no en subsistemas.

En el caso particular de la arquitectura de una aplicación Web, se establece una división en macrocomponentes denominados subsistemas Web encargados de agrupar a un conjunto de componentes Web localizados en un misma capa lógica, y además, deben estar relacionados de forma cohesionada.

Para su representación, WebSA propone el modelo de subsistemas cuyos elementos principales son los subsistemas Web y las relaciones de dependencia entre estos subsistemas. Dichas relaciones de dependencia se establecen cuando la funcionalidad de un subsistema depende a su vez de la funcionalidad proporcionada por otro subsistema de la misma capa lógica o inferior. WebSA se basa en el estándar UML para representar los elementos del modelo de subsistemas. Para ello, se define un perfil UML que permite especificar los subsistemas específicos del dominio Web y establecer las restricciones entre las posibles configuraciones.

A la hora de tomar decisiones sobre la estructuración en subsistemas Web, se considera el conjunto de requisitos no funcionales (como son el mantenimiento, la reusabilidad, la escalabilidad, la robustez, la seguridad, el rendimiento, etc.) definidos en la fase de requisitos. Todos ellos, tendrán que ser contemplados en mayor o menor medida, ya que en muchas ocasiones son contrapuestos. Por ejemplo, si se quiere cumplir un requisito no funcional como conseguir un alto rendimiento de la aplicación Web, en ocasiones va en detrimento de tener un buen mantenimiento. Por ello, es necesario tomar decisiones de compromiso, que se centren en aquellos requisitos que tengan mayor prioridad para el usuario.

Por lo tanto, la división en subsistemas se realiza mediante un proceso de refinamiento, en el cual se aplican un conjunto de patrones de distribución definidos por Renzel & Keller [100]. La aplicación de cada uno de los patrones vendrá marcada por los requisitos no funcionales, que en este caso son adaptados al dominio de las aplicaciones Web.

A continuación, se muestran las diferentes capas identificadas en una aplicación Web, sobre las que se basa el modelo de subsistemas.

5.2 El Modelo de Capas Extendido

Como se ha comentado, cada uno de los diferentes subsistemas Web está identificado con una determinada capa lógica. Por ello, es imprescindible establecer cual es la división en capas que se establece dentro de una aplicación Web.

A la hora de abordar la división en capas, tradicionalmente se ha definido un modelo de arquitectura de 3 capas para las aplicaciones cliente/servidor propuesto por Buchmann et al. [17]. Sin embargo, cada una de estas capas podrían llegarse a refinar aun más, el modelo "Seeheim-model" definido por Noack & Schienmann [84] que distingue en 6 capas diferentes una aplicación. Esta división es la que adopta WebSA para definir cada uno de los tipos de subsistemas Web. Como puede apreciarse en la Figura. 14 se procede a una división progresiva donde las flechas indican una división de una capa en 2 subcapas.

A continuación, se define la funcionalidad de cada una de las capas:

- **Interfaz de Usuario**: Capa encargada de dotar de la funcionalidad necesaria para la interacción entre el usuario y la aplicación. dicha capa contiene tanto la funcionalidad de la presentación del contenido, como la interacción o control de la interfaz de usuario. Esta interfaz se dice que actúa como un cliente pesado. A esta capa, se la puede realizar una división cliente-servidor en 2 subcapas: Presentación y Control de Usuario. De modo orientativo, poniendo un ejemplo a nivel de implementación, en una aplicación Web corresponde con un applet, un activeX o un programa flash.
- **Presentación**: Se encarga de una capa distribuida que se encarga de dotar únicamente la funcionalidad de aspecto o presentación, y recibir las peticiones del usuario que en algunos casos serán validadas. En una aplicación Web correspondería a la interfaz ofrecida por las páginas HTML, javascript, CSS.
- **Control de Usuario**: Capa que se encarga de recibir las peticiones del usuario, gestionando la navegación de la interfaz de usuario, y redireccionando las peticiones a la lógica de negocio, la cual procesará la petición y devolverá un resultado al control de usuario. De nuevo, a nivel de implementación, correspondería a un Servlet, JSP, ASP.NET, etc.

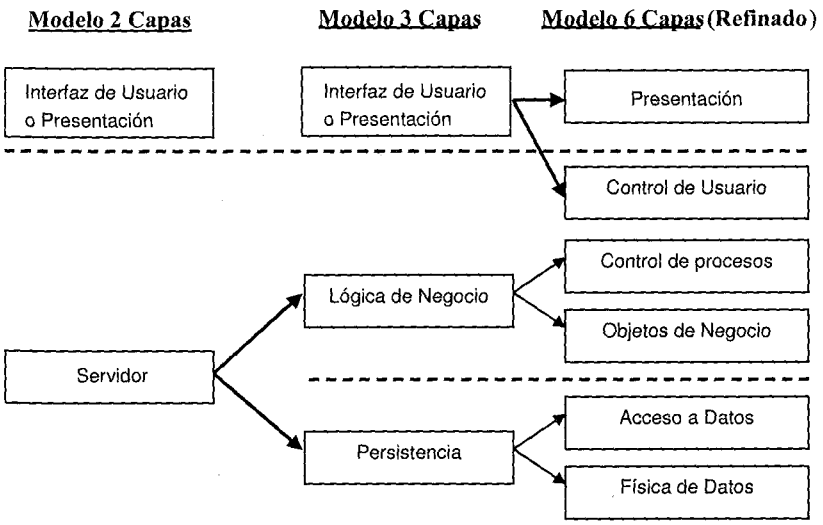

Figura. 14. Refinamiento del Modelo de Capas.

- **Servidor:** Capa que se encarga de realizar todas las tareas que necesitan cierto procesamiento interno por parte de la aplicación, independientemente de la interfaz de usuario. Esta capa puede incorporar tareas que van desde la respuesta de la interacción de la interfaz de

usuario, resolver reglas de lógica de negocio y almacenamiento de datos. Aparece en las aplicaciones Web donde la parte servidora está desarrollada de forma monolítica, en la configuración clásica Cliente-Servidor. El Cliente en este caso esta constituido o por la capa InterfazUsuario o por la capa Presentación. Mientras que el servidor se puede dividir a su vez en 2 subcapas: (1) LógicaNegocio y (2) Persistencia.

- **Lógica de Negocio**: Esta es la parte del sistema que resuelve las reglas de negocio especificadas para el dominio del problema. Es lo que se domina en ingles "middleware". En dicha capa se puede encontrar en la Web multitud de configuraciones què pueden ser distribuidas o no. Por ejemplo, dicha capa pueden ser componentes EJB distribuidos, o COM en .NET, o funciones en PHP. Por otro lado, dicha capa puede separar su funcionalidad en dos subcapas, (1) procesamiento de control y (2) objetos de negocio.
- **Procesamiento de Control**: Esta parte del sistema se encarga de atender las peticiones que se reciben del cliente, convirtiéndolas en procesos que se realizará de forma transaccional o no, de forma síncrona o asíncrona. Ejemplos de implementación son los Session de EJB o CORBA, o los componentes COM.
- **Objetos de Negocio**: En esta capa residen los elementos que contienen la representación de los objetos definidos en el dominio del problema. Ejemplos de implementación son los Entity de EJB o CORBA.
- **Persistencia**: Como su nombre indica, es la parte del sistema encargada de dotarle de almacenamiento o persistencia a los datos de la aplicación Web. Ejemplos de implementación de la persistencia puede ser cualquier base de datos con el mecanismo de acceso. La base de datos puede ser de cualquier tipo, ya sea un fichero, relacional, jerárquica o orientada a objetos. Igualmente, esta capa puede dividir su funcionalidad en 2 subcapas: acceso a datos y datos físicos.
- **Acceso a Datos**: Esta capa se encarga de contener aquellos elementos que permiten el acceso a los datos físicos desde la capa de lógica de negocio. Ejemplo de implementación serían los controladores de conexiones independientes de la base de datos.
- **Datos Físicos**: Este subsistema determina cuáles van a ser las fuentes físicas de datos que va a contener nuestro sistema. Ejemplo de implementación sería cualquier base de datos que tuviera el mecanismo de acceso independizado.

Una vez definidas cuáles son las capas que se asignan a cada uno de los subsistemas de nuestra aplicación Web, se describe el proceso de descomposición para obtener los subsistemas.

5.3 La Distribución en las Aplicaciones Web

Como se ha comentado previamente, a la hora de realizar el análisis arquitectónico se han de considerar las relaciones entre los requisitos no funcionales y la arquitectura de

la aplicación. En el caso concreto de la distribución en subsistemas expresado mediante el modelo de subsistemas, WebSA se ha basado en la propuesta de Renzel y Keller [100]. Esta propuesta definida para otro tipo de arquitecturas Cliente-Servidor, propuso la división en capas de su arquitectura en la aplicación de un conjunto de patrones de distribución donde las razones o motivaciones que justifican su aplicación, es decir, sus fuerzas generales, se basan en requisitos no funcionales.

Dependiendo de la prioridad que estableciera el usuario a cada requisito no funcional, tomará una decisión distinta que haría que se decantará hacia una u otra fuerza. Para poderlas aplicar al dominio de aplicaciones Web, WebSA ha adaptado las fuerzas propuestas por Renzel y Keller [100] considerando los requisitos no funcionales sobre los aspectos propios del dominio de las aplicaciones Web:

- **Necesidades de Negocio Frente a Complejidad en la Arquitectura**. En los últimos años, las aplicaciones Web han experimentado un gran cambio en lo que a necesidad de negocio se refiere. Cada vez más necesitan ofrecer una mayor cantidad de servicios, que van desde mostrar la información con mayor o menor estética, a poder realizar operaciones complejas como transacciones bancarias, reservas de avión y hotel, y servicios que faciliten el trabajo remoto a un cliente que posea Internet. Es por ello, que en muchas ocasiones la complejidad de la arquitectura de la aplicación está plenamente justificada. Por ejemplo, si la aplicación que se desea desarrollar es una tienda de venta por Internet, es necesario una complejidad mínima para controlar la seguridad, la escalabilidad para afrontar un gran número de clientes, el mantenimiento para mostrar las distintas ofertas, etc.
- **El Estilo de Proceso Define los Tipos de Procesamiento**. Diferentes estilos de procesamiento requieren diferente distribución. Las aplicaciones Web por lo general, requiere procesamiento transaccional en línea. De manera que los procesos que se ejecutan son por lo general síncronos, donde el componente invocador permanece a la espera que el proceso sea realizado por el invocado. Sin embargo, debido a la interacción con sistemas legados, con cada vez más frecuencia surge el procesamiento asíncrono. En otras ocasiones pueden activarse procesos por lotes pero son menos apropiados para el entorno Web.
- **Distribución Frente al Rendimiento**. Poder distribuir la aplicación Web en diferentes subsistemas, puede aportar beneficios en el rendimiento como que diferentes subsistemas puedan realizar tareas en paralelo, ciertos subsistemas de persistencia puedan tener procesamiento unido a los datos, o poder tener balanceo de carga. Sin embargo, distribuciones no adecuadas pueden causar perjuicio sobre el rendimiento final de la aplicación. Por ejemplo, un aumento desmesurado de las comunicaciones conlleva un funcionamiento más lento y un aumento de la probabilidad de en los cuellos de botella de la red. En el dominio de las aplicaciones Web, la división entre interfaz de usuario y servidor es imprescindible por la naturaleza de las aplicaciones, en este caso, se debe asumir el tiempo de comunicación como un mal necesario. Es en la parte servidora donde se pueden realizar múltiples divisiones de capas, y donde se debe procurar

que no se produzcan ni cuellos de botella, ni tampoco demasiadas llamadas a través de la red.

- **Distribución Frente a Seguridad**. El requisito de seguridad en las transacciones y comunicaciones se plantea como algo imprescindible en algunos modelos de negocio planteados en la red como los bancos o las agencias de viaje, etc. Por ello, que hay que ser cauto a la hora de aumentar la distribución, ya que un mayor número de subsistemas, los cuáles ofertan un mayor número de interfaces, incrementa el número de agujeros de seguridad. Como se ha comentado anteriormente, en las aplicaciones Web la comunicación entre interfaz y servidor es necesaria, por lo tanto, se debe procurar utilizar protocolos seguros como HTTPS o SSL. En lo que respecta a la parte servidora, se debe establecer una adecuada arquitectura de seguridad proponiendo políticas de protección de los datos mediante protocolos de encriptación, y por otro lado utilizando una adecuada política de autentificación mediante roles.
- **Distribución Frente a Consistencia**. Cuando la parte servidora de una aplicación Web se divide en componentes distribuidos puede traer problemas en su consistencia. Confiar en una única base de datos centralizada reduce los problemas de consistencia, pero en determinadas ocasiones se trabaja con sistemas legados y bases de datos distribuidas. Entonces, se debe asegurar la consistencia de los componentes que reflejan los datos en memoria, realizando transacciones distribuidas.
- **El Coste de la Distribución**. La división del sistema en subsistemas permite la distribución de los procesos dentro de la red, pero los costes de distribución, configuración, control e instalación aumentan. El coste es bajo cuando se distribuye la parte cliente en Internet, los servidores Web permite publicar los componentes de interfaz de forma económica. Sin embargo, cuando se trata de la parte servidora, tener componentes distribuidos en la lógica de negocio, en la mayoría de los casos supone tener el control de las comunicaciones mediante un ORB o broker, un servicio de transacciones distribuidas, de persistencia, etc. Los servidores de aplicaciones abaratan los costes ya que ofertan todos estos servicios de forma conjunta, sin embargo, para determinadas aplicaciones de poca complejidad no es recomendable utilizar dicha distribución.
- **Distribución Frente Reusabilidad**. Localizar la funcionalidad del sistema en diferentes macrocomponentes o subsistemas, permite que se reutilicen ciertas partes y reducir el tamaño del código a implementar. Un ejemplo evidente se produce cuando el subsistema es de la parte servidora, en este caso la parte cliente se beneficia reduciendo el tamaño de su código. Sin embargo, los datos del cliente se deben enviar al servidor y el servidor debe permitir manejar peticiones de múltiples clientes.

Así pues, inevitablemente la adopción de una política de distribución en la aplicación conlleva perjuicios como: mayor complejidad, menor seguridad, menor consistencia, mayor coste de distribución, peor tratamiento del proceso por lotes o mayor probabilidad de cuellos de botella. Pero también aporta beneficios como:

representación de las necesidades de negocio, distribución, reusabilidad, cargas de trabajo balanceadas y un mejor tratamiento del proceso offline e interactivo.
En conclusión, la justificación del uso o no de los mecanismos de distribución se basa en la prioridad que el usuario proporciona a los requisitos no funcionales,.

5.4 El Modelo de Subsistemas

Una vez se han establecido cuáles son los requisitos no funcionales que determinan cual debe ser el modelado de subsistemas, se presenta su representación mediante un modelo basado en el estándar UML.

El Modelo de Subsistemas (MS) representa el conjunto de subsistemas Web que componen nuestra aplicación. Este modelo es un estilo arquitectónico basado en el patrón *layers architecture* definido por Buchmann et al. [17]. Un subsistema encapsulará una capa o parte de una capa, de manera que cada uno de los subsistemas podrá relacionarse con los subsistemas de capas inferiores o que estén en su mismo nivel. La relación establecida entre dos subsistemas será una relación de uso, indicando que un subsistema requiere de la funcionalidad de otro subsistema para realizar su tarea.

La peculiaridad que propone este modelo respecto a cualquier modelo de subsistemas propuestos con anterioridad, es su definición en el dominio de las aplicaciones Web. Otorgándole al MS la posibilidad de definir una topología de los diferentes subsistemas identificados en las aplicaciones Web (ver sección 5.2), cada uno de los subsistemas se representa mediante un estereotipo en el perfil UML definido para el MS. Por otro lado, se establece un conjunto de restricciones OCL que restringen las posibles relaciones entre los diferentes subsistemas Web. Las restricciones aseguran que el analista especifica una correcta descomposición en subsistemas de una aplicación Web.

A continuación, para especificar y formalizar el MS se muestran los siguientes aspectos: (1) sus elementos de modelado, (2) el metamodelo de subsistemas que formalizará las instancias del modelo, (3) el proceso de descomposición definido para modelarlo, (4) el perfil UML definido para su estandarización y (5) finalmente, ejemplos o casos de estudio en los que se aplica el MS.

5.4.1 Elementos de Modelado

El modelo de subsistemas es el más sencillo de los estilos de arquitectura propuestos por WebSA, y se especifica mediante únicamente dos elementos:

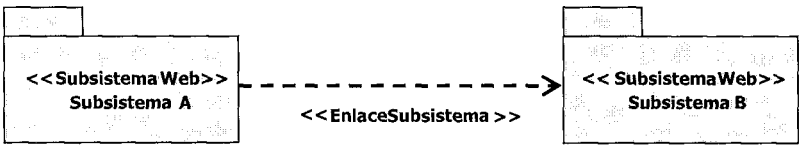

Figura. 15. Elementos de Modelado de MS.

- **SubsistemaWeb**: Nombre que recibe el macrocomponente o subsistema que soporta la funcionalidad asignada a parte o a la totalidad de una determinada capa lógica de una aplicación Web. Se representa mediante un estereotipo de un elemento paquete de UML. (Ver Figura. 15). *SubsistemaWeb* se define como una clase abstracta, es decir, a partir de él se establecerá una jerarquía de clases especializadas que representarán a los diferentes subsistemas correspondientes a cada una de las capas de la aplicación Web. Por ejemplo, se definen los subsistemas de *Presentacion*, *ControlUsuario*, *LogicaNegocio*, *Persistencia*, etc. hasta 10 tipos distintos *SubsistemasWeb*, los cuáles han sido especificados en la sección 5.2.
- **EnlaceSubsistema**: Representa una relación de dependencia de uso entre diferentes dos instancias de SubsistemaWeb. En el modelo de la Figura. 15, el enlace de subsistema indica que el subsistema A depende del subsistema B para poder completar su funcionalidad. La notación utilizada es una flecha discontinua, siguiendo la misma notación que la relación de dependencia de UML.

5.4.2 Metamodelo de Subsistemas

Para formalizar los elementos del MS y las relaciones existentes entre ellos, se ha definido el metamodelo de subsistemas. Como ya se ha especificado en el capítulo 3, un metamodelo define el lenguaje para expresar el modelo. En este caso, el metamodelo de subsistemas define los elementos del MS y las posibles relaciones entre ellos. La Figura. 16, muestra las clases del metamodelo de subsistemas. Contiene dos elementos principales *SubsistemaWeb* y *EnlaceSubsistema* y las relaciones entre ellos. Además, se establece una asociación llamada *SubsistemaProveedor* donde el *SubsistemaWeb* provee cierta funcionalidad a otros subsistemas por medio del *EnlaceSubsistema*. Por otro lado, se establece la relación *SubsistemaCliente* donde el *SubsistemaWeb* es el que obtiene la funcionalidad por medio de la relación *EnlaceSubsistema*.

La Figura. 16, muestra también que la metaclase SubsistemaWeb es una clase abstracta, es decir, no se define ninguna instancia de esta clase obligando a utilizar una de las clases especializadas para su instanciación en el MS. A partir de la clase SubsistemaWeb, extienden un conjunto de clases especializadas, formando una jerarquía arbórea que se inicia con los subsistemas que corresponden a las capas más complejas, *InterfazUsuario* y *Servidor* y va dividiéndose en cada uno de los subsistemas correspondientes a las capas más sencillas. De esta manera, se llega al nivel hoja de la jerarquía, donde se encuentran las capas definidas en el modelo de refinado de 6 capas definido por [84]. Esto permite que las propiedades de los

subsistemas más generales sean heredadas por los subsistemas más sencillos, como por ejemplo, las restricciones OCL.

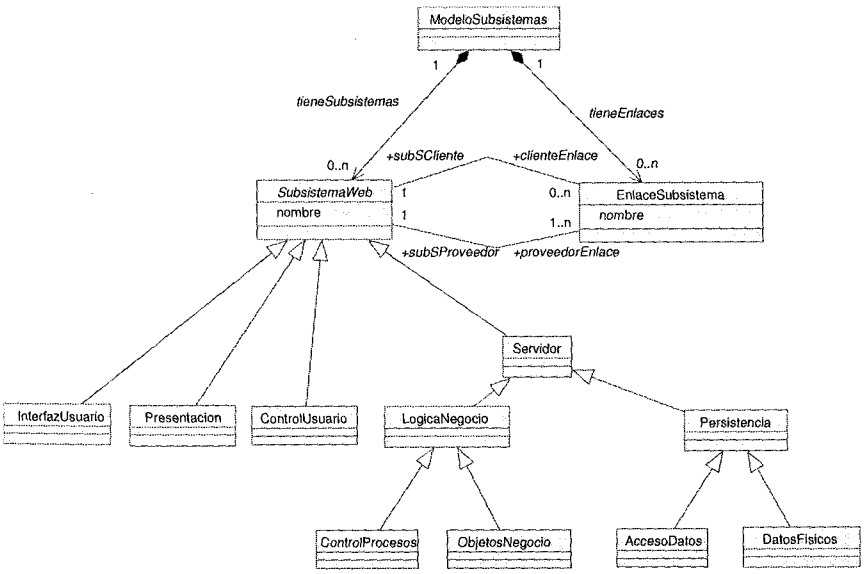

Figura. 16. Metamodelo de Subsistemas

Una vez especificado el modelo de clases que corresponde con el metamodelo de subsistemas, se establecen las restricciones entre las relaciones de los diferentes elementos. Por ejemplo, el modelo de clases permite establecer cualquier tipo de relación entre los diferentes subsistemas, ya que se establecen las relaciones al nivel de la clase abstracta *SubsistemaWeb*. Para ello, se definen las restricciones en el lenguaje OCL que establecen las posibles relaciones entre las metaclases.

A continuación, se muestran las restricciones OCL definidas para el metamodelo de subsistemas. En ellas se indican las posibles relaciones entre los diferentes subsistemas definidos en MS. Por ejemplo, se indica que un cliente es cliente de otro cuando tiene el rol *clienteEnlace* dentro de una relación *EnlaceSubsistema*. Y se indica que un cliente es proveedor de otro subsistema cuando tiene el rol *proveedorEnlace* dentro de una relación de *EnlaceSubsistema*.

5.4.2.1 Restricciones para el Subsistema InterfazUsuario

El SubsistemaWeb InterfazUsuario puede ser cliente de otro subsistema Servidor, LogicaNegocio o un subsistema de su mismo tipo. Y solo puede ser proveedor de otro subsistema de su mismo tipo.

context InterfazUsuario
inv: self.clienteEnlace ->forAll(s | s.subSProveedor oclIsTypeOf (Servidor)) or
self.clienteEnlace->forAll (l | l.subSProveedor.oclIsTypeOf (LogicaNegocio)) or
self.clienteEnlace->forAll (i| i.subSProveedor.oclIsTypeOf (InterfazUsuario))

inv: self.proveedorEnlace->forAll (u|u.subSCliente.oclIsTypeOf (InterfazUsuario))

5.4.2.2 Restricciones para el Subsistema Presentación

El subsistema Web Presentación puede ser cliente de un subsistema Servidor, Lógica de negocio, ControlUsuario o con un subsistema de su mismo tipo. Y solo puede ser proveedor de otro subsistema de su mismo tipo.

context Presentacion
inv: self.clienteEnlace ->forAll(s | s. subSProveedor.oclIsTypeOf (Servidor)) or
self.clienteEnlace->forAll (l | l.subSProveedor.oclIsTypeOf (LogicaNegocio)) or
self.clienteEnlace->forAll (i| i.subSProveedor.oclIsTypeOf (ControlUsuario))
inv: self.proveedorEnlace->forAll (u|u.subSCliente.oclIsTypeOf (Presentacion))

5.4.2.3 Restricciones para el Subsistema ControlUsuario

El subsistema Web ControlUsuario puede ser cliente de un subsistema Servidor, Lógica de negocio y de su mismo tipo. Y puede ser proveedor del subsistema Presentación, y de otro subsistema de su mismo tipo.

context ControlUsuario
inv: self.clienteEnlace ->forAll(s | s. subSProveedor.oclIsTypeOf (Servidor)) or
self.clienteEnlace->forAll (l | l. subSProveedor.oclIsTypeOf (LogicaNegocio)) or
self.clienteEnlace->forAll (c | c. subSProveedor.oclIsTypeOf (ControlUsuario))
inv: self.proveedorEnlace->forAll (i| i.subSCliente. oclIsTypeOf (ControlUsuario))
or self.proveedorEnlace->forAll (u|u.subSCliente.oclIsTypeOf (Presentacion))

5.4.2.4 Restricciones para el Subsistema Servidor

El subsistema Web Servidor puede ser proveedor de un subsistema ControlUsuario, Presentación e InterfazUsuario. Y solo puede ser cliente de otro subsistema de su mismo tipo.

context Servidor
inv: self.proveedorEnlace->forAll (i| i.subSCliente. oclIsTypeOf (ControlUsuario))
or self.proveedorEnlace->forAll (u|u.subSCliente.oclIsTypeOf (Presentacion))
or self.proveedorEnlace->forAll (p|p.subSCliente.oclIsTypeOf (InterfazUsuario))
inv: self.clienteEnlace ->forAll(s | s.subSProveedor.oclIsTypeOf (Servidor))

5.4.2.5 Restricciones para el Subsistema LogicaNegocio

El subsistema Web LogicaNegocio además de las restricciones heredadas por servidor. Añade restricciones para ser cliente de Persistencia, AccesoDatos u otro subsistema de su mismo tipo.

context LogicaNegocio
inv: self.clienteEnlace->forAll(s | s.subSProveedor.oclIsTypeOf (Persistencia)) or
self.clienteEnlace->forAll(a | a.subSProveedor.oclIsTypeOf (AccesoDatos)) or
self.clienteEnlace->forAll(l | l.subSProveedor.oclIsTypeOf (LogicaNegocio))

5.4.2.6 Restricciones para el Subsistema Persistencia

El subsistema Web Persistencia, añade a las restricciones heredadas, las restricciones que le permitan ser cliente y proveedor de otro subsistema de persistencia.

context Persistencia
inv: self.proveedorEnlace->forAll (p| p.subSCliente.oclIsTypeOf (Persistencia))
inv: self.clienteEnlace->forAll (c| c.subSProveedor.oclIsTypeOf (Persistencia))

5.4.2.7 Restricciones para el Subsistema ControlProcesos

El subsistema Web ControlProcesos añade a las restricciones heredadas, otras restricciones para (1) ser cliente del subsistema ObjetosNegocio o (2) ser cliente y proveedor de otro subsistema de ControlProcesos.

context ControlProcesos
inv: self.clienteEnlace->forAll(p|p.subSProveedor.oclIsTypeOf (ObjetosNegocio))
or self.clienteEnlace->forAll (c| c.subSProveedor.oclIsTypeOf (ControlProcesos))
inv: self.proveedorEnlace->forAll(o|o.subSCliente.oclIsTypeOf(ControlProcesos))

5.4.2.8 Restricciones para el Subsistema ObjetosNegocio

El subsistema Web ObjetosNegocio añade a las restricciones heredadas, otras restricciones para (1) ser proveedor del subsistema ControlProcesos y (2) para ser cliente y proveedor de otro subsistema de ObjetosNegocio.

context ControlProcesos
inv: self.proveedorEnlace->forAll(p|p.subSCliente.oclIsTypeOf(ControlProcesos))
or self.proveedorEnlace->forAll (o| o.subSCliente.oclIsTypeOf (ObjetosNegocio))
inv: self.clienteEnlace->forAll(n|n.subSProveedor.oclIsTypeOf (ObjetosNegocio))

5.4.2.9 Restricciones para el Subsistema AccesoDatos

El subsistema Web AccesoDatos añade a las restricciones heredadas, otras restricciones para (1) ser cliente del subsistema DatosFisicos y (2) para ser cliente y proveedor de otro subsistema de AccesoDatos.

context AccesoDatos
inv: self.clienteEnlace->forAll (p| p.subSProveedor.oclIsTypeOf (DatosFisicos))
or self.clienteEnlace->forAll (c| c.subSProveedor.oclIsTypeOf (AccesoDatos))
inv: self.clienteEnlace->forAll (o| o.subSProveedor.oclIsTypeOf (AccesoDatos))

5.4.2.10 Restricciones para el Subsistema DatosFisicos

El subsistema Web ObjetosNegocio añade a las restricciones, otras restricciones para (1) ser proveedor del subsistema AccesoDatos y para ser cliente y proveedor de otro subsistema de DatosFisicos.

context ControlProcesos
inv: self.proveedorEnlace->forAll (a| a.subSCliente. oclIsTypeOf (AccesoDatos))
or self.proveedorEnlace->forAll (d| d.subSCliente.oclIsTypeOf (DatosFisicos))

inv: self.clienteEnlace->forAll (p| p.subSProveedor.oclIsTypeOf (DatosFisicos))

5.5 El Método: Proceso de Descomposición en Subsistemas

Para realizar el proceso de descomposición en subsistemas, WebSA se basa en los patrones de arquitectura llamados patrones de distribución definidos por Renzel & Keller [100] que permiten obtener una distribución horizontal en capas basándose en los requisitos no funcionales. Posteriormente, en base a la cohesión funcional de dichas capas, se establece una distribución vertical en uno o más subsistemas.
WebSA propone, para la división en capas, el uso de cinco patrones de distribución. La aplicación de estos patrones divide cada una de las capas en dos subcapas cliente-servidor, que pueden observarse en la Figura. 17. Cada uno de estos patrones tiene un conjunto asociado de fuerzas que se corresponden con el conjunto de requisitos no funcionales propuestos para la aplicación Web. Todos los patrones de distribución, tratan de obtener la mejor distribución por capas para un sistema dado.

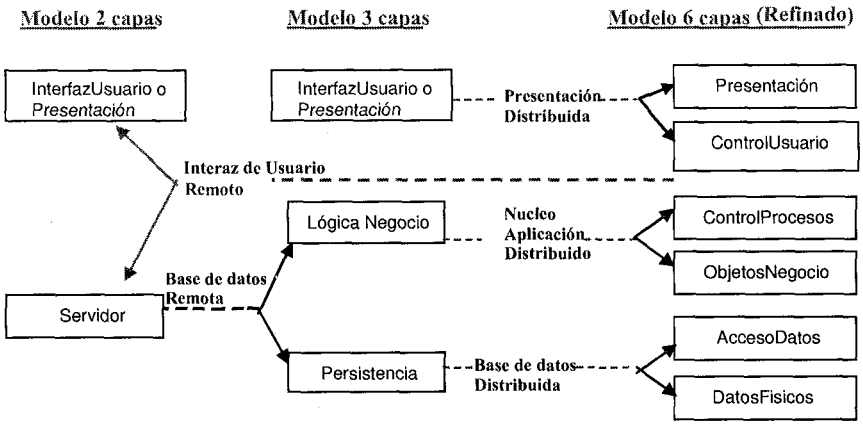

Figura. 17. Vista General de la División en Capas por los Patrones de Distribución

Ahora se describe el proceso de descomposición horizontal que permite la construcción del modelo de subsistemas, asociando un subsistema para cada una de las capas identificadas. El proceso esta constituido por cuatro pasos de refinamiento:

5.5.1 La División de la Interfaz de Usuario

El proceso comienza asumiendo una configuración mínima Interfaz de usuario/Servidor. Por su naturaleza, una aplicación Web requiere que el cliente sea ejecutado de forma remota al servidor. Sin embargo, se ha de decidir qué funcionalidad reside en el cliente y qué funcionalidad reside en el servidor. La forma de decidir por dónde se realiza la división funcional se realiza eligiendo entre dos patrones de distribución: *Interfaz de Usuario Remoto o Presentación distribuida.*

5.5.1.1 Interfaz de Usuario Remoto

Establece una separación de la funcionalidad entre la interfaz de usuario y el servidor, en la que la interfaz de usuario es un cliente pesado, es decir, posee una interfaz de usuario gráfico muy rica y permite realizar un procesamiento desconectado. Un ejemplo de este tipo de cliente en las aplicaciones Web puede ser implementado con applets Java, ActiveX o Flash. Las fuerzas o motivaciones para su aplicación son las siguientes:

- Los requisitos no funcionales favorables son: (1) tener una interfaz rica con alta usabilidad, (2) reduce el tráfico ya que el cliente establece menos comunicaciones con el servidor y (3) permite el estilo de procesamiento desconectado.
- Los requisitos no funcionales desfavorables son: (1) una menor seguridad que la presentación distribuida ya que el cliente recibe mayor cantidad de código que puede ser atacado y (2) tiene un coste alto de distribución, tanto por el código que necesita descargarse el cliente, y debido a que en algunas ocasiones requiere que el cliente tenga ciertos programas instalados (plugins) en el navegador para su ejecución.

5.5.1.2 Presentación Distribuida

Realiza la división cliente-servidor a nivel de los componentes de presentación. Una parte de los componentes de presentación son empaquetados como una unidad de distribución y son ejecutados en la parte cliente, es lo que se denomina un cliente ligero. La otra parte de componentes residen con el resto del sistema y son ejecutados en la parte servidora. Un ejemplo típico de implementación en una aplicación Web es interfaz HTML, con funcionalidad Javascript y estilo en páginas CSS. Sus fuerzas son las siguientes:

- Los requisitos no funcionales favorables son: (1) posee un bajo conste de distribución, (2) tiene mayor seguridad que la interfaz remota al descargar menos código en la máquina cliente y (3) posee un mejor nivel de reutilización
- Los requisitos no funcionales desfavorables son: (1) su baja usabilidad en una interfaz de usuario bastante pobre, (2) menor rendimiento debido a un mayor tráfico con el servidor y (3) no permite el procesamiento desconectado.

Una vez se ha elegido cual es la interfaz de usuario que se desea tener para nuestra aplicación Web, el siguiente paso consiste en dividir el servidor en subsistemas.

5.5.2 La División del Servidor

Actualmente, la aplicación está dividida en un cliente o interfaz de usuario y un servidor. Sin embargo, el arquitecto puede determinar una división del servidor en dos capas y obtener así la clásica arquitectura de 3 capas. El siguiente paso consiste en elegir si se aplica el patrón *Base de datos remota* o no.

5.5.2.1 Base de Datos Remota

Este patrón consiste en dividir la capa servidora por debajo de los componentes de acceso a la base de datos, en dos subcapas cliente-servidor. El servidor queda

entonces dividido en dos subcapas: (1) la lógica de negocio y (2) la persistencia. La persistencia es proporcionada por una base de datos remota, que es accedida por medio de algún protocolo de acceso a base de datos desde la lógica de negocio. Sus fuerzas son las siguientes:

- Los requisitos no funcionales favorables son:(1) su mayor facilidad a la hora de desarrollar la parte servidora, (2) la persistencia obtiene una reducción de carga de trabajo y (3) permite que los datos sean compartidos por múltiples clientes.
- Los requisitos no funcionales desfavorables son: (1) no permite un adecuado procesamiento por lotes, ya que se ha de ejecutar las órdenes desde la parte cliente y supone un congestionamiento de la carga, (2) debido a un alto tráfico en la red el rendimiento puede bajar y (3) la arquitectura de tres capas promueve poco la reutilización.

5.5.3 La división de la Lógica de Negocio

Una vez obtenida una arquitectura de 3 capas, si el arquitecto lo considera conveniente puede continuar el proceso de división de la aplicación Web. En este caso, se procede a dividir la funcionalidad de la lógica de negocio en 2 subcapas mediante el patrón *Núcleo de aplicación distribuido*.

5.5.3.1 Núcleo de Aplicación Distribuido

La aplicación de este patrón consiste en la división de la lógica de negocio en dos subcapas cliente-servidor. El servidor queda dividido en 2 capas: (1) Control de procesos y (2) Objetos de negocio. El arquitecto decide separar una parte de la funcionalidad del núcleo de la aplicación, que se encarga de recibir las peticiones de la interfaz de usuario y localizarla en el nodo cliente. En la otra capa localiza la representación de los objetos del dominio en el servidor. El corte entre control de procesos y objetos de negocio se comunica mediante un mecanismo de procedimientos de llamada remoto, el cual propaga los servicios invocados desde el cliente y permite al servidor devolver los resultados. En el dominio de las aplicaciones Web, se pueden encontrar ejemplos basados en plataformas como J2EE o CORBA, donde en la parte cliente existen componentes de proceso (llamados session), que se encargan de recibir las llamadas de la interfaz de usuario, iniciar las transacciones y enviar peticiones a los componentes entidad (llamados Entity), que representarían la parte servidora. Las fuerzas o motivos para la aplicación de este patrón son las siguientes:

- Los requisitos no funcionales favorables son: (1) es una solución muy flexible que permite optimizar el uso del hardware sobre el que trabaja la aplicación, (2) mayor escalabilidad, ya que el uso de una capa de control de procesos permite incrementar el número de clientes de interfaz de usuario.
- Los requisitos no funcionales desfavorables son: (1) desafortunadamente supone el reto de mayor dificultad para el diseñador, ya que el rendimiento de la aplicación depende sus decisiones a la hora de establecer el corte de funcionalidad, (2) es la solución que tiene mayor número de agujeros de seguridad, con lo que se debe incrementar los mecanismos de seguridad.

5.5.4 La División de la Persistencia

Independientemente de la división de la lógica de negocio, se puede promover una división de la capa de persistencia, que permita distribuir los datos entre diferentes máquinas. Esta división se realiza mediante el patrón de distribución *Base de datos distribuida.*

5.5.4.1 Base de Datos Distribuida

Este patrón aplica una división cliente servidor dentro de los componentes de base de datos, de forma que la base de datos queda dividida en (1) la capa de acceso a datos, (2) la capa física de datos. Se separa entonces en una parte cliente la capa de acceso de datos que consistirá en un controlador o manejador de fuentes de conexión, que maneja un protocolo de base de datos y proporciona los servicios que permiten controlar el manejo de las conexiones, ejecución de operaciones de procedimientos almacenados, obtener los resultados y manejo de transacciones de base de datos. Por otro lado, quedaría la base de datos física que podría encontrarse ubicada en cualquier lugar de la red. Las fuerzas de este patrón son las siguientes:

- Los requisitos no funcionales favorables son: (1) Esta solución es útil cuando los datos se encuentren geográficamente dispersos, por ejemplo un sistema de bancos que tenga distribuido los datos en las diferentes sucursales, (2) permite una mayor escalabilidad al poder distribuir las bases de datos en diferentes máquinas.
- Los requisitos no funcionales desfavorables son: (1) La penalización en el rendimiento, no es adecuado para las aplicaciones Web con un gran número de clientes, (2) la seguridad es menor que un sistema centralizado.

Una vez identificadas que capas son adecuadas para la aplicación Web, se puede asociar a cada una de las capas un subsistema. Sin embargo, como se ha mencionado anteriormente, en las aplicaciones Web complejas, algunas capas no se encuentran bien cohesionadas debido a que realizan tareas independientes. Por ello, se propone un conjunto de posibles casos en los que el arquitecto ha de proponer la división de un subsistema referido a una capa lógica, en más de un subsistema:

- A nivel de interfaz de usuario (y sus subcapas), la división puede deberse a que se propone una aplicación Web en la que se dispone de diferentes perfiles de usuario. Cada perfil de usuario puede tener requisitos no funcionales distintos, por ejemplo la vista de usuario de un sitio Web, tiene requisitos muy diferentes a la vista del administrador de la Web.
- A nivel de lógica de negocio y sus subcapas, puede proponerse la división de la arquitectura debido a que en determinadas ocasiones el procesamiento que se requiere es diferente. Por ejemplo, se requiere de un procesamiento que permita la interacción con la interfaz de usuario de la aplicación Web, y otro procesamiento que requiera la ejecución de procesos por lotes.
- A nivel de persistencia y sus subcapas, la división vertical en diferentes subcapas puede deberse a la división de los datos en distintas máquinas que se encuentren situadas en lugares geográficos diferentes.

5.6 El perfil UML del Modelo de Subsistemas

El principal objetivo de un perfil UML es proporcionar un mecanismo sencillo que permita adaptar los elementos de un metamodelo MOF definido para el modelo de subsistemas, a los elementos que han sido propuestos para representar los modelos de UML. En este sentido, el caso particular del perfil del Modelo de Subsistemas, consiste en adaptar los elementos propuestos en el metamodelo de subsistemas, al metamodelo especificado por UML. De este modo, el perfil podrá especificar sus elementos en cualquier herramienta comercial que tenga soporte de UML.

A la hora de realizar dicha adaptación, MS parte de un metamodelo de subsistemas definido en MOF, lo que permite establecer que cada una de las metaclases definidas en el metamodelo pueda convertirse en un elemento del perfil llamado estereotipo. Un estereotipo adapta al metamodelo de UML introduciendo nuevos términos, nueva sintaxis, nueva semántica y nuevas restricciones, o añadiendo posteriormente información por medio de atributos o valores definidos.

Por otro lado, debido a que el perfil es un mecanismo de extensión ligera que respecta el metamodelo de UML, no se permite la modificación de ninguna de las relaciones establecidas en UML. Para hacer corresponder las relaciones del metamodelo de MS en el perfil, deben representarse por medio de restricciones OCL. Por otro lado, cada una de las restricciones establecidas por el metamodelo son mapeadas también para su especificación en el perfil.

Para especificar como una metaclase es extendida por un estereotipo del perfil, UML 2.0 ha incorporado una relación denominada *extensión*. Una extensión es una relación binaria establecida entre un elemento estereotipo, el cual es dependiente de un único elemento del metamodelo. La notación de la extensión se especifica por medio de una flecha con la cabeza rellena (ver Figura. 18). Además, la extensión puede ser marcada con la etiqueta *{required}*, cuando se desea que la instancia del estereotipo o la de uno sus hijos sustituya siempre a la instancia de la metaclase extendida. En el perfil SM, el estereotipo SubsistemaWeb es creado mediante la extensión con la marca *{required}* desde la metaclase *Package* de UML.

Por otro lado, los diferentes tipos de subsistemas definidos en MS para las correspondientes capas, son reflejados en el perfil por medio de un conjunto de estereotipos que son heredados del estereotipo SubsistemaWeb. En este sentido, y siguiendo los elementos del metamodelo de subsistemas, el perfil define nueve tipos de estereotipos subsistemas llamados: *InterfazUsuario, Servidor, LogicaNegocio, Presentación, ControlDialogo, ControlProceso, ObjetosNegocio, AccesoDatos y DatosFisicos.*

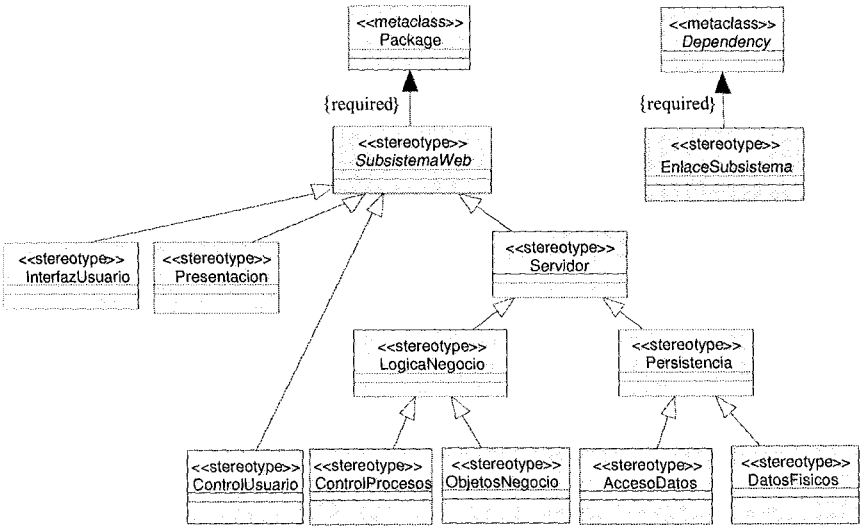

Figura. 18. El Perfil de SM.

Como se aprecia en la Figura. 18, el *SubsistemaWeb* está relacionado con la metaclase *Paquete* por medio de la relación extensión con la etiqueta *{required}*, pero debido a que el *SubsistemaWeb* es una clase abstracta, esto fuerza a que siempre el perfil tenga que instanciar una de sus subclases concretas (p.e. *Presentación*), la cual ha de estar vinculada con una instancia de *Package*. En el apéndice I se muestra con detalle el perfil de SM con los atributos del estereotipo, su semántica y sus restricciones OCL.

5.7 Ejemplos de Uso: Diferentes Tipos de Subsistemas

Una vez definidos los elementos del MS, su proceso y su notación para poder aplicarlo a las diferentes aplicaciones Web, seguidamente se muestran tres casos de estudio en los que se aplica el proceso de definición de MS siguiendo los diferentes requisitos no funcionales.

5.7.1 MS de un Chat Corporativo

Se solicita la realización de una aplicación Web para la Intranet de una gran empresa telefonía móvil. Dicha aplicación requiere la comunicación mediante un Chat entre los clientes y los proveedores. Sus requisitos no funcionales son: (1) Una interfaz amigable con alta usabilidad, que permita tratar velozmente el texto, admita conversaciones con más de una persona, creación de salas, opciones de personalización de temas, etc. (2) Alta velocidad de procesamiento en el texto de las conversaciones. (3) Procesamiento muy sencillo ya que solamente requiere que el texto se muestre a los interlocutores, este texto no debe ser almacenado.

a. SM chat corporativo b. SM agencia de viajes

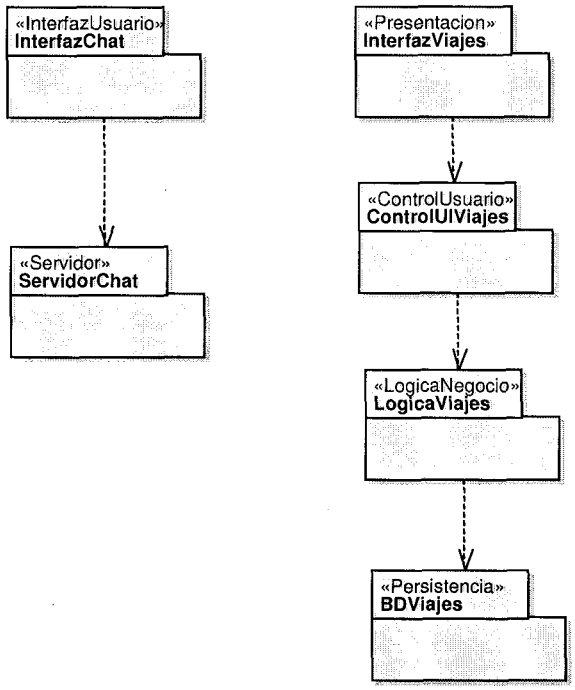

Figura. 19. MS de Chat Corporativo y Agencia Inmobiliaria.

Partiendo de estos requisitos se aplicarán los siguientes pasos del proceso de descomposición. Lo primero es aplicar la división entre interfaz de usuario y servidor, y para ello, el arquitecto se inclina por el patrón *Interfaz remoto*. Esto se debe a que lo principal en esta aplicación es disponer de una interfaz rica, y para ello se sitúa en la máquina cliente funcionalidad de control de usuario. El segundo paso consiste en aplicar la división del servidor, y para ello, se valora si es interesante separar la lógica de negocio y la persistencia. En este caso, el arquitecto decide que no lo es, ya que dicha aplicación no necesita almacenamiento y por lo tanto no se aplica el patrón *Base de datos remota*. En este punto se acaba el proceso de descomposición horizontal. En la descomposición vertical, la aplicación debido a su sencillez no requiere la separación de ninguna de las capas. De manera que el SM del Chat tiene la estructura cliente-servidor de la Figura. 6.a

5.7.2 MS de una Agencia de Viajes Virtual

Este ejemplo esta basado en la propuesta realizada en el Taller Model-Driven Web Engineering (MDWE 2006) realizado en la conferencia internacional ICWE 2006, y

especificado en WebSA en el articulo [71] y en la revista Journal Web Engineering [75].

Se propone la realización de una aplicación Web para una agencia de viajes, solicitando que la aplicación sea destinada principalmente para publicitar sus diferentes ofertas de viviendas por Internet. Para ello, a parte de los requisitos funcionales, se piden siguientes requisitos no funcionales: (1) Una interfaz Web sencilla y barata de distribuir, que funcione en cualquier tipo de navegador, ya sea Browser, PDA, TV, etc. (2) La aplicación Web debe leer la información sobre los viajes, desde una base de datos existente, la cual está poblada por otra aplicación de escritorio para la gestión de viviendas.

Partiendo de estos requisitos se aplican los siguientes pasos en el proceso de descomposición. Lo primero es aplicar la división de interfaz, y para ello, el arquitecto se inclina por el patrón *Presentación distribuida* que permite tener una interfaz sencilla y fácilmente distribuible por los diferentes tipos de navegadores. En la segunda división del servidor, se aplican el patrón *Base de datos remota* ya que al tener que acceder a una base de datos existente, se ha de separar la lógica de negocio de la persistencia. Aquí termina la descomposición en subsistemas, debido a que la división de la lógica de negocio no es necesaria por su simplicidad. Por último indicar, que no se propone ninguna separación de los subsistemas ya que presentan una cohesión adecuada. Una vez establecidos los subsistemas se establecen las dependencias, el SM de la agencia se puede ver en la Figura. 19.b.

5.7.3 Tienda Electrónica : Petstore

Para poder mostrar la capacidad y expresividad del proceso de desarrollo WebSA se ha utilizado como caso de estudio un conocido ejemplo que consiste en una aplicación Web de comercio electrónico, como es la tienda de animales o Petstore[27], definido por SUN microsystems [115].

Petstore utiliza las mejores prácticas y guías de diseño en el uso de los componentes Web distribuidos. Además, ha sido utilizado por diferentes plataformas como J2EE y .NET para realizar pruebas de rendimiento. De esta manera se trata de una aplicación conocida y se puede disponer de mucha documentación al respecto.

La característica principal de esta aplicación es que a nivel funcional tiene los mismos requisitos que una aplicación típica de comercio electrónico. Sin embargo, a nivel arquitectónico, Petstore hace un uso intensivo de patrones de arquitectura y diseño, esto hace que sea especialmente atractiva para ser representada por WebSA y permite hacer una representación arquitectónica lo suficientemente compleja y didáctica.

Como en un sitio típico de comercio electrónico, Petstore presenta al cliente un catálogo de productos. El cliente selecciona los animales en los cuáles está interesado y los coloca en el carro de la compra. Cuando el usuario ha seleccionado todos los animales que deseaba e indica que está preparado para comprar lo que contiene el carro, la aplicación le muestra una factura, es decir una lista con los animales seleccionados, la cantidad de cada uno de ellos, el precio y el coste total. El cliente revisa o cancela el pedido. Finalmente, el cliente acepta el pedido, y el cliente provee la tarjeta de crédito para pagar y una dirección de envió.

[27] Petstore: traducción al inglés de tienda de animales

Además de presentar la vista del cliente, existen dentro de la aplicación otro usuario potencial, el *administrador*. Este es responsable de mantener el inventario y de realizar otras tareas de control.

Partiendo de estos requisitos se aplican los siguientes pasos en el proceso de descomposición. Petstore posee dos vistas, una para mostrar la tienda en Internet para el cliente y otra para la administración del dueño de la tienda. Se piden los siguientes requisitos no funcionales: (1) Realizar una interfaz de usuario del sitio Web, que sea distribuible por Internet para el acceso de los clientes de la tienda. (2) Una interfaz en intranet sencillo, que permita la administración del dueño de la tienda, con tareas de introducción y consulta de datos. (3) Ambos interfaces han de mostrar los mismos artículos simultáneamente. (4) Una lógica de negocio para el sitio Web que de soporte a un gran número de clientes, alta escalabilidad, realización de operaciones complejas como el pedido de artículos. (5) Una lógica de negocio sencilla para el alta, baja y modificación en la base de datos. (6) Una base de datos que tenga los datos comunes de ambas vistas.

Una vez se han determinado los requisitos no funcionales, comienza el proceso de descomposición.

El primer paso es la división de la interfaz de usuario, se aplica el patrón *Presentación distribuida* para ambas vistas. Por un lado, la vista de Internet requiere de una alta distribución con lo que es conveniente una interfaz ligera. Por otro lado, la administración requiere de una interfaz sencilla y sin pretensiones para introducir y consultar datos y es más conveniente la interfaz de la presentación distribuida.

El segundo paso es la división del servidor, que consiste en elegir si se aplica o no el patrón *Base de datos distribuida.* En este caso, debido a que la base de datos debe ser común para ambas vistas de la aplicación, la persistencia se separa de la lógica de negocio.

El tercer paso es la división de la lógica de negocio, el arquitecto decide no separarla porque las reglas de negocio de la tienda no son lo suficientemente complejas para que la separación proporcione ventajas.

Una vez realizada la separación horizontal, se debe señalar la separación vertical en dos vistas (1) sitio Web y (2) administración, para los subsistemas *Presentación*, *ControlUsuario* y *LogicaNegocio*. Esto se debe principalmente a la falta de cohesión entre ambas vistas, ya que cada una de ellas ha de soportar requisitos no funcionales diferentes.

Por último, se establecen los EnlaceSubsistemas entre cada subsistema y el de la capa inferior. Además, para poder satisfacer el requisito 3, (indica que ambos interfaces muestren la misma información simultáneamente) se establece que la interfaz del administrador almacene en memoria las modificaciones sobre los datos en el subsistema *ControlUsuario* y el sitio Web los consulte. Así, se define un *EnlaceSubsistema* entre ambos. Puede verse el modelo resultante en la Figura. 20.

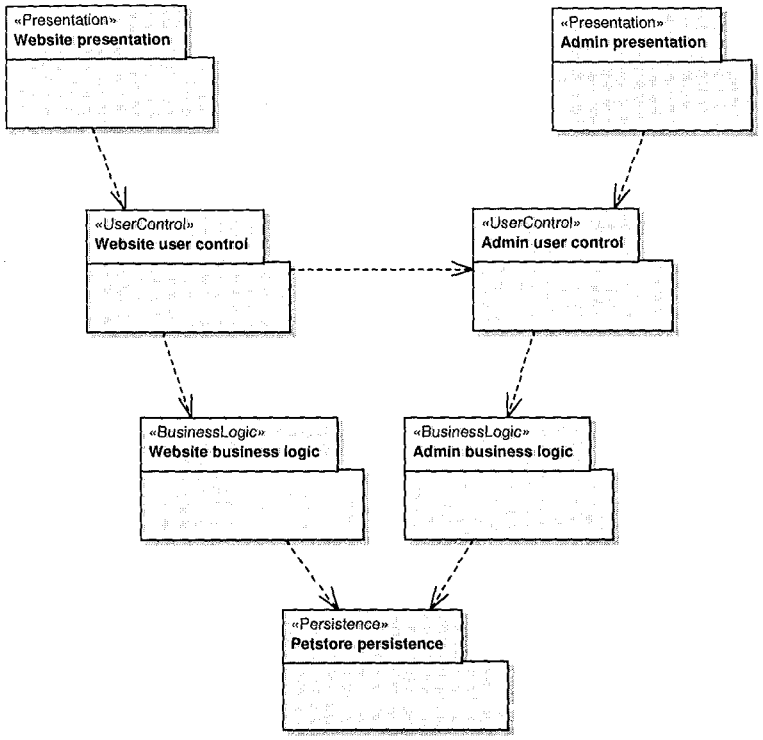

Figura. 20. Modelo de Subsistemas de Petstore.

5.8 Conclusiones del Capítulo

En el modelo de subsistemas se inicia la definición de la arquitectura de la aplicación Web, realizando una adaptación de metodologías como Renzel & Keller [100] y Buchmann et al. [17] al caso concreto de las aplicaciones Web. WebSA propone no solamente un modelo, sino también un proceso de descomposición, por el que se aplican progresivamente un conjunto de patrones de distribución dirigidos por fuerzas vinculadas a los requisitos no funcionales. Los patrones de distribución indicarán cual es la separación idónea en diferentes subsistemas de la aplicación Web.

MS se formaliza mediante la definición del metamodelo de subsistemas en MOF, esto no solamente implica la representación de cada uno de los elementos y sus relaciones. Sino establecer las convenientes restricciones en OCL, entre cada uno de los elementos, para indicar cuáles son las relaciones válidas entre sus elementos.

Además, se pretende la estandarización de MS, mediante la definición del perfil UML de subsistemas. Este perfil define todos los elementos de MS como extensiones de diferentes elementos definidos en el metamodelo UML 2.0. Esto permite representar MS en cualquier herramienta que tenga soporte de UML 2.0.

Volver al índice/Tornar a l'índex

Por último, es importante remarcar el papel que el MS tiene dentro del proceso de desarrollo de WebSA. El MS tiene como tarea principal la restricción de las posibles configuraciones arquitectónicas en la fase de diseño, siendo los subsistemas quienes establecen la distribución en módulos y los enlaces entre subsistemas quienes indican los flujos de comunicación entre los diferentes módulos. Este mapeo se establece mediante la transformación T1 entre análisis y diseño.

En el siguiente capítulo se describe el segundo de los estilos arquitectónicos definido por WebSA en la fase de análisis, el modelo de configuración.

CAPÍTULO 6

Modelo de Configuración de Componentes Web

"El objetivo de la abstracción es no perderse en vaguedades y crear un nuevo nivel semántico en el que se pueda ser absolutamente preciso".

[Edsger Dijkstra 1930-2002]

6.1 Motivación

Para completar la representación de la arquitectura Web en la fase de análisis, WebSA proporciona un estilo arquitectónico adicional llamado modelo de configuración de componentes que completa al modelo de subsistemas. Este estilo recoge un nuevo conjunto de requisitos no funcionales que no han sido recogidos por el Modelo de Subsistemas. Para ello, el modelo de configuración reside en un nivel de abstracción más preciso situado al nivel de componente, proporcionando así un pronóstico claro de nuestra arquitectura, reduciendo los riesgos y estableciendo una estimación temprana de la estructura definitiva de la aplicación Web.

Actualmente, existen aproximaciones genéricas que permiten representar la arquitectura en fases tempranas como ABD [6] pero, debido a su genericidad, no son adecuados para expresar con la suficiente precisión las características propias de las aplicaciones Web. Por otro lado, existen diferentes aproximaciones centradas en la representación de la arquitectura de las aplicaciones Web, que por diferentes motivos no son optimas para nuestro objetivo. Algunas aproximaciones están pensadas solamente para documentar como REST [31], o para otro tipo de procesos como Hassan & Holt [42], o porque se centran en el nivel de diseño y son poco escalables como WAM [24]. No se dispone de una notación adecuada para representar la arquitectura de una aplicación Web en una fase temprana, utilizando aspectos que sean propios de este tipo de aplicaciones.

WebSA (Meliá et al. [77]) propone la definición de un nuevo estilo arquitectónico centrado en el dominio de las aplicaciones Web que permite expresar de forma

sencilla y clara la configuración de nuestra aplicación Web. Este estilo arquitectónico es fruto del estudio de la familia de aplicaciones Web. En ella, se ha pretendido identificar cuáles son los diferentes tipos de componentes que configuran una aplicación Web. Una vez identificados, cada uno de estos tipos componentes tiene asociado una serie de características que los hace distintos al resto de componentes de la familia de aplicaciones Web. Cada una de estas características está vinculada con aspectos arquitectónicos o con aspectos funcionales, que permiten establecer en que posiciones dentro de los componentes se situarán los diferentes aspectos funcionales.

En consecuencia, se completa el análisis arquitectónico propuesto por el proceso de desarrollo de WebSA con la vista de configuración. La vista de configuración permite representar modelos dirigidos patrones de arquitectura y cuya unidad arquitectónica son los componentes Web. Esta vista está constituida por el modelo de configuración (MC). El MC representa la arquitectura de la aplicación Web por medio de la configuración de los diferentes tipos de componentes que intervienen y sus relaciones. Estos componentes tendrán sus características propias como atributos, operaciones y puertos. Además, sus relaciones con otros componentes se establecerán mediante conectores o interfaces.

A la hora de formalizar la vista de configuración, se ha definido un metamodelo MOF para representar el conjunto de componentes y características que constituyen a los diferentes elementos. Además, este metamodelo sirve de origen de diferentes mapeos que establecen la transformación T1, desde la fase de análisis al diseño. Como se establece en el capítulo 4, en el proceso de WebSA, el modelo de configuración se modela de forma paralela a la vista funcional, y esto permite que pueda ser especificado de forma independiente, y por lo tanto, pueda ser intercambiado y reutilizado para diferentes aplicaciones Web.

Por otro lado, el modelo de configuración ha sido definido siguiendo la premisa de estandarización propuesta por MDA. Su definición se basa en la realización de un perfil sobre un modelo estándar de arquitectura como el modelo de estructura compuesta definido por UML 2.0. Así, el perfil del MC permite que los modelos de arquitectura puedan ser definidos en cualquier herramienta con soporte UML 2.0.

El capítulo comienza situando las bases semánticas que fundamentan el estilo arquitectónico. Para ello, se define la unidad arquitectónica que describe cual es la semántica y el contenido que captura el concepto de componente Web utilizado por el estilo del modelo de configuración.

6.2 Definición de componente Web

El componente Web es el elemento central del estilo arquitectónico definido para la vista de configuración de WebSA. Por ello, es fundamental fijar su semántica y el contenido que encapsula el componente Web. Para permitir un adecuado modelado en el que se pueda disponer de una visión global y clara de la arquitectura en la fase de análisis, esta unidad arquitectónica es una abstracción que representa una entidad potencialmente grande.

La definición del concepto es la siguiente:

"Un componente Web representa una abstracción de uno o más componentes software los cuáles comparten la misma funcionalidad o tarea, la cual ha sido

identificada como común, no trivial y necesaria, dentro de la arquitectura de la familia de las aplicaciones Web".

De esta manera, para definir este estilo arquitectónico, se necesita realizar un esfuerzo de abstracción, que consiste en estudiar los diferentes tipos de componentes dentro de la familia de las aplicaciones Web, y extraer así cuáles son los elementos que la constituyen. Se han de considerar elementos cuidándose de aquellos que no sean necesarios para representar la arquitectura de la aplicación. Así por ejemplo, determinados componentes software como por ejemplo, un servidor Web, no se representa dentro de la arquitectura, puesto que no forma parte de los componentes software que WebSA desarrolla, sino que es un software preestablecido y proporcionado por la plataforma Web.

El componente Web es un elemento conceptual, similar al propuesto por otras aproximaciones como ABD [6], y centrado en un dominio específico. Esto le otorga al componente la posibilidad de ser modelado en una fase temprana y hacerlo corresponder con los requisitos planteados. Como se muestra en la sección 6.5, los componentes Web o la combinación de ellos, permiten la representación de los patrones de arquitectura generales o particulares de las aplicaciones Web.

Un ejemplo de componente Web, es el *ControladorWeb*, se trata de un componente encargado de recibir las peticiones realizadas por el usuario, reenviarlas a la lógica de negocio y establecer la reacción de la interfaz. En su definición queda claro cuáles son las tareas realizadas dentro de la arquitectura, no por ello está exento de variabilidades. Es decir, tiene asociado un conjunto de propiedades y operaciones, que pueden variar según la arquitectura de la aplicación. Por lo tanto, es necesario determinar claramente cuáles son sus tareas.

En el MC, por cada componente Web, se define un conjunto de características propias como son: (1) propiedades, (2) operaciones, (3) subcomponentes (partes) y (4) puertos. Además de expresar la comunicación entre los diferentes componentes mediante conectores e interfaces.

Cada una de las características del componente, está vinculada, según la distinción establecida por Selic [107], con aspectos propios del control o de la funcionalidad de la aplicación. Las características de control, son aquellas que permiten llevar al sistema a su estado operacional, y lo mantienen en funcionamiento frente a posibles trastornos planeados o no. Estos aspectos de control se definen de forma independiente a los funcionales. Por otro lado, los aspectos funcionales son aquellos que recogen los aspectos dependientes de la funcionalidad del problema, es decir, en caso de las aplicaciones Web, será el dominio, su navegación y el proceso. Al estar situado el MC en la vista de arquitectura, los aspectos funcionales son tratados de forma genérica, es decir, se indicará donde estarán ubicados los atributos de dominio, un determinado servicio navegacional, etc. dentro de los diferentes componentes, sin hacer referencia a ninguna instancia concreta del dominio del problema. Se consigue así, mantener la independencia de la vista de arquitectura con la vista funcional.

6.3 Topología de componentes Web

Basados en la definición de Hayes y Roth [44], *una Arquitectura del Software específica de un dominio, está constituida por el ensamblado de un conjunto de*

componentes software, cada uno de los cuáles está generalizado para el uso efectivo del usuario y definido dentro de una topología efectiva para construir aplicaciones. En nuestra aproximación se definen de arquitecturas centradas en el dominio de las aplicaciones Web, para ello se ha establecido una topología de componentes, cada uno de los cuáles desempeña una funcionalidad o tarea dentro de una aplicación Web. Para conseguir que sea una topología efectiva, se ha procurado reducir al máximo, el número de tipos de componentes que se podrán encontrar en una aplicación Web, para que al usuario le sea más sencillo poder identificar los diferentes elementos de modelado.

Para poder hacer dicha clasificación, se ha realizado un proceso de recolección a partir de definiciones basadas en elementos de arquitectura utilizados en otras aproximaciones como WAM [24], EDOC [96], REST [31], y a partir del propio estudio de variabilidades de la arquitectura dentro del dominio de las aplicaciones Web.

En el Apéndice II se describen los 19 tipos diferentes de componentes identificados. En algunas ocasiones, estos tipos se encuentran asociados a un determinado tipo de subsistema o capa lógica, mientras que en otras ocasiones los componentes pueden vivir en cualquier parte del sistema. Esta información, será utilizada en la fase de transformaciones T1 para ubicar los componentes según su capa en distintos módulos.

Cada uno de los tipos de componentes tendrá asociado un conjunto de propiedades y operaciones. Dichas propiedades pueden ser provenientes del control de la aplicación, en cuyo caso, están definidas por el arquitecto y por las propias características del tipo de componente.

Cada uno de los tipos de componentes es formalizado por medio de una clase dentro de una jerarquía de tipos de componentes dentro del metamodelo configuración.

Para presentar los distintos elementos, la topología distinguirá en tipos de componentes comunes y específicos. Los tipos de componentes específicos se refieren a su vinculación con un subsistema Web en concreto.

6.4 Modelo de Configuración (MC)

Una vez definido concepto de componente Web y los diferentes tipos que pueden encontrarse dentro la vista arquitectónica de configuración, se está en disposición de definir el modelo de configuración.

El MC proporciona una vista estructurada de la aplicación Web mediante un conjunto de componentes Web y sus conectores, que permiten representar la arquitectura de la aplicación Web, de una forma genérica, para cualquier tipo de dominio del problema. Permitiendo así, expresar la arquitectura en un nivel de abstracción adecuado para la representación de patrones de arquitectura y establecer un vínculo con los requisitos no funcionales.

Como se define en la sección 6.2, cada uno de estos componentes Web encapsulan la funcionalidad de uno o más componentes del mismo tipo y comportamiento común. Esto permite que pueda representarse la arquitectura en la fase de análisis de una forma escalable y sin entrar en detalles de diseño.

Para formalizar el MC, WebSA ha definido el metamodelo de configuración, el cual permite representar los conceptos más importantes del modelo, como son su unidad arquitectónica el componente *CompWebCM*, sus propiedades como *PuertoWeb*, *OperacionWeb* y las relaciones *ConectorWeb* e *InterfazWeb* y el elemento contenedor *PatronWeb*. Por otro lado, se ha establecido una jerarquía de tipos a partir de *CompWebCM*, definiendo para cada uno de ellos una metaclase (p.e. Entidad, ControladorWeb, etc.) que hereda las características de *CompWebCM* y define las propias de su tipo (definidas en el punto 6.3).

Para la representación del MC, se define un perfil sobre el modelo de estructura compuesta de UML 2.0. El conjunto de componentes y sus propiedades se representarán siguiendo la topología de componentes Web definida en la sección 6.3, representando cada uno de los tipos de componentes como estereotipos de la metaclase *Class* definida para el modelo de estructura compuesta.

Es importante destacar que en la arquitectura de la aplicación, hay que ser capaz de representar aspectos como el anidamiento de componentes y la representación de diferentes patrones que se repiten en distintas arquitecturas. Gracias a que MC se basa en los elementos del modelo de estructura compuesta, utiliza mecanismos de modelado que permiten representar de forma natural aspectos como la composición de un componente por otros componentes, o la definición y la aplicación de un patrón de arquitectura mediante la extensión del concepto de colaboración.

A continuación se definen cuáles son los elementos más importantes en el MC y su notación. Seguidamente se formalizará el modelo MC mediante el metamodelo de configuración estática. Se establecerá el método a seguir para definir el modelo. Y se definirá su estandarización mediante el perfil UML de configuración estática. Por último, para demostrar su versatilidad, se aplicará MC sobre dos casos de estudio.

6.4.1 Elementos del Modelo de Configuración

Esta sección define los elementos que se utilizan en el Modelo de Composición. El elemento principal de este modelo es *CompWebCM* que junto a sus propiedades, relaciones y sus especializaciones completan los elementos utilizados.

6.4.1.1 CompWebCM

Un *CompWebCM* representa una abstracción de uno o más componentes software que comparten la misma funcionalidad o tarea en el contexto de la familia de aplicaciones Web. Por ejemplo, una *PaginaEstática* es un *CompWebCM* que contiene los datos de presentación y la interacción de código con el usuario. Sin embargo, *PaginaEstática* no necesariamente se convertirá en la implementación en una única pagina física, sino que representa una tarea genérica que se realiza dentro de una aplicación Web, como es el hecho de mostrar cierta información al usuario. Las propiedades más importantes de un *CompWebCM* son las definidas por las metaclases *OperacionWeb*, *PuertoWeb*, *InterfazWeb* y *ParteWeb*.

El *CompWebCM* es también la metaclase raíz de una jerarquía de metaclases que representan los diferentes tipos de componentes identificados en la topología de configuración (ver sección 6.3). Por ejemplo, se define la metaclase *Entidad* que hereda de *CompWebCM*, como un componente que representa una clase y sus

propiedades del modelo de dominio y que además hereda las propiedades definidas por la clase raíz *CompWebCM*.

La notación utilizada por el *CompWebCM*, es la siguiente:

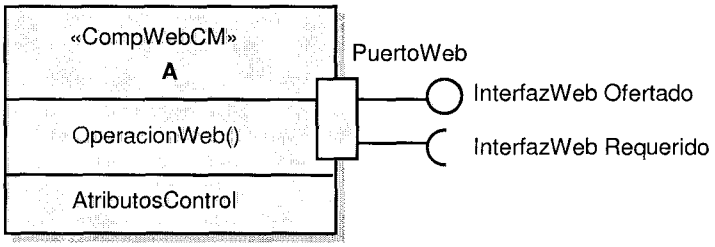

Figura. 21. Notación del CompWebCM y sus Propiedades.

Como se puede apreciar en la Figura. 21, el *CompWebCM* mantiene la notación de la clase estructura de UML. Contiene 3 rectángulos, en el primer rectángulo se encuentra el estereotipo, en este caso CompWebCM y el nombre del componente A. El estereotipo utilizado es CompWebCM indica que se trata de un componente Web genérico del MC. Su definición junto con la del resto de estereotipos se especifica en la sección 6.7. En el segundo rectángulo, se encuentra el comportamiento del componente, especificando por un conjunto de elementos *OperacionWeb*. Un *OperacionWeb* (ver sección 6.4.1.3) representa a una propiedad de comportamiento del propio componente. Por último, se especifican los atributos que permiten especificar las diferentes propiedades de las metaclases. Además, en su contorno, se definen los puntos de interconexión con otros componentes contiene elementos *PuertoWeb*, los cuáles pueden contener o no un conjunto de *InterfazWeb*.

Todos los subtipos de *CompWebCM* definidos en la sección 6.3 (p.e. *PaginaServidora, ControladorWeb, PaginaEstatica*, etc.) mantienen la misma notación.

6.4.1.2 AtributosControl

Representan a las propiedades de control del *CompWebCM* en el modelo MC, que han sido definidas como atributos en el metamodelo de MC y establecidas como atributos del estereotipo en el perfil de MC. Estas propiedades son dependientes de cada tipo de componente, y pueden hacer referencia a aspectos de control independientes de la funcionalidad. Por ejemplo, el componente Almacén tiene atributos como acceso, tipoOrganización y tipoDatos. Pueden verse más ejemplos en la sección 6.3.

6.4.1.3 OperacionWeb

Representa a las propiedades de comportamiento del *CompWebCM*, que se corresponden en ocasiones con operaciones de control u operaciones provenientes de la parte funcional. Estas propiedades no son dependientes del tipo de componente, y se pueden establecer las operaciones que el arquitecto considere conveniente. En la sección 6.3 se proponen determinados *OperacionWeb* para tipos de atributos de forma

orientativa (p.e. a *ControladorWeb* se le añaden get y post como posibles *OperacionWeb*).

6.4.1.4 PuertoWeb

PuertoWeb es el punto de interacción entre un *CompWebCM* y su entorno. Se encarga de desacoplar los aspectos internos del componente de la interacción con otros componentes, haciendo así al componente reutilizable para cualquier entorno que esté conforme con las restricciones de interacción impuestas por los *PuertosWeb*. De este modo, un *CompWebCM* solamente puede comunicarse con el exterior a través de las instancias de PuertoWeb. Su notación se representa por medio de un pequeño cuadrado o rectángulo (cuando tiene más de una *InterfazWeb*) en el borde del *CompWebCM* (Ver Figura. 21).

6.4.1.5 InterfazWeb

Una *InterfazWeb* representa la funcionalidad que el componente ofrece o requiere al resto del sistema para poder realizar su tarea. Cada *InterfazWeb* está asociado a un *PuertoWeb* que especifica la naturaleza de las interacciones que pueden ocurrir sobre él. Por un lado, los interfaces requeridos de un *PuertoWeb* caracterizan las peticiones que pueden hacer el *CompWebCM* a su entorno. Por otro lado, los interfaces ofertados por un *PuertoWeb* caracterizan las peticiones que el entorno hace al *CompWebCM*.

La *InterfazWeb* ofertada se representa con la notación lollipop, que consta de una línea que tiene como extremo un círculo completo, y la *InterfazWeb* requerida es una línea que tiene como extremo un semicírculo. Su notación puede apreciarse en la Figura. 21.

6.4.1.6 ConectorWeb

El *ConectorWeb* especifica un enlace que permite la comunicación del sistema entre dos o más *CompWebCM* y/o *ParteWeb*. Esta comunicación se establece a través de los PuertosWeb. Sin embargo, en el caso de una *ParteWeb* esta relación puede afectar o al *PuertoWeb* o a toda la *ParteWeb*. Cada *ConectorWeb* tiene asociado dos elementos *FinalConectorWeb* que indican la cardinalidad con la que los componentes se relacionan entre sí.

A la hora de especificarlo, se cuenta con dos notaciones:

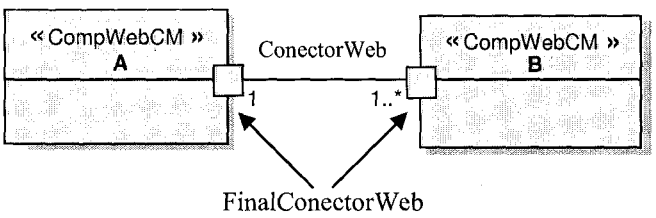

Figura. 22. Notación del ConectorWeb Directa.

Esta notación especifica un ConectorWeb une directamente a los puertos de los componentes. Como se puede apreciar en la Figura. 22, se asocia la cardinalidad, en

este caso, indicando que un componente B está relacionado con un componente A, y un componente A con uno o muchos componentes B.

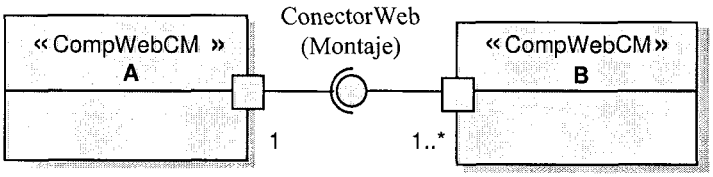

Figura. 23. Notación del ConectorWeb mediante Interfaces.

La notación que especifica la Figura. 23, indica que se establece el ConectorWeb mediante una interfaz ofertada por el componente B, que coincide con el requerido por el componente A. El ConectorWeb, se especifica con el nombre de Montaje (en inglés Assembly).

6.4.1.7 FinalConectorWeb

El elemento *FinalConectorWeb* representa el extremo del *ConectorWeb* que se une al *PuertoWeb* o a una *ParteWeb*. El *FinalConectorWeb* tiene dos propiedades: (1) cardInferior (cardinalidad inferior) que especifica el límite inferior de elementos que pueden estar unidos al conector, (2) cardSuperior (cardinalidad superior) que especifica el límite superior de elementos que pueden estar unidos al conector. Por defecto, si la cardinalidad no se especifica tendrán los valores: cardInterior = 1 y cardSuperior = 1.

6.4.1.8 ParteWeb

La *ParteWeb* representa al conjunto de instancias de un *CompWebCM* que pueden ser propiedad por composición de otro *CompWebCM* o pertenecer como actores de *PatronWeb*. La comunicación de una *ParteWeb*, puede establecerse mediante un *PuertoWeb* o directamente utilizando un *FinalConectorWeb*.

La parteWeb tiene un nombre precedido de ":" indicando que se trata de cualquier instancia del componente al que representa. Además, la *ParteWeb* tiene un atributo llamado multiplicidad, que usa la notación [x{...y}]. Donde (x) especifica en la cantidad de instancias de la *ParteWeb* que contiene el *CompWebCM* cuando este se crea. E (y) especifica la cantidad máxima de instancias de la *ParteWeb*, en cualquier momento de la vida del *CompWebCM*.

Su notación es la siguiente:

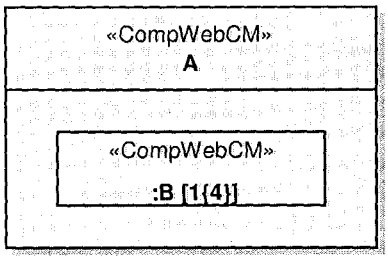

Figura. 24. Representación de la ParteWeb B en el CompWebCM A

Como se aprecia en la Figura. 24, la *ParteWeb* B se especifica como una caja rectangular, dentro del *CompWebCM* A. Dicha *ParteWeb* hace referencia a un tipo de componente Web, por ello la notación debe mostrar el estereotipo de componente que instanciado. En el ejemplo se especifica que existe una ParteWeb del componente B, en el momento de la creación de A, y hay un máximo de 4 instancias de B en la vida del componente A. Por ello, esta forma de notación restringe más la vida del componente parte al componente contenedor, además de limitar las conexiones de la parte a las que posee el contenedor.

6.4.1.9 PatronWeb

El elemento *PatronWeb* representa a un patrón de arquitectura Web que se especifica mediante un elemento compuesto, constituido por un conjunto de *ConectorWeb* y *ParteWeb* los cuáles realizan una determinada tarea, dentro de la función desarrollada por el patrón.

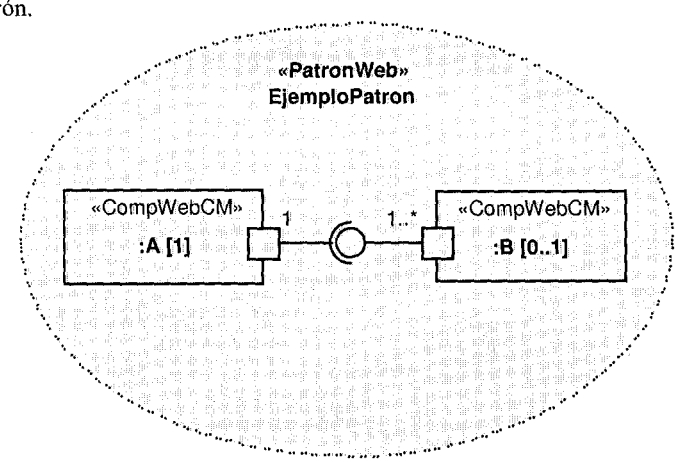

Figura. 25. Notación de la Definición de un PatronWeb

Por lo tanto, el *PatronWeb* permite definir un conjunto de elementos que pueden ser reutilizados en diferentes modelos de configuración. Por ejemplo, se pueden definir elementos *PatronWeb*, que contengan conocidos patrones como MVC definido por [17], se puede representar el patrón *Façade* definido por [36], y aplicado para los

componentes de servidor distribuidos. En la sección 6.7 podrán verse diferentes representaciones de patrones mediante elementos *PatronWeb*.

La notación para la definición de un *PatrónWeb* es la siguiente:
En la Figura. 25 se aprecia un *PatronWeb* que contiene dos *PartesWeb* que están relacionadas por medio de un *ConectorWeb* montaje. Este conector relaciona una instancia de A con una o muchas de B, y una instancia de B con una de A. Además, especifica la multiplicidad de las instancias dentro del patrón, indicando que tiene una instancia de A, y cero o una de B. Una vez definido este *PatronWeb* puede ser aplicarlo dentro del modelo, utilizando para ello la siguiente notación:

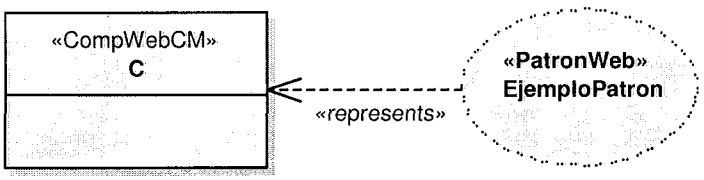

Figura. 26. Notación de la Aplicación del PatronWeb.

En la Figura. 26 se muestra como el *PatronWeb* que ha sido denominado EjemploPatron, y que se ha definido en la Figura. 25, es aplicado mediante la relación de dependencia denominada *represents* sobre el *CompWebCM* C. De esta manera, se indica que en el lugar del MC donde aparece CompWebCM C, realmente se está instanciando el *PatronWeb* EjemploPatron. En este caso, el PatronWeb siempre es aplicado sobre un componente que está estereotipado con el tipo genérico *CompWebCM*, ya que en este caso su contenido depende del patrón y no del tipo de componente.

Una vez definidos los elementos que contiene MC, se procede a formalizarlo en el metamodelo de configuración.

6.5 Metamodelo de Configuración

Para la definición del metamodelo de MOF de MC definido por WebSA, se ha considerado que los conceptos definidos para los diferentes modelos de componentes, no existen de forma aislada y pueden existir conceptos o metaclases utilizadas por más de un modelo. Así, a la hora de establecer los metamodelos que se refieren a la vista de configuración, se ha procurado reducir el número de metaclases, reutilizando aquellos conceptos que fueran comunes para los diferentes modelos.

En este sentido, se define un paquete llamado *modelos componentes* que encapsula tanto al modelo de configuración, como al modelo de integración. Esto se debe a que ambos modelos están basados en la representación de una misma unidad arquitectónica como es el ComponenteWeb, y además comparten un núcleo común, a pesar de que se definan en niveles de abstracción diferentes.

Así el paquete *modelos de componentes* tiene los siguientes elementos:

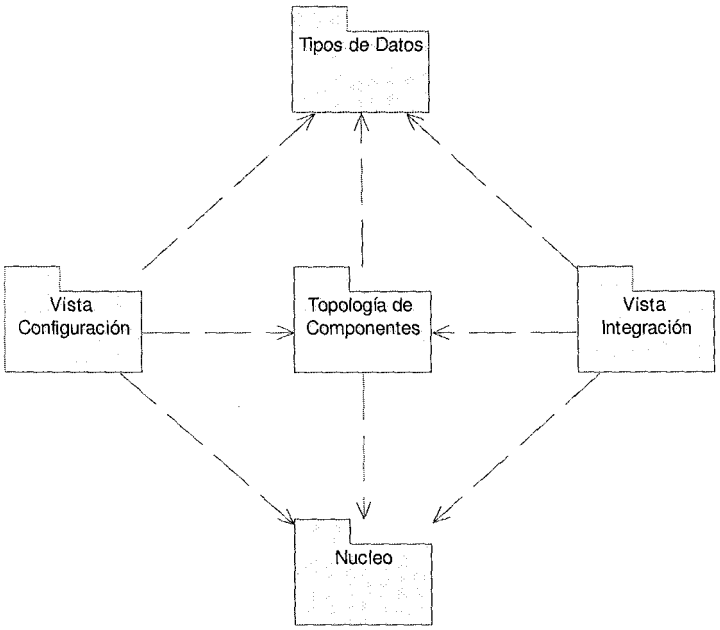

Figura. 27. Paquete Modelos de Componentes

Como muestra la Figura. 27, dentro del paquete *Modelo de Componentes* se define un paquete llamado *Núcleo* que se considera común para ambas vistas de configuración e integración. El *Núcleo* contiene los elementos básicos de los modelos de componentes como *ComponenteWeb, ConectorWeb, InterfazWeb*, etc. que son utilizados en ambos modelos. Asociado al núcleo por una relación de dependencia, se definen dos paquetes llamados *Vista Configuración* y *Vista Integración,* que contienen los elementos que son utilizados específicamente en los modelos de configuración e integración respectivamente. Por otro lado, se define un paquete llamado *Topología de Componentes*, que contiene la jerarquía de tipos de componente Web establecida para ambas vistas. Y por último, se define un paquete llamado *Tipos de Datos* donde se definen los diferentes tipos de datos que son utilizados en las propiedades de los componentes.

Una vez definidos los paquetes, seguidamente se muestra el contenido de cada uno de ellos. La descripción comienza por los paquetes que son compartidos por ambas vistas, como son los paquetes *Núcleo, Topología de Componentes* y *Tipos de Datos*. Finalmente, se profundiza en la *Vista Configuración* donde se definen los conceptos utilizados concretamente por el modelo MC.

6.5.1 Paquete Núcleo

En este paquete se definen los elementos que son comunes para los modelos de componentes, es decir, para MC y MI. En la Figura. 28, la parte del metamodelo que

contiene el paquete núcleo. En el núcleo el elemento o metaclase es el *ComponenteWeb*, que como se ha indicando anteriormente es la unidad arquitectónica utilizada en MC y MI. El *ComponenteWeb* contiene un conjunto de propiedades (como se ha definido en la sección 6.4.5) que se representan a su vez mediante metaclases. Estas metaclases están relacionadas con *ComponenteWeb* utilizando una relación de composición. Por ejemplo, un *ComponenteWeb* contiene un conjunto de instancias *OperacionWeb* mediante la relación de composición *CTieneOperacionesWeb*, además un *OperacionWeb* pertenece únicamente a un *ComponenteWeb*. Un *OperacionWeb* contiene a su vez a un conjunto de instancias *ParametroWeb*. Cada *ParametroWeb* indica el nombre, su tipo y si es de entrada, salida o ambas.

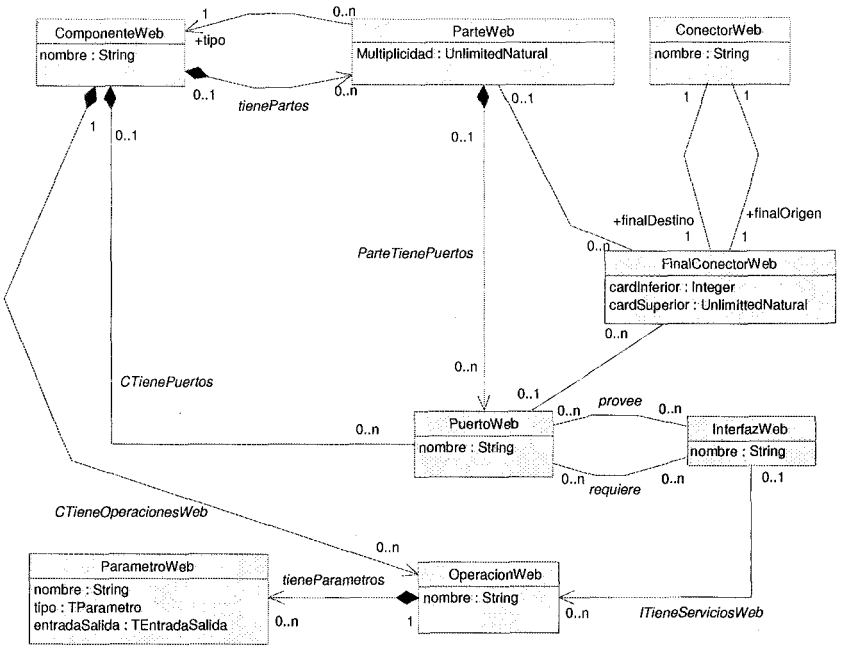

Figura. 28. Paquete Núcleo del Metamodelo de WebSA

Además, el ComponenteWeb también tiene una relación de composición con sus otras dos propiedades *ParteWeb* y de *PuertoWeb*. La *ParteWeb* tiene el atributo multiplicidad para indicar cual es el número de instancias de la parte que puede contener el *ComponenteWeb* o el *PatronWeb*. A su vez, la parte está asociada mediante una asociación a un *ComponenteWeb*, que representa el tipo de componente que está instanciando la ParteWeb.

Siguiendo con la descripción del núcleo, se establece una relación entre los conceptos *PuertoWeb* e *InterfazWeb*, mediante dos relaciones de asociación. Por un lado, se establece la relación *provee*, que indica que el *PuertoWeb* provee a la interfaz

al resto del sistema. Por otro lado, la relación *requiere* indica que el *PuertoWeb* requiere la *InterfazWeb* del sistema.

Por otro lado, el núcleo también representa los elementos que sirven de enlace entre los *ComponentesWeb*, es decir, los conectores. Para ello, se define la metaclase *ConectorWeb*, la cual va a contener dos elementos *FinalConectorWeb*, que representan a cada uno de los extremos del conector. Cada elemento *FinalConectorWeb*, contiene los atributos *cardInterior* (cardinalidad mínima) y *cardSuperior* (cardinalidad máxima) y esta relacionado con un *PuertoWeb* o con una *ParteWeb*. Así se fija la restricción mediante la cual un *ComponenteWeb* solo se relaciona con el exterior a través de un puerto.

Una vez definidos los elementos del metamodelo del paquete Núcleo, se describe la topología de componentes que es utilizada por los modelos MC y MI.

6.5.2 Paquete Topología de Componentes

Como se ha especificado previamente, WebSA propone un estilo arquitectónico en el que se define una topología de componentes donde cada tipo de componente desempeña una funcionalidad o tarea no trivial dentro de una aplicación Web. Para poder formalizar dicha topología, se ha introducido en el metamodelo de componentes. Así, cada uno de los tipos de componentes definidos en la sección 6.3, se representa mediante una metaclase en el metamodelo. Además, la relación de herencia es utilizada para expresar tipos y subtipos, y establecer un vínculo los tipos de componente y las diferentes capas definidas en el modelo de subsistema.

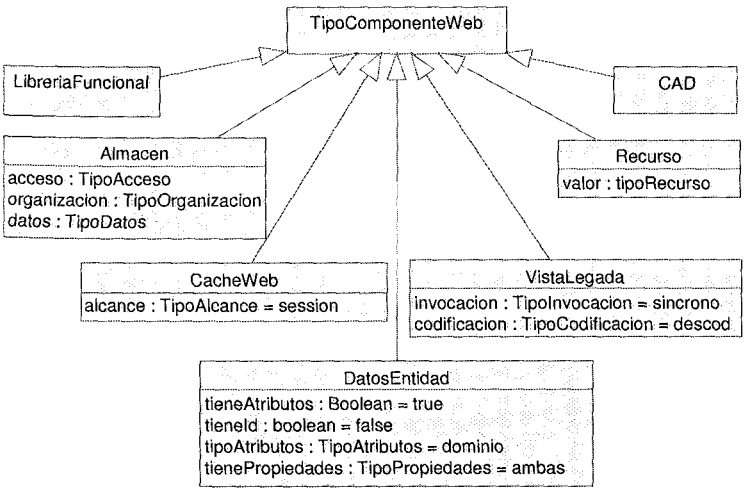

Figura. 29. Paquete de Componentes Comunes

La topología de componentes está dividida en 3 subpaquetes, cada uno de los cuáles representa la categoría de los tipos de componentes que contiene. Los paquetes que la constituyen son los siguientes: (1) Componentes Comunes, (2) Componentes de Interfaz de Usuario y (3) Componentes de Servidor.

La Figura. 29 muestra la jerarquía de componentes que se encuentra en el paquete Componentes Comunes. En este paquete se especifican los tipos de componentes que pueden asociarse a cualquier capa lógica. La primera metaclase a destacar es la llamada *TipoComponenteWeb* que representa a la clase raíz de la jerarquía de componentes, es un tipo genérico de componente Web que sirve para especificar aquellos componentes nuevos que son demasiado concretos para ser asociados a un tipo de la jerarquía y para los casos en los que un componente Web representa un patrones de arquitectura definido en MC. En el metamodelo de WebSA *TipoComponenteWeb* juega un papel fundamental, ya que no solamente es la raíz de la jerarquía de componentes, sino representa el puente con las metaclases que definen a los componentes en los modelos MC y MI, representando su tipo.
El resto de tipos de componentes comunes heredan directamente del tipo genérico *TipoComponenteWeb*. Los diferentes tipos de componentes, contienen en la mayoría de los casos un conjunto de atributos que representan a cada una de las propiedades que se han especificado en la topología de la sección 6.3. Por ejemplo, una *VistaLlegada* tiene dos atributos, *tipoInvocación* que puede ser síncrono o asíncrono, y que por defecto es síncrono, y *codificación* cuyo tipo TipoCodificación se define en el paquete *Tipos WebSA* y que por defecto es *descod* (descodificación). El resto de componentes contiene como atributos los mismos componentes especificados en la sección 6.3. Sin embargo, repasando otro tipo de componente como *CAD*, se indicaba que *CAD* tenía un conjunto de operaciones asociadas como *insertar, modificar, borrar,* etc. Estas operaciones son especificadas a través de la relación que *ComponenteWeb* tiene con un conjunto de instancias *OperacionWeb*. Queda en manos del arquitecto la decisión de asociarle o no a CAD las operaciones Web.

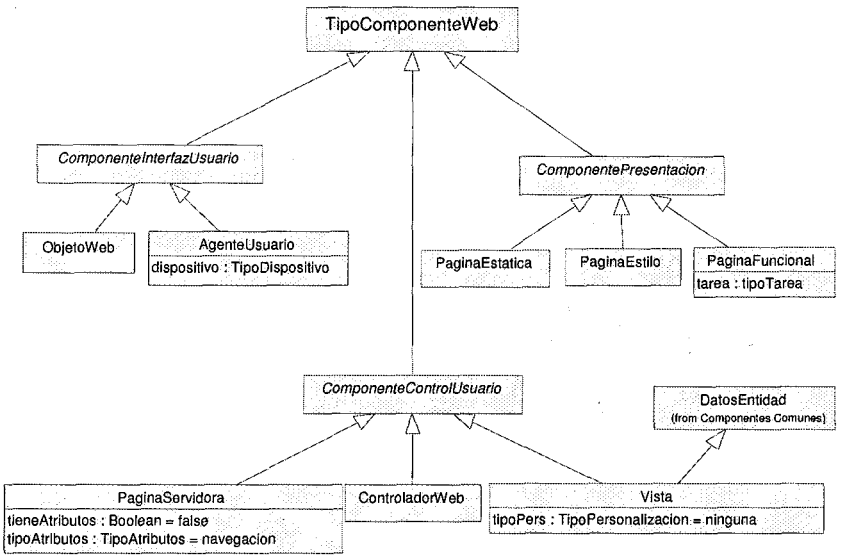

Figura. 30. Paquete Componentes de Interfaz de Usuario

La Figura. 30 muestra el paquete de componentes de interfaz de usuario que contiene los tipos de la jerarquía de componentes Web que son especificados para la arquitectura de la interfaz de una aplicación Web. Como se aprecia, se establece una jerarquía que está acorde con los diferentes tipos de subsistemas o capas lógicas que se encuentran en la interfaz de usuario. Se dispone de tres tipos de subsistemas de interfaz de usuario, y a cada uno de ellos se le asocia un tipo de componente abstracto que vinculado a cada uno de ellos, *ComponenteInterfazUsuario, ComponentePresentación y ComponenteControlUsuario.* Cada uno de estos componentes hereda directamente de el tipo raíz *TipoComponenteWeb* y su tienen como misión representar a los diferentes subtipos de componentes que residen en este subsistema. Esto permite la definición de transformaciones T1, permitiendo unificar al conjunto de componentes de una determinada capa, p.e. los tipos de componentes que heredan de tipo *ComponentePresentación* se ubican en un módulo de tipo *Presentación.*

Finalmente, es importante reseñar la herencia múltiple definida para el tipo de componente Vista. Por un lado, Vista hereda de *ComponenteControlUsuario* asociándole a dicha capa lógica, y por otro lado, hereda de *DatosEntidad* todos sus atributos (*tieneAtributos*, *tipoAtributos*, etc.) a los cuáles incorpora el *tipoPers* donde la Vista introduce la personalización a nivel de interfaz. El resto de tipos de componentes, contienen en algunos casos como *AgenteUsuario*, *PaginaServidora* y *PaginaFuncional* un conjunto de atributos que representan a cada una de las propiedades que se han especificado en la topología de la sección 6.3.

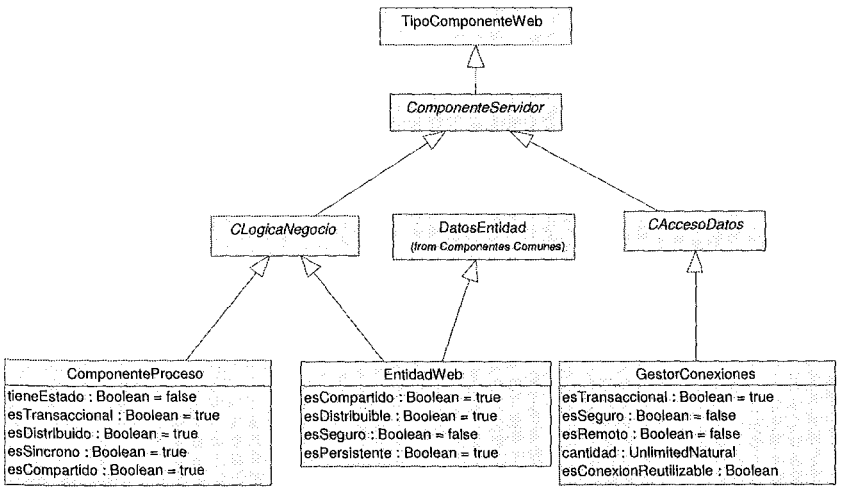

Figura. 31. Paquete Componentes de Servidor

Para finalizar la jerarquía de tipos de componentes, se describe en la Figura. 31 el paquete Componentes de Servidor, en el cual se ubican aquellos componentes que pueden vivir en cualquiera de los tipos de subsistemas que contienen funcionalidad con procesamiento en servidor.

La jerarquía de componentes de servidor al igual que ocurre en la interfaz de usuario, establece una descomposición en los diferentes tipos de subsistemas que se encuentran localizados en la parte servidora de la aplicación Web. Se define así, un conjunto de tipos de componentes abstractos que se corresponden con cada una de las capas lógicas de servidor *ComponenteServidor, CLogicaNegocio, CAccesoDatos.*

Estos tipos genéricos de componentes, permiten definir reglas de transformación que unifiquen todos los tipos de componentes que se encuentren asociados a una determinada capa. Los nuevos tipos de componentes definidos residen en las hojas de esta jerarquía heredando dicha tipología. Por ejemplo, un *ComponenteProceso*, es un a su vez un tipo de componente *CLogicaNegocio*, que a su vez es *ComponenteServidor*, que por último es un *ComponenteWeb.*

Es importante destacar la herencia múltiple definida para el tipo de componente EntidadWeb. Por un lado, Entidad hereda de *ComponenteCLogicaNegocio* asociándole a dicha capa lógica y contiene un conjunto de atributos propios como son (*esCompartido, esDistribuible, esSeguro y esPersistente)*. Por otro lado, es una especialización de *DatosEntidad*, heredando todos sus atributos (*tieneAtributos*, *tipoAtributos*, etc.). Los otros dos tipos de componentes, *ComponenteProceso* y *GestorConexiones*, contienen un conjunto de atributos que representan a cada una de las propiedades especificadas en la topología de la sección 6.3

6.5.3 Paquete Tipos de Datos

Este paquete contiene los diferentes tipos de datos asociados a los atributos definidos por las metaclases del metamodelo de componentes. Su papel en el metamodelo de WebSA es muy importante ya que los tipos de datos permiten fijar los posibles valores que tienen los atributos de los componentes Web, haciendo más sencillo conocer al modelador cuáles son sus posibles valores y evitando errores al introducir los valores. Por otro lado, en este paquete también se definen los tipos de las instancias de ParametroWeb que son introducidas en los diferentes OperacionWeb de los componentes en los modelos MC y MI.

En la Figura. 32 se muestran los diferentes tipos de datos definidos dentro de metaclases estereotipadas como enumeration, es decir, son tipos de datos cuyos valores son especificados mediante un conjunto de literales representados como atributos de la clase. P.e. el tipo *TEntradaSalida* es un tipo de dato cuyos posibles valores son los que tiene definidos como atributos *entrada, salida* y *ambos.*

Realizando una descripción descendente de la Figura. 32, en la parte superior se cuenta con el tipo genérico *TDParametro* que representa el tipo de dato que tiene un parámetro y que se especializa en cuatro subtipos a su vez: (1) *TDPrimitivo*: que contiene los tipos primitivos comunes para cualquier lenguaje de programación que puede tener un parámetro; (2) *TDDominio*: contiene dos tipos de valores genéricos que son obtenidos desde el modelo de dominio, *idClaseDominio* que representa el identificador de una clase de dominio y *claseDominio* representa al tipo definido por una clase del dominio; (3) *TDNavegacion*: contiene dos tipos de valores genéricos que son obtenidos desde el modelo de navegación *idClaseNavegacion* representa el tipo del identificador de una clase navegacional y *claseNavegacion* representa al tipo definido por una clase navegacional; y por último, (4) *TDProceso*: contiene dos tipos de valores genéricos que son obtenidos desde el modelo de proceso *idClaseProceso*

representa el tipo del identificador de una clase proceso y *claseProceso* representa al tipo definido por una clase proceso.

El resto de metaclases enumeración definidas en la parte inferior, corresponden a los tipos de los atributos que han sido definidos sobre los tipos de componentes que contiene el paquete Topología de componentes (ver sección 6.4.6.2).

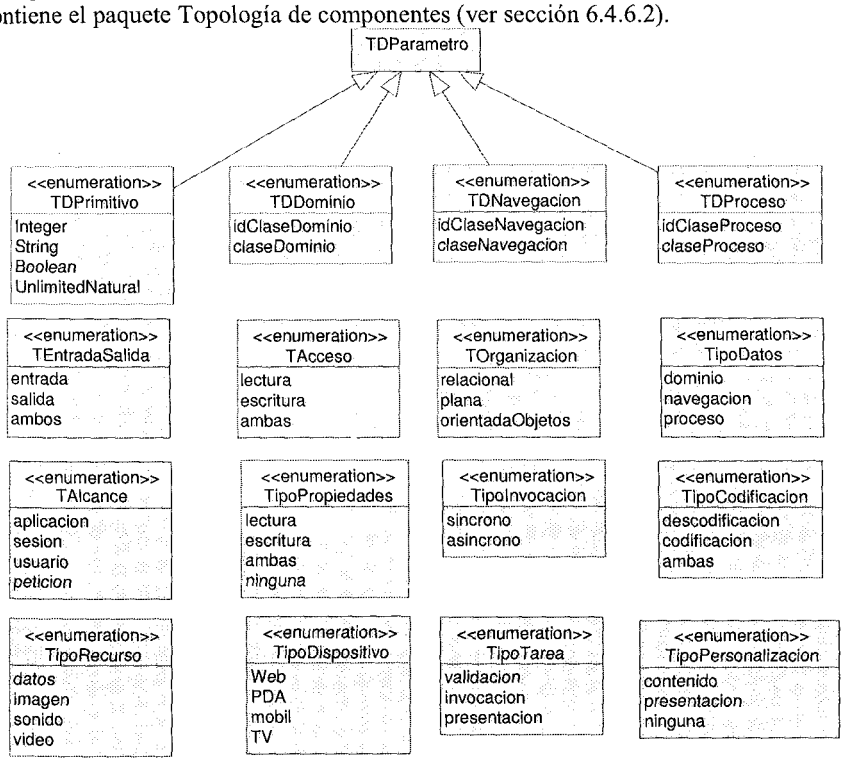

Figura. 32. Paquete Tipos de Datos

6.5.4 Paquete de Vista de Configuración

Este paquete contiene la parte del metamodelo de componentes, donde se encuentran aquellos elementos que son específicos de MC, es decir, que son utilizados únicamente en el modelo. Como muestra la Figura. 27 al comenzar la descripción del metamodelo de componentes, el paquete Vista de Configuración depende del paquete Núcleo, debido a que referencia a las metaclases comunes y las extiende introduciendo los nuevas metaclases que son específicas para el modelo MC.

La Figura. 33 muestra el metamodelo de la Vista de Configuración. El elemento principal es la metaclase MC, que representa al propio concepto del modelo de configuración, conteniendo a todos los elementos que participan en él. MC está relacionado por una relación de composición con las siguientes metaclases: (1) *CompWebCM*, representa la unidad arquitectónica utilizada en el modelo MC, consiste en una especialización de ComponenteWeb, del cual heredará todas las propiedades que se han definido en el paquete Núcleo (ver Figura. 28), al que además

incorpora características que son propias del modelo MC. (2) *PatronWeb*, permite representar patrones de arquitectura en este estilo arquitectónico. (3) *InterfazWeb* y (4) *ConectorWeb* que ya se han definido en el paquete núcleo. Considerando al componente como elemento central de este modelo, es *CompWebCM* en este modelo quien adquiere mayor importancia y las propiedades que contiene. En este caso componente contiene por un lado a *ParteWebCM* que consiste es una especialización de ParteWeb para poder utilizar las propiedades de *CompWebCM*. Por otro lado, también contiene otro concepto particular de este modelo como es *OperacionWebCM* los cual son especialización de *OperacionWeb*. Esta metaclase contiene un atributo llamado tipoComportamiento, que indica cual va a ser el cometido de la operación, sus valores son: (1) Control (por defecto), se trata de una operación de control introducido por el arquitecto, (2) dominio, si representa a cualquier operación del modelo de dominio, (3) navegación, si representa a cualquier operación del modelo de navegación, (4) proceso, si representa a cualquier operación del modelo de proceso.

Por otro lado, *CompWebCM* y *ParteWebCM* están relacionados con la metaclase *TipoComponenteWeb* la cual les proporciona su tipo. Gracias a que *TipoComponenteWeb* es la raíz de la jerarquía de tipos de componentes, su tipo podrá ser cualquiera de los 19 distintos tipos de componentes definidos.

Es importante destacar, que el elemento MC se beneficia de la extensión del paquete núcleo para obtener aquellas relaciones básicas y no tener que repetirlas dentro de su paquete del metamodelo. Por ejemplo, no es necesario establecer una relación entre *ParteWebMC* y *CompWebCM* ya que sus clases padres, *ParteWeb* y *ComponenteWeb* establecen sus propias relaciones y sus hijos en este caso no necesitan extenderlas.

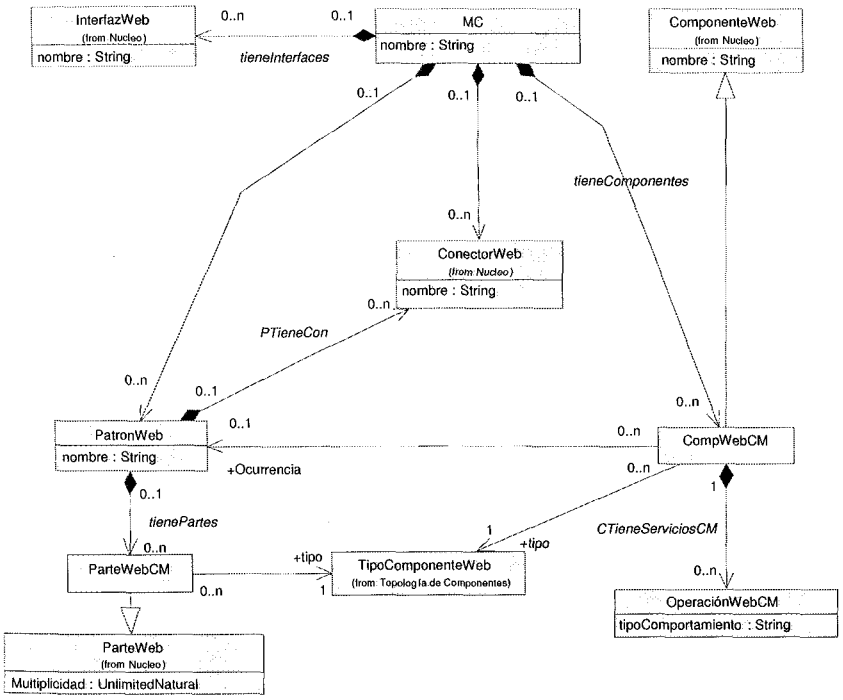

Figura. 33. Paquete de Vista de Configuración.

6.6 Método

A la hora de proponer un método para definir el estilo arquitectónico propuesto por el MC, se deben dar las pautas para modelar que tipos de componentes Web son los adecuados, qué operaciones y atributos contiene, los conectores con los que interactúan y los patrones de arquitectura utilizados. Es por lo tanto necesario, tener un conocimiento general de la arquitectura de la aplicación Web, y por eso es recomendable que el MC sea modelado por analistas o arquitectos conocedores de arquitectura software de una aplicación Web. A continuación, se muestran cuáles son los diferentes pasos que deben realizarse para modelar el MC:

1. Identificar aquellos patrones de arquitectura que se desea aplicar en nuestra aplicación, basándose en las fuerzas de los patrones.
2. Aplicar aquellos elementos *PatronWeb* que ya estuvieran definidos. En caso contrario, se debe definir primero los elementos *PatronWeb*, y a continuación, aplicarlos sobre el modelo de nuestra aplicación Web.
3. Identificar cada uno de los componentes Web que fueran adecuados para nuestra arquitectura. Para ello, se recomienda introducir los siguientes elementos siguiendo estos requisitos:

a) En caso de que la aplicación ofrezca una interfaz de usuario mediante un cliente ligero, debe introducir los componentes *AgenteUsuario* para interactuar con los clientes de la aplicación.
b) En caso de que se muestre la interfaz de usuario mediante un cliente pesado, se introducirá un componente *ObjetoWeb*.
c) Si los interfaces ligeros contienen dinámica, se pueden utilizar los componentes *PaginaServidora*, *ControlladorWeb* y *Vista*.
d) Si se muestran interfaces sin parte dinámica, introducir un componente *PaginaEstática*.
e) Si se debe ofrecer una conectividad con un sistema legado, se han de introducir los componentes *VistaLegada* adecuados.
f) Si se almacena información de forma persistente, se debe introducir los componentes *Almacén* adecuados.
g) Si se almacena la información de forma temporal, se debe introducir los componentes *CacheWeb* adecuados.
h) Si se desea representar entidades de dominio, se tiene que introducir componentes *DatosEntidad*.
i) Para definir la arquitectura de la lógica de negocio, se introducirán según la necesidad los diferentes componentes *ProcesoComponente*, *Entidad*.

4. Introducir los valores etiquetados y *OperacionWeb* de control obligatorios que tienen asociados los componentes Web introducidos previamente.
5. Introducir los *OperacionWeb* de control opcional que se consideren adecuados para los componentes Web.
6. Identificar cuáles son los *OperacionWeb* funcionales, que van a tener asociados cada uno de los componentes Web.
7. Crear los elementos *PuertoWeb* que sean necesarios para cada componente, y que le permitan interactuar con el resto del sistema.
8. Definir el conjunto de *InterfazWeb* que va a contener cada uno de los PuertoWeb del componente.
9. Establecer las conexiones entre componente mediante *ConectorWeb* directos o mediante *ConectorWeb* de tipo *montaje* establecido mediante *IntertazWeb*.
10. Finalmente, se debe establecer los conectores entre los componentes *PatronWeb* y el resto del sistema se ha definido anteriormente.

6.7 El Perfil UML del Modelo de Configuración

Una vez definidos los elementos y el metamodelo del MC, se procede a establecer la estandarización de este modelo para que sea utilizado desde cualquier herramienta UML y además sea más fácil de aprender para aquellos analistas familiarizados con este estándar. Para poder expresar los elementos del MC, se ha optado por definir un perfil a partir del modelo estructura compuesta definido en la nueva especificación de UML, algunos autores como Krobyn [63] y Selic [107] consideran como una de las mayores mejoras entre las novedades introducidas por UML 2.0. Esto se debe a que el modelo de estructura compuesta permite especificar la arquitectura del software siguiendo una más adecuada que la notación basada en componentes de UML 1.X,

introduciendo conceptos para la composición de componentes (p.e. Parte), el concepto de puerto que mejora la encapsulación de los componentes, y los conectores como relación entre componentes, etc.

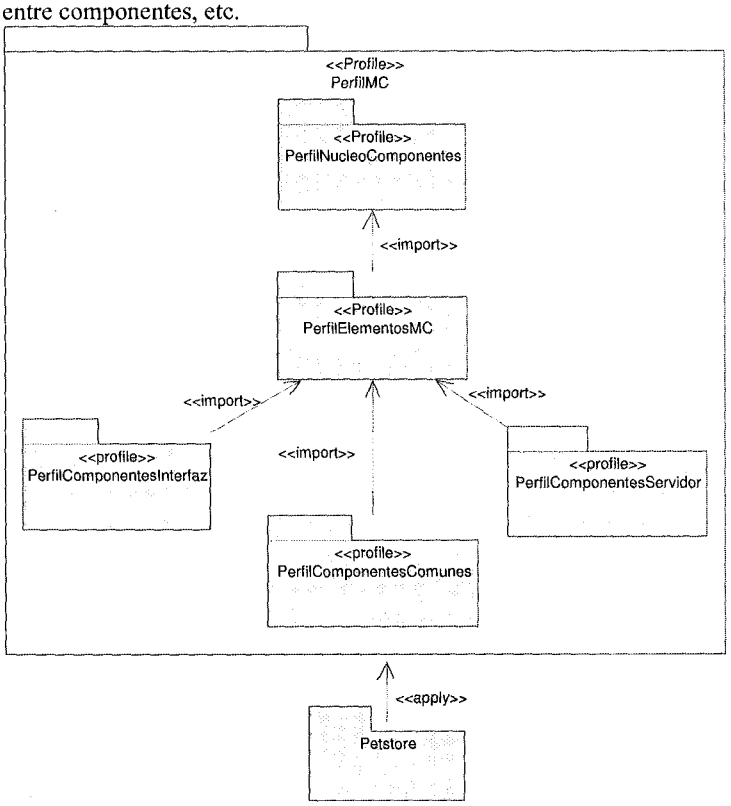

Figura. 34. Estructura del Perfil MC

El perfil MC establece por cada una de las metaclases que se han definido en el metamodelo del MC, un estereotipo que extiende de las metaclases UML. Los estereotipos definidos para el MC introducen la semántica especificada por los elementos del metamodelo para el dominio de las aplicaciones Web, además de expresar mediante valores definidos los diferentes atributos que contienen las metaclases del metamodelo.

A la hora de especificar el perfil MC, se ha estructurado en un conjunto de paquetes estereotipados como <<profile>> que deben ser aplicados para poder utilizar el perfil.

La Figura. 34 muestra el paquete *PerfilMC* que esta estructurado en 5 paquetes de tipo perfil cada uno de los cuáles contiene un conjunto de estereotipos. Así, el paquete inicial *PerfilNucleoComponentes* contiene estereotipos que son comunes y serán extendidos por los modelos de configuración e integración. A continuación, el paquete *PerfilElementosMC* extiende los elementos del núcleo incorporando las especializaciones específicas del MC. Y por último, a partir de *PerfilElementosMC*

definen 3 paquetes: *PerfiComponentesInterfaz, PerfilComponentesComunes,* PerfiComponentesServidor. Cada uno de los paquetes contiene los estereotipos pertenecientes a los diferentes tipos de componentes definidos en las aplicaciones Web. Cada estereotipo de estos subpaquetes se obtiene a partir de la herencia múltiple entre el *CompWebCM* y el tipo de componente que representa (p.e. a partir de *EntidadWeb* se obtendrá un estereotipo con el mismo nombre).

Por otro lado, para poder convertir las relaciones del metamodelo del MC al de UML, deben ser representarlas por medio de restricciones OCL. Estas restricciones junto a una descripción exhaustiva de cada uno de los estereotipos del perfil MC pueden encontrarse en el Apéndice III.

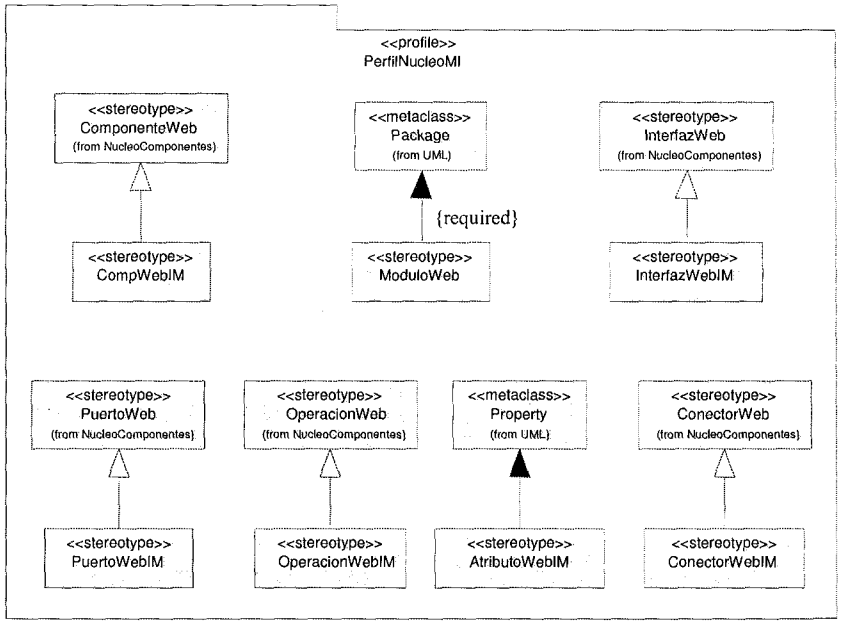

Figura. 35. Paquete PerfilNucleoComponentes.

El perfil MC sigue el paralelismo del metamodelo de componentes (ver Figura. 27), definiendo un paquete *Núcleo* donde están definidos los elementos comunes de las vistas de configuración y de integración. Así, en el perfil MC se ha definido un paquete *PerfilNucleoComponentes* que contiene como estereotipos aquellos elementos definidos dentro del paquete núcleo del metamodelo.

Se puede apreciar en la Figura. 35, todos los estereotipos definidos en el paquete *NúcleoComponentes*, tienen junto a la relación extensión la etiqueta {required}, indicando así que siempre que se cree una instancia de la metaclase de UML referenciada una instancia del estereotipo o una especialización de esta será creada. Por ejemplo, cada vez que se instancia una clase *Class* de UML, se creara una instancia del estereotipo *ComponenteWeb* o alguna de sus especializaciones. Este

perfil permite establecer las bases sobre las cuáles se establece el perfil específico del MC.

Para completar el perfil del núcleo de componentes, se han de representar las relaciones entre las metaclases mediante restricciones, esto podrá ser consultado en el apéndice III.

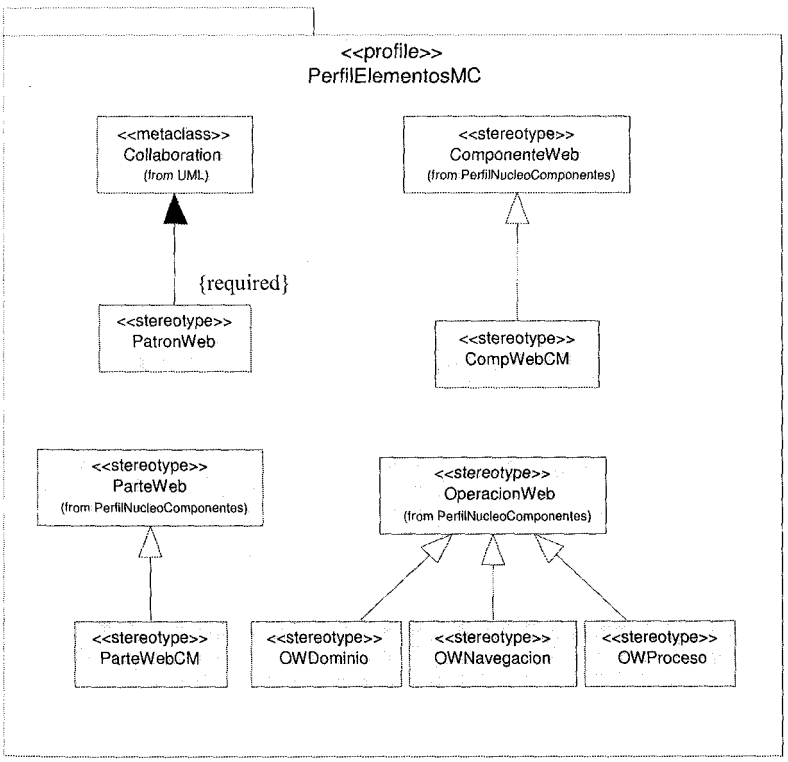

Figura. 36. Paquete PerfilElementosMC

Una vez definidos los elementos comunes del Núcleo de componentes, se procede a definir aquellos elementos que son específicos del modelo MC. Estos elementos se encuentran en el paquete del *PerfilElementosMC* (ver Figura. 36). Siguiendo el paralelismo entre el metamodelo y el perfil, los estereotipos definidos en *PerfilElementosMC* se corresponden con las metaclases nuevas definidas en el paquete VistaConfiguración del metamodelo (ver Figura. 33). En el *PerfilElementosMC* (ver Figura. 36), se han definido cuatro estereotipos. El primer estereotipo es *PatronWeb*, el cual se ha extendido de forma requerida el partir de la metaclase *Collaboration* de UML. Esto implica que una vez aplicado el perfil el elemento solamente aparece como instancia de *PatronWeb*, siendo utilizado para definir patrones de arquitectura Web. Por otro lado, se han extendido estereotipos del *PerfilNucleoComponentes* por aquellos estereotipos que son específicos del modelo MC. La extensión más importante es *ComponenteWeb* por el estereotipo

CompWebCM, que representa la unidad arquitectónica que se define en el nivel de abstracción de MC, diferenciándolo del componente genérico aplicable para cualquier modelo que se ha definido en el Núcleo. Además, como se aprecia en el metamodelo (ver Figura. 33) *CompWebCM* contiene un tipo proporcionado por la metaclase *TipoComponenteWeb*, representándose en el perfil como una restricción OCL del estereotipo con el mismo nombre (ver Anexo A). En las mismas condiciones que *CompWebCM*, se extiende *ParteWeb* en *ParteWebCM*. Por último, se ha extendido el concepto *OperacionWeb* definiendo tres estereotipos que se corresponden a los tres tipos de comportamiento que tiene la metaclase *OperacionWebCM* con el atributo *tipoComportamiento* (ver Figura. 33). *OWDominio* para los operaciones de dominio, *OWNavegación* para los operaciones de navegación, *OWProceso* para los operaciones de proceso. En el caso de que se utilice directamente el estereotipo padre OperacionWeb se tratará entonces de un tipo de comportamiento de control (por defecto).

Siguiendo la descripción del *PerfilMC* mostrado anteriormente (ver Figura. 34), en la parte inferior el perfil de MC contiene tres paquetes que extienden *PerfilElementosMC*, cada uno de los cuáles corresponde con la topología de componentes definida en el metamodelo de componentes (ver Figura. 29, Figura. 30, Figura. 31) para representar la arquitectura del software de las aplicaciones Web. Se ha definido un estereotipo por cada una de las metaclases representadas en la topología de componentes, pero en este caso este estereotipo necesita tener las características propias de un componente de MC. Para ello, cada uno de los estereotipos obtiene por un lado, las propiedades del tipo de componente, y por un lado, extiende del estereotipo *CompWebCM*, obteniendo así las propiedades de la unidad arquitectónica del modelo MC, y pudiéndose aplicar de forma tipada sobre los modelos.

A continuación, se presenta en la Figura. 37 uno de los tres paquetes a modo de ejemplo, concretamente el paquete que contiene los tipos de componentes comunes *PerfilComponentesComunes*. Cada uno de los tipos de componentes se representa como estereotipos, que contiene un conjunto de valores definidos que representan a los atributos obtenidos por los tipos de componentes. Se aprecia además, que sigue una estructura muy similar a la del metamodelo, en la que cada tipo de componente extiende del componente genérico. Pero en este caso el tipo genérico es la propia unidad arquitectónica de MC, es decir, el estereotipo *CompWebCM*. El resto de paquetes, es decir, *PerfilComponentesInterfaz* y *PerfilComponentesServidor* siguen así la misma política que *PerfilComponentesComunes,* para el caso concreto de sus correspondientes tipos de componentes.

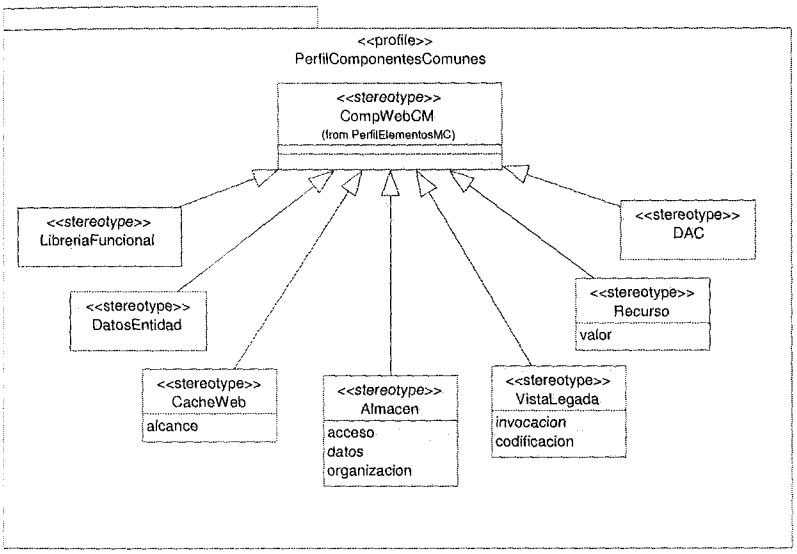

Figura. 37. Paquete de PerfilComponentesComunes

Por último, se presentan los dos casos de estudio que están sirviendo para ejemplificar cada uno de los modelos especificados en el proceso de desarrollo de WebSA.

6.8 Aplicación a un caso de estudio

Una vez definidos los elementos de MC, su proceso y su notación para poder se aplicado a las diferentes aplicaciones Web, a continuación se muestran se continua con los dos casos de estudio que se están especificando desde el capítulo 4 en los que se puede mostrar la aplicabilidad de este modelo. Por un lado, se especifica el modelo MC aplicado a una agencia de viajes online, y por otro lado, la tienda de electrónica Petstore.

6.8.1 MC de la Agencia de Viajes Virtual

Se describe una agencia de viajes que ha decidido poner una aplicación en Internet para la venta de viajes a sus clientes de forma electrónica. Para poder especificar los aspectos arquitectónicos que se representan en el MC, estos son inferidos a partir de los requisitos de accesibilidad y no funcionales. De este modo, esta aplicación presenta los siguientes asunciones arquitectónicas:

- Existe una separación entre la interfaz de usuario y el servidor que ha de adaptarse a diferentes dispositivos. La interfaz puede ser: un teléfono móvil, una PDA o un ordenador con navegador Web. Y además, la presentación debe ser común para todos los usuarios.

- Debido a que los requisitos de navegación son diferentes para cada dispositivo, se localiza la navegación de una forma independiente al código mediante un fichero externo o almacén.
- Continuamente la aplicación debe presentar ofertas de diferentes viajes y la interfaz de usuario debe modificarse. Esto conduce a un mantenimiento continuo de la interfaz de usuario.
- Debido a que la previsión de número de clientes en la agencia de viajes no es muy elevada, se opta por realizar una lógica de negocio no distribuida.
- El almacenamiento de la información de la agencia de viajes se realiza mediante una base de datos remota.

La Figura. 38 muestra el modelo MC que representa la configuración la aplicación Web de la agencia de Viajes. En la parte frontal del modelo MC se encuentran tres tipos diferentes de componentes *AgenteUsuario*, que representan los diferentes componentes cliente que interactúan con el usuario final. Los tres tipos de *AgenteUsuario* contienen el atributo *tipoDispositivo* indicando los tres valores distintos: navegadorWeb, PDA y móviles. Para poder devolver los diferentes tipos de páginas que requiere cada *AgenteUsuario*, el componente *PaginaServidora* dispone de tres interfaces cada uno de los cuáles corresponde se comunica con cada dispositivo. Además, contiene dos instancias de OperacionWebCM de tipo control, que contiene dos parámetros, el primero donde se envía la información desde la página cliente, y el segundo de tipo string que indica el tipoDispositivo de origen, y en función de ello tendrá un tipo de presentación. Una vez le llega a la *PaginaServidora* una petición de operación de dominio, entonces este componente llama al componente *EntidadWeb* a su OperacionWebCM llamado servicioLogica estereotipado como OWDominio. Esto indica que representa a cualquier operación cuyo comportamiento sea realizar una tarea de lógica de negocio de la aplicación Web.

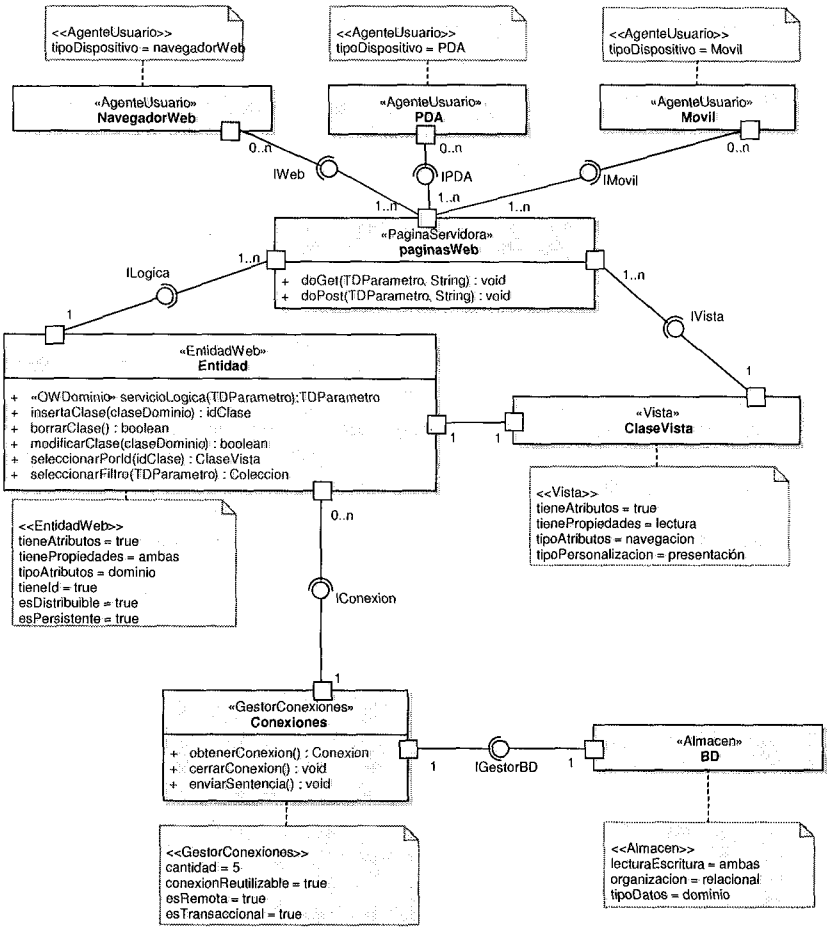

Figura. 38. MC de la Agencia de Viajes

Además, contiene varias instancias OperacionWebCM de tipo control llamadas *insertarClase*, *borrarClase*, *modificarClase*, *seleccionarPorId* y *seleccionar-PorFiltro*, que contienen la funcionalidad para realizar las tareas de actualización y consulta de la base de datos. Para ello *EntidadWeb* invoca al componente *GestorConexiones* y a sus operaciones (*obtenerConexion, cerrarConexion e invocarSentencia*) para gestionar la conexión a la BD y invocar sentencias sobre ella. Además, sus atributos indican que *GestorConexiones* contiene 5 conexiones remotas que son reutilizables y remotas, y permiten definir transacciones. La BD es representada por un componente Almacén en el que ha establecido que es de lectura-escritura, de organización relacional y almacena datos que proceden del dominio. Por último, cuando el componente *EntidadWeb* devuelve los resultados lo hace almacenándolos un componente Vista, el cual contiene únicamente los atributos y propiedades de lectura que permiten a las páginas mostrar la información.

6.8.2 MC de la Tienda de Animales Electrónica: Petstore

El ejemplo de la tienda de animales definido por SUN Microsystems [115], es un sistema que contiene las mejores prácticas y uso de patrones para el desarrollo de una aplicación J2EE. Como se ha comentado anteriormente, MC permite representar un modelo independiente de la plataforma destino, permitiendo que se pueda definir todos los patrones de arquitectura que define Petstore, y reutilizar estos conceptos para cualquier otra aplicación que utilice los mismos independientemente de la plataforma en la que se implemente. Debido a que MC permite estructurar el modelo utilizando el concepto de PatronWeb, a continuación se realiza una descripción general de la arquitectura de Petstore, para posteriormente profundizar en cada uno de los patrones que se han ido aplicando.

La Figura. 39 muestra una vista general del MC de la arquitectura de Petstore, la cual está constituida por un conjunto de componentes y conectores descritos a continuación.

En la parte frontal del modelo se encuentra un componente AgenteUsuario llamado *Navegador* que representa al navegador Web que accede como cliente a nuestra aplicación. Este componente recibe las peticiones de los clientes y realiza la renderización de las páginas estáticas. A continuación, *Navegador* realiza las invocaciones por medio de la InterfazWeb *manejadorCliente* al componente *MVC2* que representa al PatronWeb Modelo-Vista-Controlador 2. Este componente *MVC2* recibe dichas peticiones y establece la definición de la interfaz por medio de la interfaz *DefiniciónPantalla* ofertado por las paginas servidoras. En este caso, las páginas Web están estructuradas por una PaginaServidora llamada *PlantillaBase*. Esta plantilla base está especificada siguiendo el patrón MasterTemplate definido por [24]. Siguiendo la estructura de este patrón (ver Figura. 39), se ha definido una pagina principal que contiene a su vez instancias de PaginaServidora cada una de las cuáles define una parte de la pantalla, componiéndose de las paginas *IndiceSuperior*, *Banner*, *Pie* y *Cuerpo*. Cada una de estas páginas se define mediante el elemento ParteWeb, que establece que las instancias viven dentro de la plantilla. Cada una de las ParteWeb tiene una multiplicidad indicando cuantas diferentes pueden vivir dentro del componente *PlantillaBase*. Así, cuenta con dos posibles Banners, y un número no determinado de paginas de tipo Cuerpo (definido con la multiplicidad 1..*), la cual representar el cuerpo de cada página en cada vista navegacional. De manera que las instancias de Cuerpo serán establecidas por las clases navegacionales del modelo de navegación.

Por otro lado, una vez el componente *MVC2* ha recibido navegaciones de servicio realiza una invocación al componente que representa el patrón Façade por medio de la interfaz *ILogicaNegocio*. El componente patrón Façade que se tratará con más detalle en la Figura. 41, contiene los componentes que constituirán la lógica de negocio. Esta lógica de negocio es ofertada tanto a la interfaz de usuario de nuestra aplicación Web como a un sistema legado representado por el componente *PedidosExternos* de tipo VistaLegada, que permite la comunicación de forma asíncrona con los proveedores.

Por otro lado, los componentes de lógica de negocio invocan al componente CAD para realizar la conversión de la información almacenada en los componentes, a los registros de la base de datos. En este caso, CAD oferta un conjunto de operaciones llamadas (*insertarClase*, *borrarClase*, *modificarClase*, etc.) que permiten la creación, modificación y consulta de los registros de una determinada clase. Por último indicar

que se dispone de un GestorConexiones que permite el acceso a una base de datos relacional llamada *BD* mediante 10 conexiones remotas, reutilizables y transaccionales.

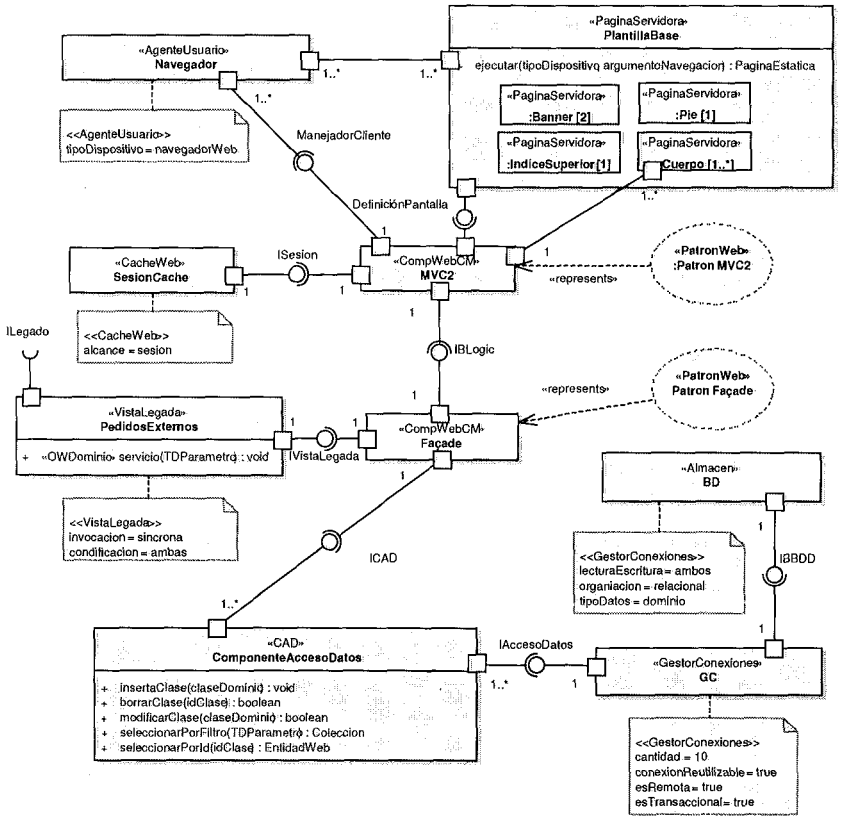

Figura. 39. MC General de Petstore

6.8.3 MC del Patrón Modelo-Vista-Controlador 2

Seguidamente se define el PatronWeb *MVC2* que ha sido aplicado al ejemplo Petstore. Este patrón (ver Figura. 40) esta constituido por un componente ControladorWeb llamado *ControladorPrincipal* que tiene tres InterfacesWeb, uno para recibir las peticiones del AgenteUsuario llamado *ManejadorCliente,* otra interfaz para la definición de las diferentes páginas Web llamado *IDefiniciónPantalla*, y por último, *ISesion* que le permite interactuar con el componente CacheWeb que almacena la sesión. El ControladorWeb tiene tres operaciones para la recepción de peticiones del cliente: (1) *doPost* para la invocación de enlaces de servicio y (2) *doGet* para la recepción de enlaces de travesía. Además, dispone de un método llamado (3) *leer* que se encarga de consultar al Almacén llamado *AlmacenNavegación* donde están ubicados las rutas navegacionales de la aplicación. El controlador se comunica con un conjunto de componentes DatosEntidad llamados *Modelo* que

representa a las clases del modelo de dominio en la interfaz. Por ello, el componente DatosEntidad contiene los atributos pertenecientes al dominio y las propiedades para consultar y modificar los atributos (*tienePropiedades = ambas)*, y además oferta un OperacionWeb estereotipado como OWNavegacion que representa a cualquier enlace de servicio del modelo de navegación. Para poder llevar a cabo estos enlaces de servicio, el componente se comunica con la lógica de negocio por medio de la interfaz *IBLogic*. Por último, se cuenta con un conjunto de ParteWeb de tipo Vista llamado *ComponenteVista,* encargadas de representar el estado de las ClaseNavegación y de ofertarlo a las instancias de PaginaServidora, mediante la interfaz *IDatosVista*. Para ello, *ComponenteVista* contiene los atributos definidos en cada clase de navegación, y las propiedades de lectura y escritura para obtener y modificar el valor de estos atributos.

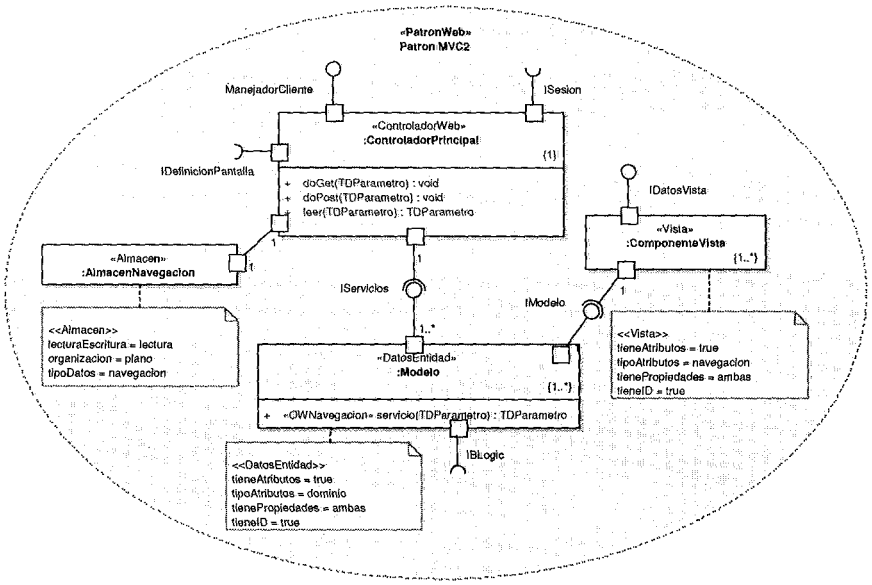

Figura. 40. PatronWeb Modelo-Vista-Controlador 2

6.8.4 MC del Patron Façade

El PatronWeb llamado *Patrón Façade* (ver Figura. 41) está basado en el patrón con el mismo nombre definido por [36]. Este patrón está adaptado a los componentes distribuidos, y para ello, define un grupo de uno o más ComponenteProceso que realizan la tarea de fachada o receptor de las peticiones de la interfaz de usuario a través de la interfaz *IBLogic*. Este ComponenteProceso llamado *LogicaNegocio* se trata de un componente sin estado, que permite que puedan crearse multitud de clones de este componente que podrán atender a gran número de peticiones. Además, se ha definido con el atributo *esTransaccional=true* con lo que se encargará de realizar la tarea de inicio y fin de las transacciones en los servicios de lógica de negocio. Por otro lado, contiene dos operaciones de factoría, para crear y borrar componentes de proceso, y ofertará mediante la OperacionWeb estereotipada como OWDominio,

aquellas operaciones que sean definidas en las clases de dominio. Por otro lado, para poder realizar las operaciones de dominio, el ComponenteProceso debe invocar al componente EntidadWeb que contiene el estado de una determinada entidad de dominio. EntidadWeb oferta dos interfaces, *InterfazFactoría* que provee las operaciones de creación y borrado de los componentes de entidad, e *InterfazRemoto* que proporciona las operaciones de modificación del estado del componente EntidadWeb.

El componente EntidadWeb llamado *Entidad* se define como un componente distribuido con lo que puede vivir en un servidor distribuido, y contiene los atributos y la identidad de una determinada clase de dominio. En la parte dinámica, el componente EntidadWeb contiene las propiedades para obtener y modificar los atributos, el método *existe* para comprobar que hay una determinada instancia de EntidadWeb, los métodos de factoría crear y borrar, y los métodos que permiten hacer persistente el estado de una entidad llamados cargar y almacenar. Para comunicarse con parte encargada de la persistencia, el componente EntidadWeb accede a la interfaz IAccesoDatos.

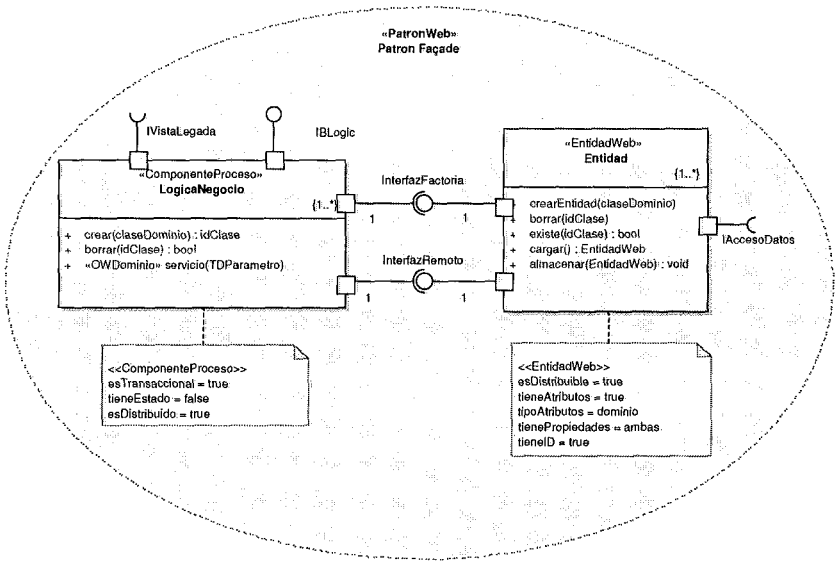

Figura. 41. PatronWeb Façade

6.9 Conclusiones del Capítulo

En este capítulo se ha presentado una nueva vista arquitectónica en la fase de análisis llamada Vista de Configuración. Esta vista esta constituida por el modelo de configuración (MC), el cual define un nuevo estilo arquitectónico centrado en el dominio de las aplicaciones Web. MC permite expresar de una forma sencilla y clara cual es la configuración de nuestra aplicación Web. El estilo arquitectónico de MC, es fruto del estudio de la familia de aplicaciones Web, en el cual se ha identificado los

Volver al índice/Tornar a l'índex

diferentes tipos de componentes que configuran una aplicación Web, basándose en trabajos previos como [24], [96], 31] y en la propia experiencia.

A continuación, se han definido los elementos que permiten representar MC, mediante un metamodelo MOF. Este metamodelo forma parte del metamodelo de WebSA y se presenta de forma unificada en el llamado *metamodelo de componentes* que recoge tanto los elementos del modelo MC como del modelo de integración (MI). La representación de MC se ha basado en la definición de un perfil UML sobre el modelo estructurado compuesto. Finalmente, se ha aplicado MC sobre los casos de estudio Agencia de Viajes y Petstore. Sobre ellos, MC ha servido no solo para la representación temprana de la arquitectura de las dos aplicaciones Web. Sino que en Petstore se ha demostrado que MC permite la representación de patrones de arquitectura, permitiendo así dotar de las mejores soluciones desde las fases iniciales.

CAPÍTULO 7

Modelo de Integración

7.1 Motivación

En el flujo de trabajo de análisis del proceso de desarrollo de WebSA, la funcionalidad y la arquitectura del software se definen en vistas separadas para obtener así entre otras cosas, una mejor reutilización de modelos, incorporación de métodos conocidos y una mejor separación en roles y tareas que permita una aceleración del proceso de desarrollo de una aplicación Web. Sin embargo, ambos aspectos se han de unificar para obtener finalmente la aplicación Web completa. Esta unificación de los modelos de análisis se realiza mediante la transformación T1 en el llamado modelo de Integración (MI).

El MI se sitúa dentro del proceso de desarrollo de WebSA en la siguiente tarea después del análisis, es decir, en el flujo de trabajo de diseño software. El diseño del software se ha definido por Pressman [98] como "*la primera fase técnica que permite traducir los requisitos analizados de un sistema en una representación del software* y *tras posteriores refinamientos conducirá a una representación de diseño muy cercana al código fuente*". El diseño software y por lo tanto, el modelo de integración debe ser una visión completa de la aplicación Web pero que abstrae todavía ciertos aspectos que están ligados exclusivamente a la implementación.

Para la obtención del MI, la transformación T1 realiza un proceso de descomposición en el cual los componentes de mayor granularidad definidos en la fase de análisis arquitectónico, tanto en el modelo de subsistemas como en el modelo de configuración son descompuestos según sus tipos y completados con la funcionalidad proporcionada por el análisis funcional, obteniendo así un nivel de detalle que representa con una paridad 1 a 1 a los componentes de la implementación.

El MI posee la característica adicional que abstrae cualquier aspecto relacionado con la plataforma constituyéndose el modelo de Integración por modelos independientes de plataforma, lo que MDA denomina PIM. El objetivo es que el modelo de Integración sea equivalente para cualquiera de las plataformas de ejecución que pueda encontrar en el mercado de las aplicaciones Web, pudiendo posteriormente establecer diferentes transformaciones PIM a PSM que introduzcan los aspectos

dependientes de plataforma como J2EE, .NET, PHP, etc. Se obtienen así múltiples implementaciones de un mismo diseño e incluso la posibilidad de definir sistemas heterogéneos formados por componentes pertenecientes a diferentes plataformas

Sin embargo, la definición de un artefacto dentro del proceso de desarrollo Web como el MI difiere sensiblemente de otras aproximaciones dentro de la Ingeniería Web como OOH [19], OOHDM [20], WebML [20]. Debido a que estas aproximaciones sitúan en el flujo de trabajo de diseño, otros artefactos de modelado relacionados únicamente con la funcionalidad como son el modelo de dominio, navegación y proceso. Sin embargo, estos modelos están más vinculados al espacio del problema donde se definen los requisitos funcionales y no representan a los componentes que constituyen la aplicación final. Por lo tanto, existe un gap o salto semántico en estas aproximaciones entre sus modelos de diseño y la implementación final. Este gap es cubierto por WebSA mediante el modelo de Integración.

Otro motivo por el cual se introduce un modelo de integración y no se salta directamente a la implementación es por su papel fundamental para obtener una aplicación Web de calidad. Según Pressman [98], la importancia del diseño se puede sentar con una única palabra "calidad". El diseño produce las representaciones del software de las que puede evaluarse su calidad y es la única forma mediante la que se traduce con precisión los requisitos del cliente en un producto o sistema terminado. En este sentido, WebSA mediante la transformación T1 convierte los modelos de análisis en los modelos de diseño, consiguiendo formalizar la trazabilidad entre los requisitos representados en el análisis y la solución aportada en la fase de diseño.

Por lo tanto, el MI juega un papel muy importante en el proceso de desarrollo de WebSA, debido a que ciertas características de la aplicación son identificables únicamente cuando se consideran los aspectos funcionales y arquitectónicos juntos. Por ejemplo, para poder determinar la granularidad de los componentes de lógica de negocio, es necesario que se conozcan ambos aspectos, la arquitectura (p.e. si la lógica de negocio va a ser distribuida) y su funcionalidad (cuáles son las tareas que tiene asignadas). Esto permite identificar aspectos no deseados en la fase de diseño, que son detectables antes de llegar a la implementación, reduciendo el tiempo de detección de errores y de su corrección. Cuando se encuentren aspectos no deseables dentro de alguno de los modelos de la Vista de Integración y se desee solucionar, es importante recordar que han sido obtenidos por medio de la transformación T1, y si son alterados entonces perderían la trazabilidad con la fase de análisis. Por lo tanto, WebSA propone que en lugar de modificar el MI directamente, utilizar dos mecanismos: (1) si ha sido un error de análisis se modifican los modelos de la fase de análisis y se vuelve a transformar, (2) sobrescribir reglas de la transformación T1 con reglas específicas para este sistema en concreto, que permitan obtener resultados a la carta. Este último mecanismo es el más adecuado cuando se desea tratar aspectos de diseño que son claramente específicos de nuestra aplicación (p.e en el dominio de una aplicación Web de comercio electrónico se desea que la clase Carrito no tenga persistencia).

El MI se define con la premisa de estandarización propuesta por MDA. Así, el MI está formalizado mediante un metamodelo MOF que representará el conjunto de elementos y de relaciones que constituye MI. Además, este metamodelo permitirá establecer las transformaciones del análisis al diseño, ya que son el destino las

diferentes reglas de transformación que establece la transformación T1 y del diseño a la implementación ya que es el origen para la transformación T2.

Además, siguiendo la estandarización, para definir la notación del MI se ha definido un perfil UML basado en los modelos estándares de arquitectura como el modelo de estructura compuesta de UML 2.0. El perfil permite representar el modelo de integración en cualquier herramienta que tenga soporte UML 2.0.

A continuación se recorrerán todos los aspectos del MI, sus elementos, su metamodelo, su método de aplicación y la aplicación del MI a casos de estudio. Por último, se obtendrán las conclusiones del capítulo.

7.2 Modelo de Integración

El Modelo de Integración (MI) es un modelo de arquitectura que define el diseño de la estructura de nuestra aplicación, proponiendo para ello la definición completa de los componentes y las relaciones que constituyen la aplicación Web en todas sus capas. Se basa para su definición en los estilos arquitectónicos propuestos por MC y MS, restringiéndose a los elementos arquitectónicos que ambos estilos definen. Para ello, propone un nuevo tipo de componente llamado *ModuloWeb* que permite representar los subsistemas que el MS propone pero en la fase de diseño, estableciendo las conexiones entre las diferentes capas, y conteniendo el conjunto de elementos asociados a dicha capa. Los componentes definidos en este nivel, llamados *CompWebIM* incorporan tanto los aspectos arquitectónicos como funcionales, y no solo eso, para algunos tipos componentes definidos en el MC (p.e. EntityWeb, EntityData, etc.), se obtendrán n instancias de componentes en el MI que corresponden a las n instancias de las clases definidas en un modelo funcional. Por ejemplo, se obtienen n instancias de *CompWebIM* del tipo EntityWeb a partir de las n instancias de Clase del Modelo de Dominio.

El nivel de detalle del modelo de diseño MI es cercano a la implementación, y similar al de propuestas como WAM [24]. Sin embargo, a diferencia de dicha propuesta, este modelo no es modelado directamente, sino que WebSA lo obtiene por medio de la transformación T1. Con ello se obtienen dos ventajas: (1) el consiguiente ahorro de modelado ya que el número de elementos modelado en la fase de análisis es sensiblemente inferior al número de elementos que supone la representación de todos los elementos de la fase de diseño, y (2) asegurar una exacta trazabilidad en la fase de diseño respecto a los aspectos de arquitectura y a los aspectos funcionales especificados en la fase de análisis.

Así, este modelo pretende representar el diseño de la solución final, abstrayendo la representación de los aspectos propios de una plataforma de implementación. En este sentido, MI representa la estructura de las aplicaciones implementadas en cualquier plataforma, manteniendo una paridad 1 a 1 entre los componentes definidos en ella y los componentes de la implementación final.

Para formalizar el MI, se ha definido un metamodelo de Integración, que define únicamente los conceptos que son propios de dicha Vista. Sin embargo, hay que destacar que este metamodelo está íntimamente ligado al metamodelo de configuración, ya que comparte ciertos elementos definidos en ambas vistas. Esto

permite reutilizar muchos de los conceptos, evitar duplicidades y reducir el tamaño de ambos metamodelos.

Por otro lado, el MI conserva en la fase de diseño la jerarquía de tipos definida para el modelo de configuración, ya que por medio de las transformaciones se almacena en los componentes de integración la información sobre su tipo (p.e. PaginaServidor, ControladorWeb, etc.). El tipo del componente aporta información muy valiosa a la hora de transformar desde el diseño a la implementación, ya que cada tipo de componente está asociado a una implementación distinta.

Para la representación del MI, se define un perfil sobre el Modelo de Estructura Compuesta de UML 2.0. Este perfil esta formado por un conjunto de 6 estereotipos, cada uno de los cuáles extenderá de las metaclases de UML o de aquellos estereotipos que ya se ha definido para MC.

En las siguientes subsecciones, se presentan los elementos más importantes en el MI y su notación. Primero se formaliza MI mediante el metamodelo de integración. Seguidamente se establece el método a seguir para representar el modelo y se definirá su estandarización mediante el perfil UML de integración. Por último, se aplicará MI a un caso de estudio.

7.2.1 Metamodelo del Modelo de Integración

Siguiendo la arquitectura definida por el metamodelo de WebSA, en el metamodelo del MI se definen únicamente aquellos elementos que son nuevos para este modelo, reutilizado aquellos conceptos que ya fueron propuestos por el otro modelo componentes estructurado como es el MC. La Figura. 27 especifica la estructura del metamodelo de WebSA, y muestra el paquete *modelos de componentes,* en el se encuentra el subpaquete *Vista de Integración* y *Núcleo*. En este caso, el metamodelo del MI se encuentra dentro del paquete de *Vista de Integración* y comparte con el MC un conjunto de metaclases comunes localizadas en el paquete *Núcleo*. En el paquete *Núcleo* (ver sección 6.1.7.1) se definen las metaclases como *ComponenteWeb, ConectorWeb, InterfazWeb*, etc. que constituyen los conceptos básicos que rigen un modelo de componentes. Además, el MC y el MI comparten la misma jerarquía de componentes localizada en el subpaquete *Topología de componentes* donde se definen los 19 tipos de componentes localizados en el dominio de las aplicaciones Web (ver Apéndice II).

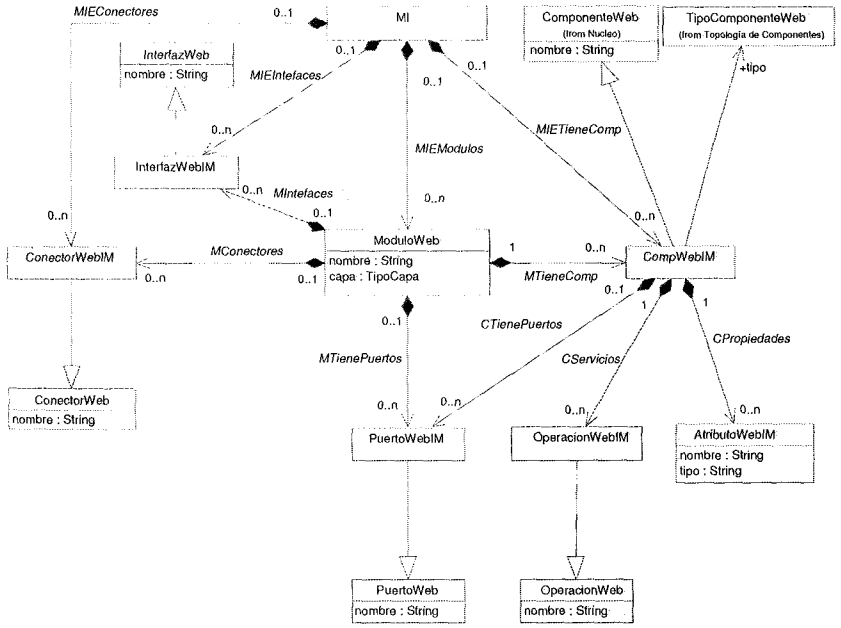

Figura. 42. Metamodelo de MI

La Figura. 42 presenta el metamodelo del MI. El elemento principal de este metamodelo es la metaclase *MI* que representa al concepto de modelo conteniendo a aquellos elementos que participan en él. Los elementos que contiene el MI son: (1) *ConectorWebIM* que representa un enlace de comunicación en MI y que es una especialización de *ConectorWeb* definido en el paquete Núcleo, (2) *InterfazWebIM*, que consiste en un elemento que proporciona o requiere funcionalidad de un componente de integración. Esta metaclase también es una especialización de *InterfazWebIM*. (3) *CompWebIM* que representa el concepto de componente Web en la fase de diseño, y para ello hereda de *ComponenteWeb* las propiedades estructurales que este posee, y obtiene el tipado de la jerarquía de componentes Web por medio de una asociación como *TipoComponenteWeb* de la jerarquía de componentes mediante el rol *tipo*. Además, *CompWebIM* contiene tres nuevas propiedades que son específicas de MI, *PuertoWebIM* como punto de interacción con el exterior, *OperacionWebIM* que representa a una propiedad de comportamiento del componente y *AtributoWebIM* que representa su propiedad estática. El último elemento que contiene MI es (4) *ModuloWeb* que como se ha explicado previamente es un contenedor que representa a un SubsistemaWeb en la fase de diseño, para ello posee los componentes y sus relaciones, por lo tanto, en el metamodelo un *ModuloWeb* tiene una relación de composición con *CompWebIM* y a todas sus propiedades, y además con *ConectorWebIM* e *InterfazWebIM* para establecer las relaciones de estos componentes. Por último, un *ModuloWeb* para poder relacionarse con otras instancias de *ModuloWeb* contiene un punto de interacción como el *PuertoWebIM*.

7.2.2 Elementos de MI

Esta sección describe los elementos que se utilizan en el MI. Sus componentes principales son el *ModuloWeb* y *CompWebIM*. Además, se definirán sus propiedades, relaciones y las especializaciones que completan los componentes de diseño definidos.

7.2.2.1 ModuloWeb

Es el elemento arquitectónico de mayor granularidad del MI. Un *ModuloWeb* es un macrocomponente que realiza la tarea de contener y vincular a un conjunto de componentes Web que realizan una funcionalidad lógica por si mismos. Esta funcionalidad normalmente se asocia a una capa lógica, que en WebSA se corresponde al *SubsistemaWeb* definido en la fase de análisis por MS. Cada instancia contiene un conjunto de *CompWebIM* y las relaciones que forman parte del *ModuloWeb*. Por otro lado, un *ModuloWeb* posee elementos que permiten la interacción con otras instancias de componentes, encapsulando así su funcionalidad, y ofertando únicamente aquella que se considere visible. Los elementos *PuertoWebIM* e *InterfazWebIM* permiten ofertar y requerir funcionalidad, y *ConectorWebIM* establecer relaciones con otras instancias de *ModuloWeb*.

Cada *ModuloWeb* posee dos propiedades:

- **nombre**: identificador de la instancia del modulo.
- **capa**: nombre del SubsistemaWeb que se encarga de representar este componente en el diseño.

Un componente *ModuloWeb* utiliza la misma notación que la metaclase *Class* del modelo estructura compuesta de UML, ya que permite la composición de elementos que *ModuloWeb* requiere. En la Figura. 43, se puede apreciar un ejemplo de el elemento *ModuloWeb*.

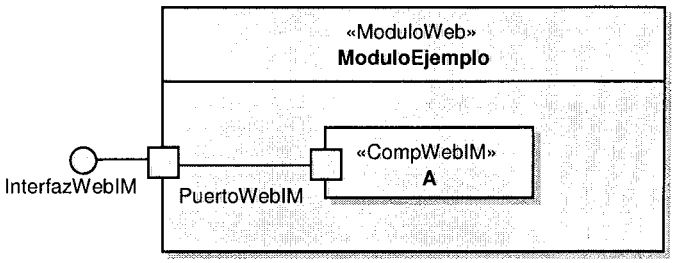

Figura. 43. Notación del ModuloWeb

La Figura. 43 muestra la mínima expresión de un componente *ModuloWeb* llamado *ModuloEjemplo*. Este *ModuloWeb* contiene un único componente *CompWebIM* de llamado *A* (ver sección 7.2.1.2). Además, el *ModuloWeb* posee un *PuertoWebIM* o punto de interacción con el resto del sistema, que permite que el componente A se comunique con el resto de módulos. El *PuertoWebIM* a su vez oferta una *InterfazWebIM* que oferta la funcionalidad a otros módulos.

7.2.2.2 CompWebIM

CompWebIM representa la forma básica de encapsulación en el diseño de la estructura de una aplicación Web, abstrayendo únicamente los aspectos que son dependientes de una plataforma concreta. El componente posee tres aspectos principales: (1) el tipo de componente del dominio de las aplicaciones Web, obtenido a partir de la topología propuesta por el estilo arquitectónico MC (ver sección 6.3), (2) la funcionalidad obtenida a partir de los modelos de la vista funcional (dominio, navegación y proceso), y (3) la interacción del componente con el resto del sistema que permite definir la estructura y el comportamiento de la aplicación Web.

La notación de un *CompWebIM* es la siguiente:

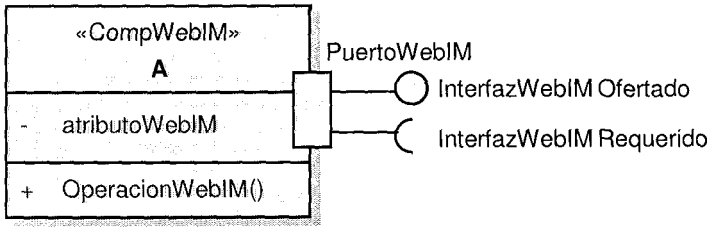

Figura. 44. Notación del CompWebIM

CompWebIM mantiene la notación de la clase estructura de UML (ver Figura. 44). En el interior de su caja, contiene los elementos *AtributoWebIM* y *OperacionWebIM* que representarán los aspectos funcionales. Y en su contorno, como puntos de interconexión con otros componentes contiene elementos *PuertoWebIM*, los cuáles pueden contener o no un conjunto de *InterfazWebIM* que ofertarán o requerirán funcionalidad a otros componentes.

7.2.2.3 PuertoWebIM

PuertoWebIM es el punto de interacción de un *CompWebIM* o un *ModuloWeb* y su entorno. Se encarga de desacoplar los aspectos internos del componente de la interacción con otros componentes, haciendo así al componente reutilizable en cualquier entorno que sea conforme con las restricciones de interacción impuestas por los *PuertosWebIM*. Un *CompWebIM* y un *ModuloWeb* solamente pueden comunicarse con el exterior a través de los *PuertosWebIM*. Su notación se representa por medio de un pequeño cuadrado o rectángulo cuando tiene más de una *InterfazWebIM* (ver Figura. 44 para *CompWebIM* y la Figura. 43 para el *ModuloWeb*).

7.2.2.4 InterfazWebIM

Una *InterfazWebIM* representa la funcionalidad que el *CompWebIM* ofrece o requiere al resto del sistema, para poder realizar su tarea, siendo una especialización del concepto *InterfazWeb* definido para el MI. Cada *InterfazWebIM* está asociado a un *PuertoWebIM* que especifica la naturaleza de las interacciones que pueden ocurrir sobre él. Por un lado, los interfaces requeridos de un *PuertoWebIM* caracterizan las peticiones las cuáles pueden hacer el *CompWebIM* y el *ModuloWeb* a su entorno. Por otro lado, los interfaces ofertados por un *PuertoWebIM* caracterizan las peticiones que el entorno hace al *CompWebIM* o al *ModuloWeb*.

La *InterfazWebIM* ofertada se representa con la notación lollipop, que consta de una línea que tiene como extremo un círculo completo, y la *InterfazWebIM* requerida es una línea que tiene como extremo un semicírculo. Su notación puede apreciarse en la Figura. 44 para el *CompWebIM* y la Figura. 43 para el *ModuloWeb*.

7.2.2.5 ConectorWebIM

El *ConectorWebIM* especifica un enlace que permite la comunicación del sistema entre dos o más *CompWebIM*, se define como una especialización de *ConectorWeb* para la fase de diseño. La comunicación se establece a través de los *PuertosWebIM*. Cada ConectorWebIM tiene asociado dos *FinalConectorWeb*. *ConectorWebIM* tiene la misma notación que su clase padre *ConectorWeb*, se puede establecer la conexión directamente sobre los puertos o hacerlo mediante interfaces con un conector tipo montaje.

7.2.2.6 AtributoWebIM

Un *AtributoWebIM* es una propiedad estática del componente CompWebIM. Un *AtributoWebIM* tiene asociado un nombre y un tipo. La procedencia de un *AtributoWebIM* se debe a la parte funcional a partir de los atributos del modelo de dominio o del modelo de navegación.

7.2.2.7 OperacionWebIM

Un *OperacionWebIM* es una propiedad de comportamiento del componente *CompWebIM*. Un *OperacionWebIM* tiene asociado un nombre y un conjunto de parámetros de entrada o/y salida. Un *OperacionWebIM* es obtenido a partir de dos aspectos: (1) Funcional, el *OperacionWebIM* se obtiene a partir de una Operación del modelo de dominio o a partir de un *ServicioNavegacional* del modelo de navegación (2) Control, aquellas operaciones definidas en el análisis arquitectónico por MC son reflejados en el diseño de MI. Cada *CompWebIM* hace público la operación por medio de un *PuertoWebIM* o una *InterfazWebIM*.

7.2.2.8 ParametroWeb

Un ParametroWeb es la especificación de un argumento usado para pasar información dentro y fuera de la invocación a un OperacionWebIM. Cada ParametroWeb contiene un nombre y un tipo. Se obtendrán las instancias de ParametroWeb a partir de los Parámetros definidos a las Operaciones del modelo de dominio y a los parámetros de las instancias de ServicioNavegacional.

7.2.3 Método del Modelo de Integración

Como ya ha sido indicado, MI es un modelo que se obtiene mediante un proceso de transformación automática en la que partiendo de los modelos de análisis funcional y análisis arquitectónico se combina la información para dar lugar a este modelo. Por lo tanto, no es necesario ni conveniente modificar el modelo directamente ya que se produciría dos efectos: (1) una perdida de trazabilidad entre el análisis y el diseño con lo que no se asegura la representación de los requisitos recogidos en la fase de

análisis, (2) una posible introducción de errores por la intervención humana, que en el caso de las transformaciones es evitada.
Sin embargo, se recomienda una revisión del MI por parte del diseñador, para que se examine que los elementos recogidos por las transformaciones son los correctos y no existan errores que no hayan sido detectados en los modelos de análisis.

Consiguientemente, las dos acciones a tomar para solucionar los errores encontrados son modificar los modelos de análisis responsables de estos errores e introducir transformaciones específicas del sistema que especialicen la transformación T1 para aquellos casos que el diseño obtenido no es considerado óptimo por el diseñador.

A continuación se muestran dos ejemplos en los que se presentan soluciones diferentes:

- **Ejemplo 1**. El diseñador detecta que la granularidad de los componentes no es la adecuada porque existe un componente distribuido por cada clase del modelo de dominio cuando se utiliza la transformación T1. Esto produce que las comunicaciones entre componentes sean numerosas, aumentando mucho el tráfico y reduciendo su rendimiento. Para solucionar este problema, se debe abordar desde la incorporación o modificación de las transformaciones que tratan la creación de los componentes de dominio. Para ello, se define una nueva transformación que hereda de la transformación T1 y establece un nuevo criterio. Por ejemplo, la nueva transformación establece que las clases compuestas o relacionadas por composición con otras clases, se localicen en un solo componente.
- **Ejemplo 2**. El diseñador detecta que el componente DatosEntidad que reside en la capa de control de usuario no contiene los atributos necesarios para mostrar la información de una determinada página Web. Esto obliga a que se modifique en el modelo de navegación, los atributos de la ClaseNavegacional que dio lugar al componente.

Por lo tanto, es fundamental una revisión del diseño previa a la ejecución de la transformación que lleve a la implementación final.

7.2.4 El Perfil UML del Modelo de Integración

Una vez definidos los elementos y el metamodelo MOF del MI, se establece la estandarización de la notación de este modelo para que sea utilizado desde cualquier herramienta UML y sea más fácil de aprender para aquellos diseñadores familiarizados con el estándar. Para poder expresar los elementos del MI, se ha optado por definir un perfil a partir del modelo estructura compuesta definido en la nueva especificación de UML 2.0. Como ha ocurrido con los perfiles anteriores, la definición del perfil MI se basa en la justificación establecida por Atkinson [4], en la que el perfil define una clasificación estereotipos como medio de emular las extensiones del metamodelo. Al reproducir el metamodelo, el perfil conserva una estructura similar a esta.

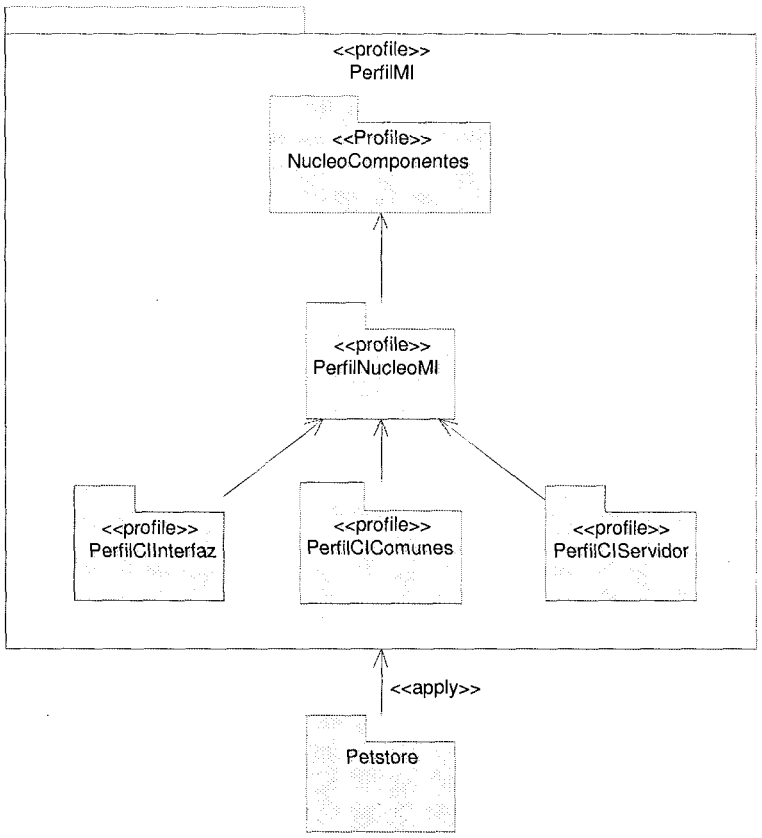

Figura. 45. Estructura de Paquetes del Perfil MI

La Figura. 45 muestra el perfil MI que está constituido por 5 paquetes estereotipados como perfil. Esta estructura jerárquica del perfil MI viene derivada del metamodelo de MI, debido a que se representan cada una de las metaclases como estereotipos conservándose así la misma estructura. Los paquetes que constituyen MI son los siguientes: (1) *NucleoComponentes,* es un perfil común para los dos modelos de componentes estructurados del MC y el MI, ya que en él se definen los estereotipos principales de estos modelos de componentes (Ver Figura. 35). Seguidamente, se define el paquete (2) *PerfilNucleoMI* donde se definen los estereotipos principales que representan a las metaclases nuevas del metamodelo MI (ver Figura. 42). Desde el *PerfilNucleoMI* se definen tres paquetes cada uno de los cuáles son perfiles dependientes de él: (3) *PerfilCIInterfaz* donde es definen los estereotipos de la jerarquía de componentes que viven en la capa de interfaz de usuario, (4) *PerfilCIComunes*, define los estereotipos de los componentes de MI que son comunes a cualquier capa y (5) *PerfilCIServidor*, establece los estereotipos de componentes de MI de la capa servidora. Por último, se puede apreciar como el paquete *Petstore* que

representa a una aplicación Web cualquiera, tiene una relación de dependencia <<apply>> sobre el perfil perfilMI, esto representa que dicha aplicación usa el perfil para su representación.

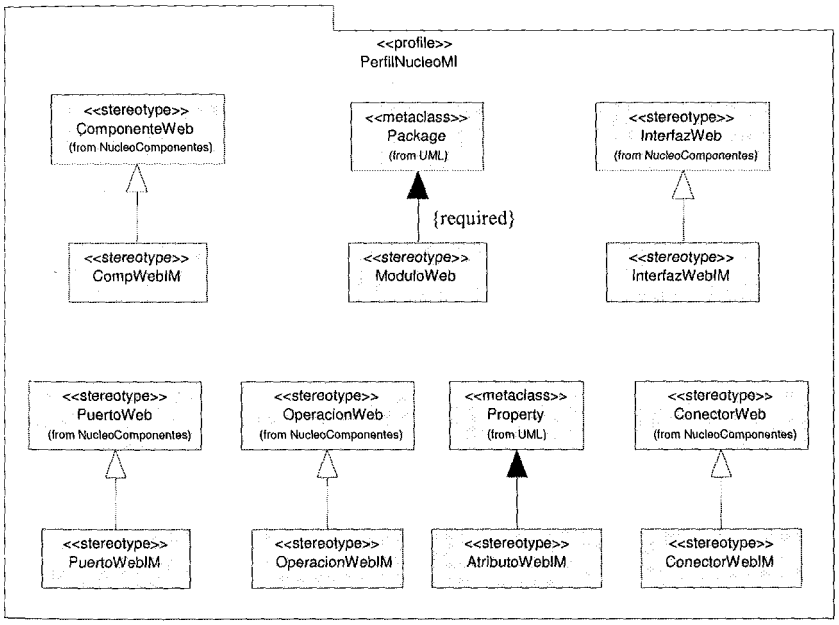

Figura. 46. Paquete PerfilNucleoMI

La descripción de *perfilMI* continúa profundizando en cada uno de los perfiles que la constituyen. La Figura. 46 describe los estereotipos del perfil *PerfilNucleoMI*. Para cada metaclase nueva definida en el metamodelo MI (ver Figura. 42) existe un estereotipo en este paquete, de los cuáles, algunos heredan de estereotipos definidos previamente o extienden directamente de las metaclases del metamodelo de UML. Este último caso lo cumplen dos estereotipos: (1) ModuloWeb, que extiende de *CompositeStructures::StructuredClasses::Class*, del cual va a obtener las propiedades de composición y de comunicación como los puertos e interfaces, y (2) *AtributoWebIM* que extiende de la metaclase *Classes::Kernel:Property* de forma requerida, es decir, obligatoriamente se utilizará este estereotipo en lugar de *Property* siempre que se aplique el perfil.

Entre los estereotipos que heredan de otros ya existentes, el más importante es el elemento *CompWebIM*. Este estereotipo representa al componente de integración y hereda de otro estereotipo *ComponenteWeb* de tipo genérico, del cual obtendrá sus propiedades y la jerarquía de tipos. El resto de estereotipos *PuertoWebIM, ConectorWebIM, InterfazWebIM y OperacionWebIM* extienden los estereotipos que se han definido en el perfil *NucleoComponentes* introduciendo las restricciones específicas del MI.

Para abordar la definición de cada uno de los estereotipos que corresponden a la topología de componentes Web dentro del perfil MI, se ha de considerar que dicha topología es la misma que la que se definen para el MC. El mecanismo para su definición es que cada uno de los tipos de componentes definidos en el MI, hereda por un lado de *CompWebIM* y por otro, del estereotipo abstracto del tipo. Para entenderlo mejor, la Figura. 47 muestra un ejemplo en el que se define el estereotipo *EntidadWebIM*, este estereotipo hereda los valores definidos del tipo *EntidadWeb* y las propiedades del componente *CompWebIM*.

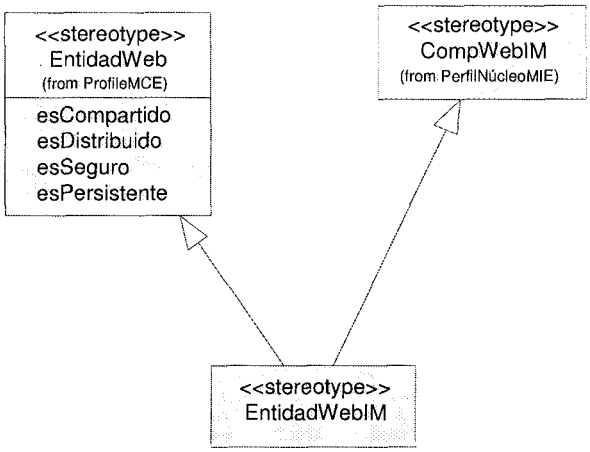

Figura. 47. Definición del Estereotipo EntidadWebIM

Si se desea consultar la descripción completa del perfil MI de cada uno de los estereotipos se encuentra en el Apéndice IV.

7.2.5 Casos de Estudio

Siguiendo los casos de estudio que se han presentado en los modelos de análisis arquitectónico, en el modelo de diseño MI se van a tratar las mismas aplicaciones Web abordadas: la agencia de Viajes y la tienda virtual Petstore. Esto permite apreciar como se ha realizado la integración de los aspectos funcionales y arquitectónicos en el MI.

7.2.5.1 Modelo de Integración de la Agencia de Viajes Virtual

Para abordar la especificación del MI de la agencia de viajes virtual es preciso basarse en el análisis realizado previamente. El análisis arquitectónico a partir de los requisitos no funcionales y de accesibilidad ha sido descrito en capítulos anteriores: (1) en la sección 5.7.1 donde se especifica la distribución en subsistemas en el MS, y (2) en la sección 6.8.1 donde se propone una de configuración de componentes en el MC.

Al mismo tiempo ha sido realizado el análisis funcional, y para ello han sido especificados los modelos funcionales de dominio y de navegación que pueden consultarse en el artículo [75].

Una vez especificado el análisis se ejecuta la transformación T1 que permitirá obtener el MI que se muestra a continuación. La transformación T1 es explicada con profundidad en el capítulo 8.

Este modelo está constituido por cuatro ModulosWeb: *Presentación, ControlUsuario, LogicaNegocio* y *Persistencia.* Estos elementos ModuloWeb se han obtenido a partir de la distribución de cuatro capas especificado en el MS (sección 5.4.8) y al aplicar la transformación T1. Cada *ModuloWeb* contiene un conjunto de *CompWebIM* con sus propiedades y de elementos *ConectorIM* localizados en una capa específica (p.e. el ModuloWeb ControlUsuario tiene asociados una serie de tipos de CompWebIM como son el tipo *PaginaServidora, ControladorWeb, Vista*, etc.)

Realizando una descripción de arriba-abajo, el primer ModuloWeb que se aborda es el de *Presentación* que representa a una interfaz de usuario delgada que contiene únicamente aspectos de presentación y tiene ubicada la funcionalidad en el *ControlUsuario.* Este cliente delgado tiene un aspecto gráfico sencillo, pero tiene un coste de distribución bajo y un buen nivel de reuso. En este Subsistema se ubican los componentes específicos como *AgenteUsuario, PaginaEstática, PaginaFuncional, AgenteUsuario*, y además cualquiera de los componentes de tipo común (ver sección 6.4.6.4).

Sin embargo, basado en el MC especificado para la agencia de viajes solamente tres componentes *AgenteUsuario* son candidatos para este módulo. La Figura. 48 muestra el contenido del ModuloWeb *Presentación.*

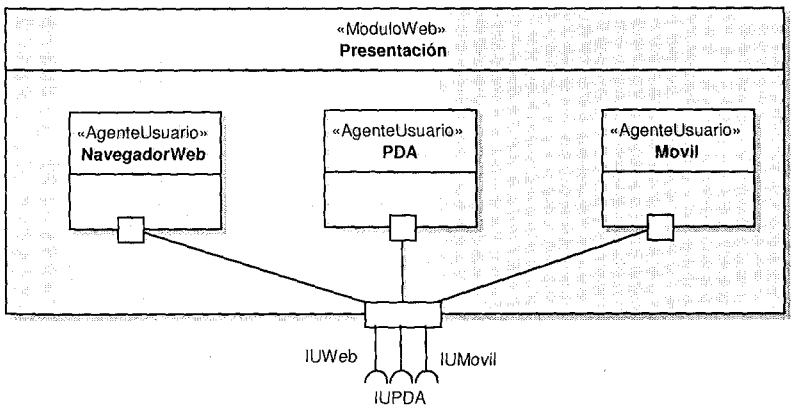

Figura. 48. ModuloWeb Presentación de la Agencia de Viajes

La Figura. 48 muestra como el ModuloWeb presentación contiene tres tipos de AgenteUsuario: *NavegadorWeb, PDA* y *Móvil* los cuáles fueron definidos en el MC (ver Figura. 39). Cada AgenteUsuario está conectado con el exterior del ModuloWeb por medio un PuertoWebIM que es el único punto de interacción o comunicación entre el ModuloWeb Presentación y el resto de los módulos. Este puerto contiene tres interfaces requeridos cada uno de los cuáles corresponde a cada AgenteUsuario:

IUWeb que corresponde al *NavegadorWeb*, *IUPDA* que corresponde a la *PDA* y *IUMovil* que corresponde al Móvil. Estos tres interfaces son ofertados por el ModuloWeb *ControlUsuario* el cual recibirá las peticiones de cada uno de los agentes.

Siguiendo la descripción descendente, el siguiente ModuloWeb es *ControlUsuario*. Este ModuloWeb recibe y controla las peticiones de la *Presentación*, establece la navegación y redirecciona las peticiones al ModuloWeb *LogicaNegocio*.

La Figura. 49 muestra un simplificado ModuloWeb ControlUsuario de la agencia de viajes. En la parte superior del módulo se aprecian los tres interfaces que provee la funcionalidad a la presentación y que son ofertados por los componentes de tipo PaginaServidor.

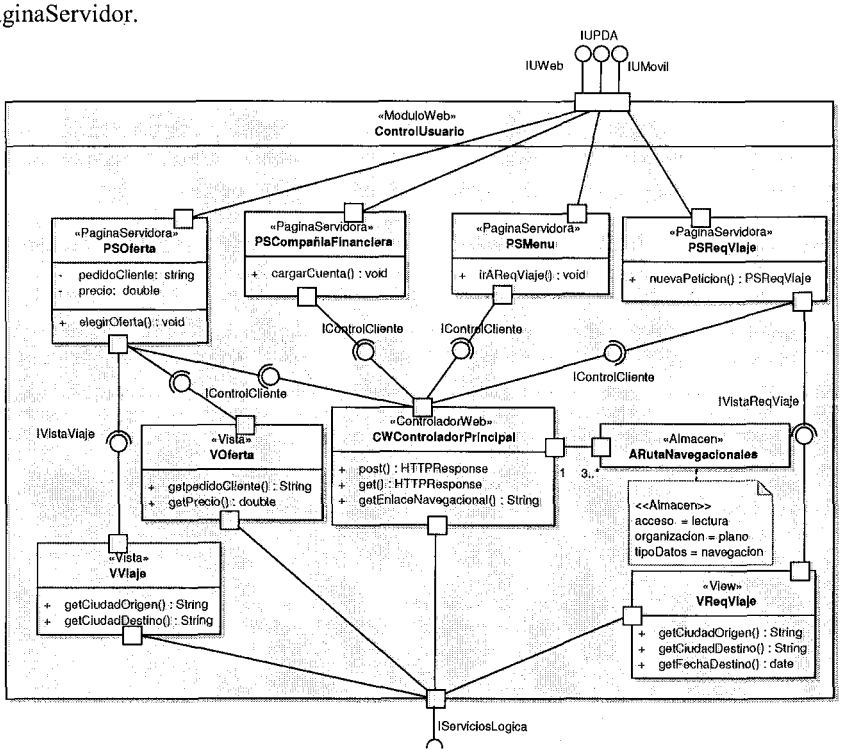

Figura. 49. ModuloWeb ControlUsuario de la Agencia de Viajes

Cada uno de los componentes PaginaServidora son obtenidos a partir de una o más clases navegacionales (p.e. Oferta, CompañiaFinanciera, Menu, etc.). Además, cada PaginaServidora tiene una interfaz requerida para acceder al componente Vista (p.e. la PaginaServidora *PSReqViaje* tiene una InterfazWebIM llamada *IVistaReqViaje* para acceder al componente Vista *VReqViaje*) y otra interfaz requerida llamada *IControlCliente* para acceder al componente ControladorWeb llamado *CWControladorPrincipal*. El componente ControladorPrincipal recibe las peticiones a través de la interfaz *IControlCliente* e invoca por un lado al componente Almacén *ARutasNavegacionales* para averiguar la próxima navegación y a la interfaz

IServiciosLogica para invocar a la funcionalidad de lógica de negocio. Por otra parte, se han generado un componente de tipo Vista por cada clase del modelo del dominio (p.e. *VViaje* a partir de la clase Viaje). Cada componente vista tiene una conexión con la lógica de negocio a través de la interfaz *IServiciosLogica*.

Finalmente, se aborda las capas inferiores, formadas por los módulos *LogicaNegocio* y *Persistencia*. El ModuloWeb *LogicaNegocio* tiene como tarea resolver las reglas de negocio establecidas por el modelo de dominio. Por otro lado, el ModuloWeb *Persistencia* provee el mecanismo de almacenamiento para la aplicación Web.

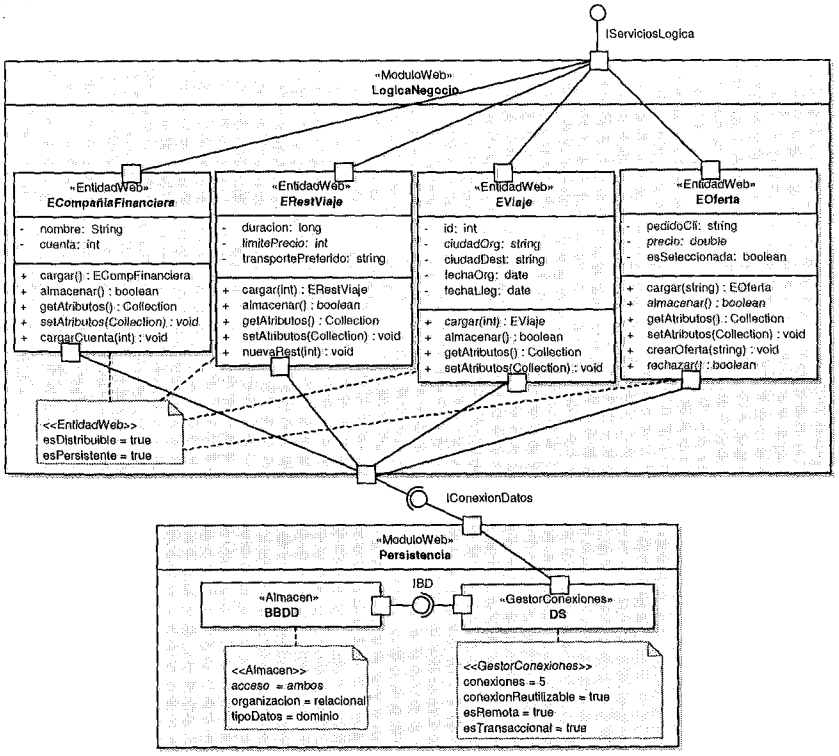

Figura. 50. ModuloWeb LogicaNegocio y Persistencia de la Agencia de Viajes

La Figura. 50 muestra la parte del MI de la agencia de Viajes que representa una simplificación del ModuloWeb *LogicaNegocio* y el ModuloWeb *Persistencia*, cada uno de los cuáles contienen un conjunto de componentes poblados a partir de la transformación T1. En parte superior, el ModuloWeb *LogicaNegocio* contiene la InterfazWebIM *IServiciosLogica* que obtiene las peticiones que proceden del ModuloWeb *ControlUsuario*. Esta interfaz garantiza el acceso a los diferentes componentes EntidadWeb. *LogicaNegocio* contiene un EntidadWeb por cada una de las clases definidas en el modelo de dominio. Cada EntidadWeb además de contener los atributos y operaciones procedentes del dominio, posee las operaciones de control para almacenar y recuperar los datos, y para comunicarse con el modulo de

persistencia. Respecto a los atributos que fueron definidos para el componente EntidadWeb de MC, únicamente han quedado aquellos que son relevantes para la transformación T2, es decir, los atributos que indican que EntidadWeb son distribuibles y persistentes. El resto de los atributos fueron utilizados para obtener los atributos y las operaciones en la transformación T1.

Por último, el ModuloWeb *Persistencia* está compuesto por dos componentes. El GestorConexiones llamado *DS* que provee una o más conexiones físicas para almacenar los datos, en este caso contiene 5 conexiones, de tipo remoto que admiten transacciones. El gestor está conectado a un componente Almacén llamado *BBDD* de organización relacional, que contiene los datos del dominio y permite un acceso de lectura y escritura.

7.2.5.2 Modelo de Integración de la Tienda Virtual Petstore

Para entender el MI de la tienda virtual conviene recordar los modelos de análisis especificados en los capítulos anteriores. El análisis arquitectónico a partir de los requisitos no funcionales ha sido descrito en: (1) en la sección 5.7.3 se especifica la distribución en subsistemas en el MS, y (2) en la sección 6.8.2 se propone una de configuración de componentes en el MC. Simultáneamente, se ha realizado el análisis funcional, y han sido especificados en (3) la sección 4.5.2.3 muestra el modelo de dominio y (4) en la sección 4.5.3.3 el modelo de navegación.

Este modelo está constituido por siete ModulosWeb: 2 módulos de Presentación, 2 módulos ControlUsuario, 2 módulos LogicaNegocio y 1 modulo de Persistencia. Estos ModulosWeb se han obtenido a partir de un proceso de distribución en subsistemas especificado en el MS, basado en la aplicación de los patrones de distribución. Además, Petstore posee una división en 2 vistas principales de administración y cliente que parten en dos a los subsistemas superiores.

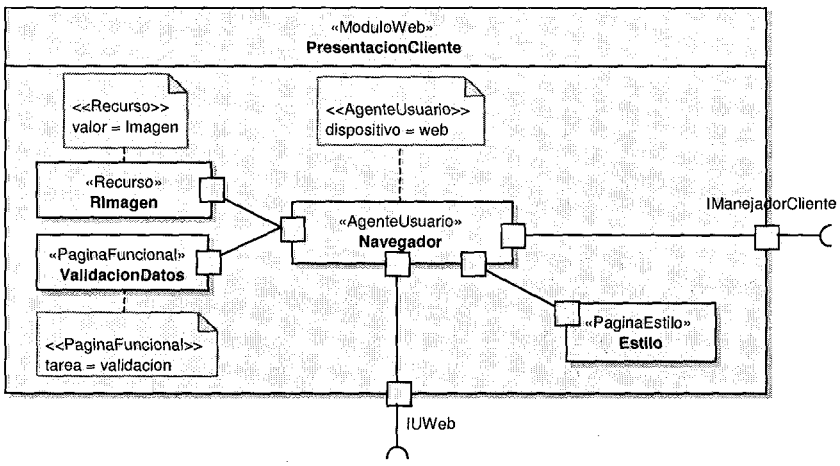

Figura. 51. ModuloWeb PresentaciónCliente de PetStore

A continuación, se describen los módulos que representan la vista de cliente del MI de Petstore, por ser la vista con la arquitectura más compleja. Realizando una

descripción de arriba-abajo, primero se aborda el ModuloWeb *PresentaciónCliente* (ver la Figura. 51) que representa a una interfaz de usuario delgada que contiene únicamente componentes de presentación. El componente central como se puede apreciar es el AgenteUsuario *Navegador* que como indica su atributo, es un tipo de dispositivo proporciona una interfaz Web al cliente del sistema. Además, el componente *Navegador* está vinculado con imágenes proporcionadas por el Recurso *RImagen*, y obtiene su estilo mediante la PaginaEstilo *Estilo*. Por último, indicar que la interfaz valida los datos enviados utilizando la funcionalidad de la PaginaFuncional *ValidacionDatos*.

Este ModuloWeb *PresentaciónCliente* está conectado con el ModuloWeb *ControlUsuarioCliente* mediante dos interfaces, *IUWeb* que proporciona las paginas cliente al navegador e *IManejadorCliente* que recibe las peticiones HTTP de la interfaz.

El siguiente ModuloWeb es *ControlUsuarioCliente* (ver Figura. 52). Este ModuloWeb recibe y controla las peticiones de la *Presentación* estableciendo la navegación y redireccionando las peticiones al ModuloWeb *LogicaNegocio*.

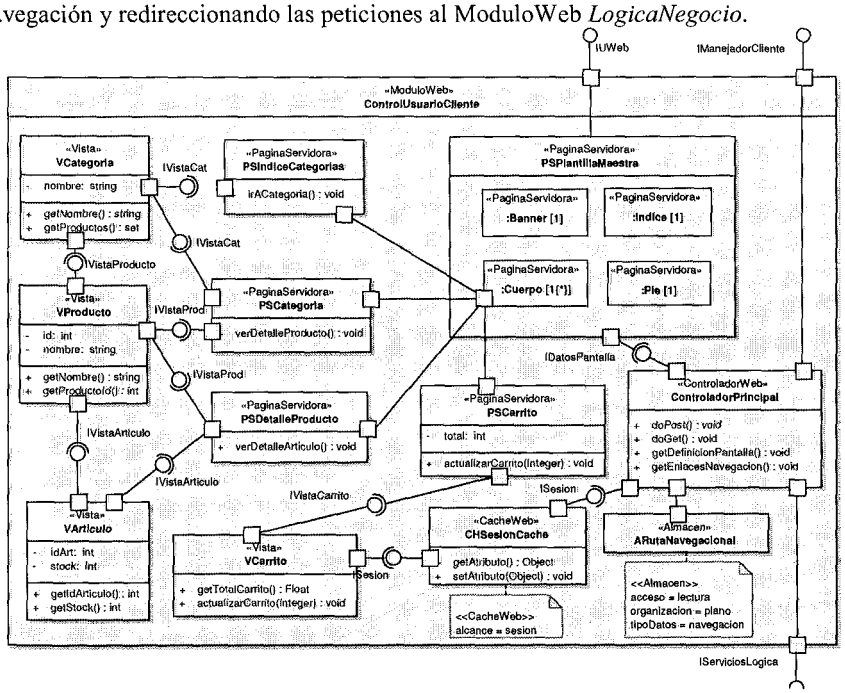

Figura. 52. ModuloWeb ControlUsuarioCliente de Petstore

El modulo *ControlUsuarioCliente* se ha representado simplificado para que pueda ser fácilmente entendible. En la parte superior del módulo se ofertan dos interfaces que corresponden a los ya mencionados *IUWeb* e *IManejadorCliente*. *IUWeb* es una interfaz que devuelve las páginas, y para ello se conecta con la PaginaServidora *PSPlantillaMaestra* que configura las páginas de la aplicación Petstore desde la definición de una plantilla formada por 4 elementos ParteWeb de tipo

PaginasServidora llamados *Banner*, *Indice*, *Pie* y Cuerpo. Cada una de las partes son instancias de PaginaServidora que son intercambiables y configurarán la página final. La ParteWeb Cuerpo contiene una multiplicidad de 1 o muchos respecto a la plantilla, debido a que en dicha parte de la plantilla se van a representar las diferentes PaginaServidora obtenidas del modelo de navegación con el contenido de la aplicación. Así está relacionada con las paginas *PSIndiceCategorias*, *PSCategoria*, *PSDetalleProducto* y *PSCarrito*. Por su parte, cada PaginaServidora está relacionada a su vez, con una o más componentes de tipo Vista que contienen la información que se muestra en dichas páginas. Por ejemplo *PSCategoria* está conectada a la vista *VCategoria* y a *VProducto*.

Por otro lado, las peticiones obtenidas desde la interfaz mediante *IManejadorCliente* son enviadas al ControladorWeb *ControladorPrincipal*. El componente gestiona las peticiones y establece la navegación a partir del almacén *ARutaNavegacional*, y la definición de las páginas accediendo a la interfaz *IDatosConsulta* de la *PSPlantillaMaestra*. El ControladorPrincipal está conectado a su vez con una CacheWeb *CHSesionCache* que contiene la información almacenada temporalmente para una sesión, en este caso la información es el contenido del carrito de la compra representado por la clase *VCarrito*. Por último, el ControladorPrincipal se conecta con la lógica de negocio por medio de la interfaz *IServiciosLogica*.

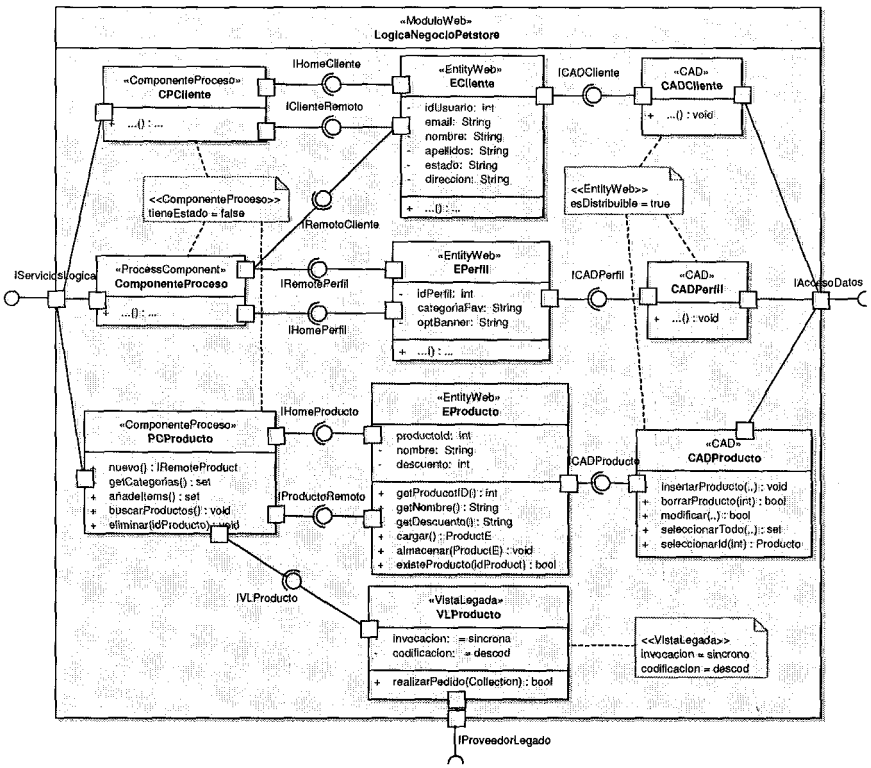

Figura. 53. ModuloWeb LogicaNegocio de Petstore

La Figura. 53 muestra la parte del MI de Petstore que representa una simplificación del ModuloWeb *LogicaNegocioPetstore* que contiene los componentes que establecen las reglas de negocio de la aplicación y se comunican con la base de datos. La entrada a este modulo se realiza a través de la InterfazWebIM *IServiciosLogica* que recibe las peticiones procedentes del control de usuario. Los servicios son enviados a cada uno de los correspondientes ComponenteProceso dependiendo de la operación invocada. En la Figura. 53 se han representado los componentes que proceden de las clases definidas en el modelo de dominio (*Cliente*, *Perfil* y *Producto*) (ver Figura. 9). Solamente en los componentes de la clase *Producto* es donde se ha introducido toda su información. Los dos primeros componentes asociados a cada clase vienen determinados por el patrón Façade definido en el capítulo 6 (ver Figura. 41). El patrón Façade define un ComponenteProceso sin estado, encargado de recibir las peticiones del cliente y de invocar al componente EntidadWeb. Los datos y el estado de cada entidad de dominio son contenidos por el componente EntidadWeb. El ComponenteProceso contiene los métodos de negocio procedentes del modelo de dominio y se encarga de crear a los componentes EntidadWeb utilizando su interfaz de factoría (p.e. *CPProducto* invoca a la interfaz *IHomeProducto* para crear un componente *EProducto*) y por último invocar a la operaciones de negocio de los componentes EntidadWeb a través de su interfaz remoto (p.e *IProductoRemoto*). En este ejemplo, se puede apreciar como *CPProducto* invoca además a un componente de tipo VistaLegada llamado *VLProducto* que se encarga de invocar a un sistema legado para que realice la operación *realizarPedido*, encargado de solicitar a los proveedores los productos que necesita la tienda virtual.

Por otro lado, cada componente EntidadWeb se comunica con los componentes CAD (Componente de Acceso a Datos) que contienen los métodos dependientes de la base de datos, necesarios para crear, modificar, borrar y consultar una instancia de cada una de las clases de dominio.

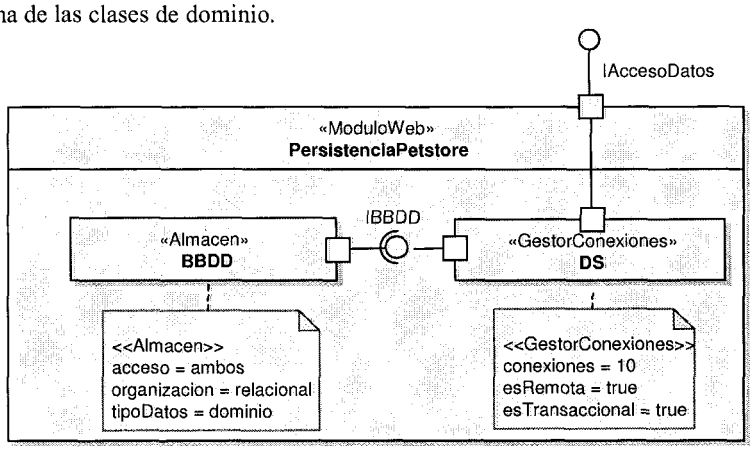

Figura. 54. ModuloWeb PersistenciaPetstore

Para terminar la descripción del MI de Petstore, se aborda el ModuloWeb *PersistenciaPetstore* que contiene los componentes necesarios para dotar de persistencia a la aplicación Web. Este modulo recibe las peticiones a través del PuertoWebIM *IAccesoDatos* y son enviadas al componente GestorConexiones llamado *DS*, este componente está configurado con 10 conexiones que se establecen de forma remota y transaccional. Dichas conexiones se establecen con el componente de tipo Almacén llamado *BBDD*, el cual contiene la información de Dominio de la aplicación, su organización es mediante un sistema Relacional y permite un acceso tanto de escritura como lectura.

7.3 Conclusiones del Capítulo

En este capítulo se ha presentado la vista de integración definida en la fase de diseño del proceso propuesto por WebSA. Esta vista esta constituida por el modelo de arquitectura MI, y se obtiene a partir de la aplicación de la transformación T1 desde los modelos de la fase de análisis. MI pretende cubrir el gap existente entre los modelos de diseño definidos por aproximaciones como OO-H [19], WebML [20] y la implementación obtenida finalmente de las aplicaciones Web. El MI presenta un modelado de la aplicación Web cercano a la implementación final, en la que se representan tanto los aspectos funcionales como los aspectos de arquitectura, pero que abstrae de los aspectos puramente dependientes de la plataforma. Permitiendo que se puedan definir desde el MI diferentes transformaciones a las plataformas de implementación como J2EE, .NET, etc.

Para definir el MI, se han presentado los elementos en un metamodelo MOF, que recoge tanto sus elementos y las relaciones entre ellos. A continuación, se ha definido su notación y para ello se ha definido un perfil que representa sus elementos mediante el modelo de estructurado de componentes propuesto por UML 2.0. Seguidamente, se establece el método por el cual se debe actuar cuando se obtenga el MI que no sea adecuado. Y por último, se ha mostrado los dos casos de estudio que presenta la tesis, la agencia de viajes y la aplicación de comercio electrónico Petstore.

El siguiente capítulo presenta las transformaciones T1 y T2, las cuáles va a unir cada una de las fases del proceso de desarrollo de WebSA.

Volver al índice/Tornar a l'índex

PARTE III

Modelado e Implementación de las Transformaciones

Esta parte se centra en la representación de las transformaciones de modelo a modelo y de modelo a código definidas dentro de WebSA. En el capítulo 8 se define un lenguaje de transformaciones con notación gráfica que se utiliza para representar las transformaciones modelo a modelo que son definidas dentro del proceso de desarrollo de WebSA. Seguidamente, las transformaciones modelo a texto se definen en un lenguaje de plantillas para su representación. En el capítulo 9 se muestra la herramienta WebTE que implementa un motor para la introducción de los modelos y las transformaciones definidas en WebSA.

CAPÍTULO 8

Modelado de Transformaciones en WebSA

"Nada se destruye, nada se crea, todo se transforma"

[Antoine Lavosier]

8.1 Motivación

La principal característica de MDE, y a su vez, de su estándar MDA (Model Driven Architecture) [89], es que el desarrollo de aplicaciones es visto como un proceso refinamiento que transforma modelos en otros modelos hasta que puedan ser ejecutados en una determinada plataforma. Por lo tanto, en MDA el elemento realmente novedoso es el concepto de transformación de modelos, elemento que además posee el importante papel de unificar y dotar de trazabilidad al proceso de refinamiento.

Las transformaciones permiten relacionar cada uno de los modelos especificados, indicando cuáles son los pasos necesarios para evolucionar un modelo o convertir un modelo en otro modelo o en código. Esto permite que las aproximaciones dirigidas por modelos ofrezcan, de forma parcial o completa, la posibilidad de realizar pasos de conversión automática, que serán realizados por motores o transformadores que a partir de dichos modelos de transformaciones permitirán obtener el modelo o código final. Mediante el uso de las transformaciones se obtienen las siguientes ventajas: (1) una aceleración del proceso de desarrollo al realizarse determinadas tareas automáticamente, (2) una reducción del número de errores al no existir intervención humana en la ejecución de las transformaciones y (3) una trazabilidad bi-direccional de los modelos, lo que permite que se pueda transformar los modelos en un sentido y en el contrario.

En este sentido, la OMG ha propuesto un lenguaje de naturaleza híbrida declarativa/imperativa para la definición de transformaciones modelo a modelo denominado Query/Views/Transformations (QVT) [91]. Sin embargo, la

especificación QVT es muy reciente y las primeras herramientas para su implementación como ATL GMT Proyect [5], Together 2006 [117], UMLX [121] no han proporcionado todavía las soluciones esperadas.

Este trabajo presenta una nueva aproximación llamada UPT (UML Para Transformaciones) [76] que propone la definición de lenguaje de tranformaciones basado en UML para representar las transformaciones declarativas basadas en QVT Relation. UPT viene acompañado de un motor de tranformaciones (presentado en el capítulo 9) que ha permitido la materialización de las transformaciones de WebSA. Así, los modeladores pueden usar cualquier herramienta UML para definir sus transformaciones, utilizar el motor de transformaciones e intercambiar estas transformaciones con otras herramientas.

Como ya ha sido indicado, el proceso de desarrollo de WebSA especifica dos transformaciones principales: (1) T1 es una transformación que se encarga de hacer converger los modelos funcionales propuestos por diferentes aproximaciones dentro de la ingeniería Web y los modelos de arquitectura definidos por WebSA. Dicha transformación será especificada mediante la aproximación UPT. Mientras que la transformación T2 (2) convierte los modelos de integración definidos en la fase de diseño en la implementación de la plataforma destino. T2 es una transformación de modelo a código y necesita de un mecanismo de obtención de texto. Para ello, se basa en otro estándar proporcionado por la OMG llamado MOFScript [94]. En este sentido MOFScript aporta un lenguaje que permite consultar un metamodelo MOF y generar texto mediante un conjunto de reglas de transformación que extienden de QVT. Actualmente, existe una única implementación, pero que no está basada en MOF, sino en EMF que es un framework de la herramienta Eclipse. Es importante destacar, que aunque para representar T2 se opta por la representación mediante MOFScript, su implementación final en la herramienta se especifica en Velocity [123].

Para resumir, WebSA se sirve de las transformaciones principalmente para:

- Acelerar el desarrollo de las aplicaciones Web desarrolladas.
- Integrar dos aspectos ortogonales de las aplicaciones Web como son los modelos funcionales y los modelos arquitectónicos en la fase de diseño.
- Integrar los aspectos dependientes de la plataforma de implementación en un modelo específico de la plataforma.
- Complementar diferentes aproximaciones de Ingeniería Web reconocidas como OOH [19], UWE [59], etc.
- Expresar aquellos casos especiales dentro de una determinada aplicación Web.

Este capítulo presenta la descripción de las transformaciones propuestas en el proceso de WebSA. Para ello, en la sección 8.2 se realiza una introducción a QVT. A continuación, en la sección 8.3 se presenta el lenguaje UPT. UPT se especifica a través de sus elementos, metamodelo y el perfil UML definido. Seguidamente, en la sección 8.4 se aborda la descripción de la transformación T1 constituida a su vez por 5 transformaciones. Una vez, realizada dicha descripción, en la sección 8.5 comienza la transformación T2 mediante una introducción a MOFScript, y la descripción de la transformación T2, mediante el flujo principal y las diferentes reglas de transformación.

8.2 QVT: El Lenguaje Estándar para Transformaciones entre Modelos

La OMG reconoció que las transformaciones de modelos son de crucial importancia para el éxito de MDA y el 24 de abril de 2002 distribuyó la primera propuesta de especificación llamada Query/Views/Transformations (QVT) RFP [90]. Este documento establecía cuáles eran los requisitos deseables que debería tener el lenguaje de transformaciones basado en metamodelos. Fueron presentadas 8 propuestas diferentes cada una de las cuáles provenía de diferentes empresas y universidades, proporcionando características y propiedades que fueron estudiadas por trabajos como [85], en las cuáles se destacaban las bondades y defectos de cada una de ellas. Sin embargo, siguiendo con el proceso de la estandarización, la OMG propuso que se reunieran los participantes de las diferentes propuestas y unificaran su conocimiento en una única propuesta. Dicha propuesta ha tenido diferentes versiones parciales, y su versión definitiva se ha presentado recientemente el 1 de noviembre de 2005 [91].

La especificación QVT es un lenguaje de transformaciones de naturaleza híbrida, es decir, declarativa e imperativa. Además, su parte declarativa está dividida en una arquitectura de 2 capas. Por un lado, un lenguaje llamado Relations destinado para el usuario o modelador que proporciona un emparejamiento mediante patrones de objetos complejos y plantillas de objetos. El lenguaje *Relations* puede ser expresado mediante una notación gráfica y textual. Por otro lado, un lenguaje llamado *Core* cuya función es ser utilizado internamente por las herramientas de transformaciones y que define las transformaciones usando una extensión mínima de MOF y OCL. Por último, también comentar que posee una notación imperativa llamada *Operational Mappings*.

El lenguaje *Relations* es crucial para el éxito de QVT en la comunidad de desarrolladores de MDA ya que es el interfaz principal que ofrece a los modeladores de transformaciones. En este sentido, siguiendo la filosofía propuesta por la Ingeniería Dirigida por Modelos, donde el modelo es el artefacto más importante dentro del proceso de desarrollo y considerando que una transformación es otro modelo, su representación más conveniente es la notación gráfica. Sin embargo, QVT es muy reciente y todavía no ha sido desarrollada hasta la fecha ninguna herramienta que satisfaga completamente este aspecto. Entre otras razones porque el lenguaje grafico de QVT Relations proporciona una notación propietaria que hace necesario que los desarrolladores de herramientas la incorporen. De hecho, hasta hoy no hay una herramienta comercial que soporte la notación gráfica de QVT.

Por este motivo, en este trabajo se ha definido un lenguaje para definir transformaciones llamado UPT, el cual proporciona una notación gráfica declarativa sobre UML basado en un emparejamiento de patrones similar a QVT Relations.

8.3 UML Para Transformaciones (UPT)

WebSA, como aproximación MDA, establece la automatización de su proceso de desarrollo mediante el uso de las transformaciones. Entendiendo la automatización como la conversión de los modelos que representan al sistema hasta la implementación del sistema. Sin embargo, MDA deja abierto la elección del tipo de

notación utilizada para definir las transformaciones. A partir de ese punto, esta aproximación se ha basado en la misma idea que en la definición de los modelos de representación del sistema. Se entiende que un modelo estándar proporciona como ventaja principal que permite que sea comprensible por un mayor número de personas, se reduzca la curva de aprendizaje y disponga de un mayor número de herramientas para su representación.

Basándose en los beneficios que la propuesta QVT ofrece y principalmente la parte de relaciones (QVT Relations) para la representación de las reglas de transformación, WebSA había apostado inicialmente por este estándar para definir sus transformaciones como se muestra en el trabajo de Meliá & Gómez [75]. Sin embargo, debido a que no existe ninguna implementación que de soporte a QVT Relations completamente y a la necesidad de materializar las transformaciones de WebSA para refinar y mejorar el proceso, surge la necesidad de definir un lenguaje que si lo haga. Por un lado, se incorpora una nueva notación gráfica basada en el diagrama de clases de UML para la representación de las transformaciones, haciendo más sencilla su especificación. Por otro lado, se simplifica el metamodelo MOF de QVT-Relations, estableciendo los elementos necesarios para su especificación usando el diagrama de clases y añadiendo otros elementos que permiten la definición de la composición de las transformaciones.

El resultado es una solución para la definición de las transformaciones que se denomina UPT (UML para Transformaciones) que propone la representación de las transformaciones declarativas basadas en QVT Relations mediante el uso del diagrama de clases UML.

Las razones principales para la utilización de UML como lenguaje gráfico para la representación de las transformaciones son cuatro:

1. Existe un gran número de herramientas actualmente que permiten representar gráficamente modelos UML. Reduciéndose así el coste que supone definir nuevas herramientas para las transformaciones QVT.
2. UML permite el intercambio de los modelos de transformación entre diferentes herramientas y analistas mediante el estándar XMI.
3. UML permite la introducción de nuevos conceptos y semántica a QVT mediante el uso de diferentes modelos.
4. La utilización del diagrama de clases para representar las transformaciones de QVT-Relations, proporciona el modelo más usado por los analistas dentro de UML, como demuestran trabajos que estudian el uso de UML como el de Dobing & Parsons [26]. Esto otorga una menor curva de aprendizaje para los modeladores que quieran definir las transformaciones.

A continuación, se procede a la definición de los diferentes elementos que constituyen las transformaciones de UPT.

8.3.1 Los elementos de UPT

Para poder representar las transformaciones mediante el lenguaje UPT, se describen cuáles son los elementos que lo constituyen. Los elementos definidos para UPT están basados en un subconjunto de los elementos proporcionados por QVT Relations. Sin embargo, se han introducido modificaciones que mejoran la expresividad de su notación gráfica, reduciendo sus ambigüedades y basándose en el diagrama de clases.

Para realizar la descripción se realiza un recorrido de descendente desde los conceptos más genéricos a los más concretos.

8.3.1.1 Transformación

Una transformación especifica un conjunto de relaciones o reglas de transformación que deben llevarse a cabo o satisfacerse entre los elementos de un conjunto de modelos candidatos. Cada modelo candidato y los tipos de sus elementos están restringidos a un metamodelo. A este modelo candidato se le denomina modelo tipado. Así que la transformación contiene una referencia a los diferentes modelos tipados que participan en ella.

En QVT y UPT, las transformaciones se definen a nivel de metamodelo, es decir, se especifican relaciones entre las metaclases de un conjunto de metamodelos origen y las metaclases de un conjunto de metamodelos destino. Sin embargo, su aplicación es a nivel de modelo y sus definiciones permiten que sean aplicables a cualquier tipo de modelo que cumpla con las reglas de transformación definidas.

Para representar una transformación UPT se utiliza el elemento Class de UML. Se trata de una clase contenedora que encapsula el conjunto de reglas de transformación o de relaciones que deben llevarse a cabo.

8.3.1.2 Relación

La Relación es la unidad básica dentro de la especificación declarativa de UPT. Una relación indica cuáles son las restricciones que deben cumplirse por elementos de los modelos candidatos. Cada relación consta de dos o más dominios y de un par de restricciones conocidas como la cláusula *when* (o guarda) y la cláusula *where*. Cada dominio designa un modelo candidato y en el caso más simple un tipo a emparejar en ese modelo.

La relación en UPT está representada por un modelo de clases de UML, en el cual se define una clase estereotipada con el nombre <<Relacion>>. Dicha clase va a contener una serie de propiedades y relaciones asociadas a dicha relación (ver Figura. 55). Cada relación tiene las siguientes propiedades:

- **esRaíz**: representada como un atributo de tipo booleano. Indica cuando su valor es verdadero, que la relación es invocada al iniciarse la ejecución de la transformación, y no desde ningún where de otra relación. Cuando su valor es falso indica que la relación es invocada por la parte where de otra relación. Cada transformación debe tener una o más relaciones topes que marcarán el inicio de la ejecución de la misma.
- **When** (Cuando): se representa como una restricción de la clase <<Relation>>. La cláusula When específica una condición OCL que debe ser cumplida por las variables de la relación, para que esta sea llevada a cabo. Consiste en un predicado OCL, que debe ser ejecutado antes de la relación. Ver la Figura. 55.
- **Where** (Donde): representado mediante una restricción de la clase <<Relation>>. La cláusula where específica una condición que debe ser satisfecha por todos los elementos que han participado en la relación, y puede restringir cualquiera de las variables de la relación. Consiste en un predicado OCL, formado normalmente por un conjunto de llamadas a

otras relaciones. La notación para representar las llamadas consiste en un conjunto de llamadas separadas por el símbolo ';'. El where es ejecutado una vez se ha cumplido la relación.

En la siguiente figura se aprecia como se asocia a la clase <<Relation>> la parte when y where como restricciones (Constraints).

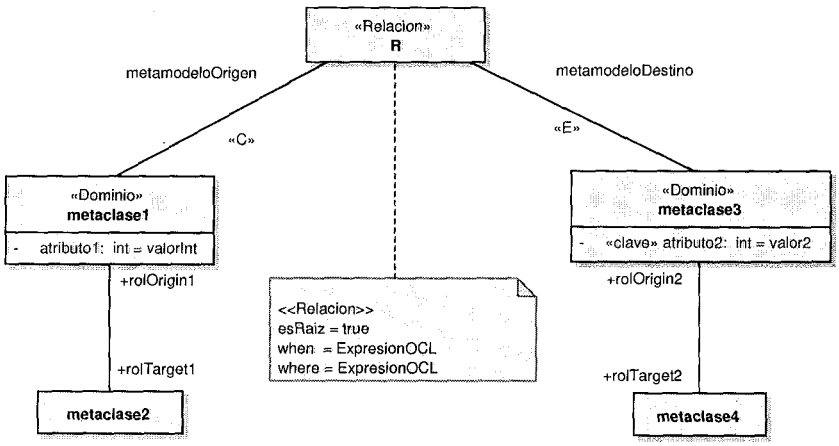

Figura. 55. La Notación de una Relación UPT

Además de las propiedades, cada clase <<Relacion>> contiene una serie de relaciones de asociación con los diferentes dominios que participan en ella.

8.3.1.3 Dominio

Un dominio especifica el conjunto de elementos de un metamodelo que son de interés para una relación. Un dominio dentro del perfil UPT, vendrá representado por una relación de asociación entre la clase Relación y la clase Dominio estereotipada con la etiqueta <<Dominio>> (ver la Figura. 55).

Cada relación de asociación entre la clase relación y la clase dominio contiene las siguientes propiedades asociadas a los dominios:

- **ModeloTipado**: se representa mediante el nombre de la relación de asociación. Un modelo tipado indica la parte del metamodelo que participa en la relación. En la Figura. 55 se puede apreciar como el modelo tipado se especifica por medio del nombre del metamodelo, (p.e *metamodeloOrigen o metamodeloDestino)* en el cual se navega desde fuera hacia dentro, hasta el paquete concreto del metamodelo en el cual se requiere los elementos del dominio.
- **Checkonly o Enforceable**: esta propiedad se representa mediante el estereotipo de la relación de asociación. Puede tener los valores <<C>> si el dominio es checkonly (en castellano "solo comprobación") y <<E>> si el dominio es enforceable (en castellano "hacer cumplir"). Si es un dominio es checkonly, indica que se comprueba que los elementos definidos en el dominio se encuentran de la misma manera en el metamodelo referenciado.

Si al comprobarlo no coinciden la relación falla. Sin embargo, si el dominio tiene la etiqueta <<E>> (enforceable) indica que si los elementos del dominio no se encuentran igual en el metamodelo referenciado, deben modificarse para conseguirlo. Para ello, la relación deberá crear elementos, borrar elementos o modificar elementos en el metamodelo referenciado. Normalmente este dominio se corresponde al metamodelo destino, mientras que los dominios con la etiqueta <<C>> se corresponden con los metamodelos origen.

- **VariableRaiz**: en UPT el dominio también es representado mediante una clase estereotipada con <<Dominio>>, en lo que se denomina como la clase RelacionDominio. El nombre dicha clase se denomina variableRaiz, y representa el elemento del modelo que marca el punto de entrada sobre el modelo tipado. En la notación UPT de la Figura. 55 metaclase1 y metaclase3 son variableRaiz.

8.3.1.4 PatrónDominio

Cada clase Dominio tiene asociado un PatrónDominio. En UPT un PatronDominio especifica una plantilla de clases en forma grafo arbitrariamente complejo, en la cual las clases representan instancias de las diferentes metaclases del metamodelo referenciado por el dominio. El PatronDominio debe ser localizado (Dominio checkeable) o creado (Dominio enforceable) para satisfacer la Relación. P.e. un PatronDominio comprueba que existe una instancia de la metaclase Class contenida en una instancia de la metaclase Package dentro del metamodelo de UML. Dentro de un PatrónDominio se encuentran los siguientes elementos:

- **PatrónClase**: referencia a una instancia de una metaclase. En UPT se representa mediante una clase. Si se repasa la arquitectura de capas de metamodelado de la OMG [88], la instancia de una metaclase es una clase. Por ello, la clase es su representación más apropiada. Esto no ocurre en QVT, que utiliza un objeto o colecciones de objetos, saltando 2 niveles la arquitectura de metamodelos. Un PatrónClase puede contener a su vez un conjunto de elementos PatronPropiedad. La Figura. 55 muestra cuatro elementos PatronClase cada uno corresponde a una metaclase diferente.
- **PatrónPropiedad**: especifican restricciones en los valores que las instancias PatronClase pueden tomar. Existen dos tipos, (1) relaciones de asociación entre diferentes PatronClase, (2) propiedades o atributos pertenecientes a la clase definida por un elemento PatrónClase.
    - **Atributo:** se especifica de la misma forma que un atributo de una clase UML. Se compone del nombre, un tipo y el valor que toma el atributo. El valor puede ser una variable si se especifica sin comillas, lo que le indica a la relación que este atributo puede tomar cualquier valor. Si el valor está entrecomillado indica que es un literal, restringiendo el valor del atributo al literal. En UPT se añade el tipo, respecto a la especificación de QVT, esto le permite restringir el rango de valores que toma el atributo. Por último, indicar que uno o más atributos de una clase pueden estar estereotipados como <<clave>>, esto permite indicar que los

atributos identifican de forma unívoca a cada instancia de la metaclase referenciada. En el caso de que el atributo clave, pertenezca a un PatronClase contenido en una composición (parte), su identidad será única para cada instancia del PatronClase compuesto o todo. En la Figura. 55 el atributo2 es clave y permite identificar a un objeto de la metaclase3.

  - **Rol:** aparece cuando el PatronPropiedad es el extremo de una asociación. En cuyo caso, el tipo de PatronPropiedad será el tipo del PatronClase que tenga la asociación en el extremo. Posee la **Cardinalidad:** especificado por dos números naturales que indican el número menor y mayor de instancias que contiene el rol (1, 0...1, 0...n, 1...n. etc.). El hecho de especificar una cardinalidad menor igual a cero en el rol de un PatronClase indica que se trata de una parte opcional del patrón especificado por la Relación. Mientras que cuando se pone una cardinalidad mayor igual a 1 se indica la obligatoriedad del rol. Esto introduce una semántica más rica que la representación utilizada mediante el diagrama de objetos de QVT, donde las relaciones entre instancias siempre son obligatorias. Además, el rol tiene la propiedad de **esComposición,** indicando cuando el PatronPropiedad está contenido por composición en otro PatronClase, es decir, es responsable de la existencia (interfiere en su identidad) y el almacenamiento de este.

- **PatronAsociación:** establece una relación entre las instancias de dos PatronClase. Permite reproducir la relación de asociación que se establece en el metamodelo referenciado por el dominio entre las instancias de dos de sus metaclases. Pero con dos semánticas distintas, (1) si el dominio es checkonly se comprueba que las instancias cumplen dicha relación, (2) si es enforceable se crea, borra o modifica las instancias necesarias para que se cumpla la asociación.

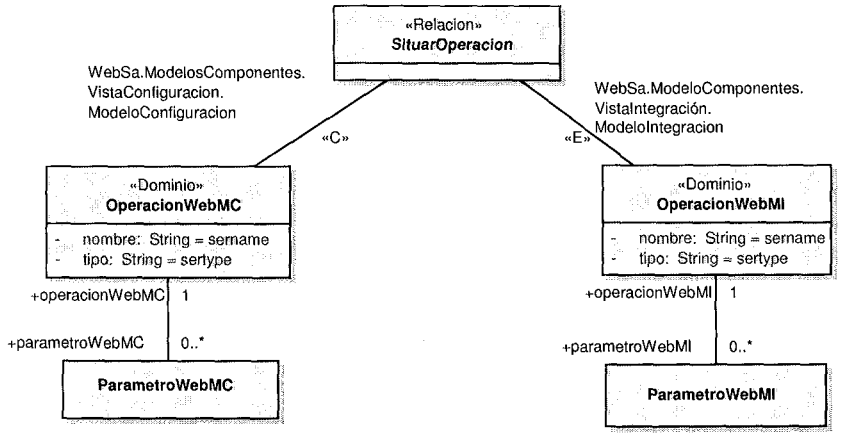

Figura. 56. Relación SituarOperacion de la Transformación T1 de WebSA

La Figura. 56 muestra la relación *SituarOperacion* definida dentro de la transformación T1 del proceso de WebSA. Gracias a la sencillez otorgada por la notación gráfica de UPT, puede verse a simple vista como la relación consulta en el dominio en el checkonly a un elemento *OperacionWebMC* con un conjunto de *ParametroWebMC* y obliga la existencia en el dominio enforceable (creará, modificará o borrará) de un *OperacionWebMI* que obtiene su nombre y su tipo del *OperacionWebMC*. Además obtiene el conjunto de sus *ParametroWebMI* a partir de cada *ParametroWebMC*.

A pesar de su sencillez, la relación contiene a la mayoría de los elementos que se han definido anteriormente. Puede apreciarse como los dominios estereotipados como <<Dominio>> corresponden a dos elementos PatronClase, llamados *OperacionWebMC* y *OperacionWebMI* respectivamente. Ambos elementos comparten los mismos PatronPropiedad, un atributo llamado *nombre* de tipo String y valor *sername*, y un atributo llamado *tipo* de tipo *String* y valor *sertype*. Estos valores al ser especificado sin comillas indican que se tratan de variables, a las cuáles puede aplicarse cualquier valor dentro del rango del tipo String. Además cada PatronClase tiene una relación de asociación, por un lado, *OperacionWebMC* tiene una relación de asociación con el PatronClase *ParametroWebMC*, en la cual se indica con la cardinalidad que por cada *OperacionWebMC* existen cero o muchos *ParametroWebMC*, y por cada *ParametroWebMC* existe al menos una instancia de *OperacionWebMC*. Por otro lado, se tiene una relación de asociación entre un *OperacionWebMI* y un PatronClase llamado *ParametroWebMI*, con la misma cardinalidad que la anterior relación. Por último, indicar que los roles especificados en las relaciones de asociación son utilizados como variables a la hora de invocar a un PatronClase dentro de de las restricciones OCL When y Where.

8.3.2 Metamodelo

En el metamodelo UPT se formalizan los elementos y las posibles relaciones que permiten representar gráficamente las transformaciones y las relaciones en el

lenguaje. Este metamodelo está basado en el metamodelo de QVT Relations, sin embargo, se ha realizado un esfuerzo de simplificación para que el metamodelo sea compatible con el metamodelo de UML, y además, debido a la naturaleza de la descripción de las relaciones basada en clases en lugar de objetos, el metamodelo de UPT posee algunas diferencias respecto al metamodelo de QVT como la eliminación de conceptos como las colecciones de objetos, el uso de las cardinalidades, el uso de la composición de patrones, definir una dependencia de orden de ejecución entre las transformaciones y relaciones, etc.

Se pretende realizar una descripción del metamodelo, enfocándose principalmente en las nuevas características introducidas por nuestra aproximación.

La descripción del metamodelo de UPT se realiza de forma descendiente, desde los conceptos más genéricos a los más concretos. Así, comienza la descripción con los dos paquetes en los que está constituido el metamodelo (ver Figura. 57). Por un lado, el paquete *Transformaciones* que contiene el concepto Transformación y los elementos que la constituyen y relacionan. Por otro lado, el paquete *Relaciones* que contiene los conceptos vinculados con las relaciones o reglas de transformación. Ambos paquetes están vinculados, puesto el concepto de *Regla* se define en el paquete *Transformaciones* y es utilizado y extendido por el concepto *Relación* en el paquete Relaciones.

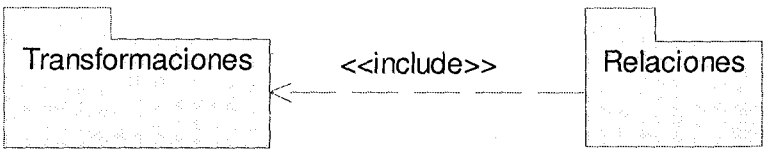

Figura. 57. Paquetes Principales del Metamodelo de UPT

La Figura. 58 muestra el paquete *Transformaciones* del metamodelo. Su concepto principal es la metaclase *Transformación*, la cual tiene una relación recursiva que permite establecer la relación de herencia entre diferentes transformaciones. Por otro lado, se ha definido la clase *OrdenTrans* que permite establecer un orden de ejecución (sucesor y predecesor) entre las diferentes transformaciones, esta característica será muy útil a la hora de definir el mapa de transformaciones. A su vez, una *Transformación* contiene un conjunto de instancias de la clase *Regla* (clase abstracta que sobrescribirá *Dominio*). Cada regla define su nombre y una relación recursiva que permite que una regla pueda sobrescribirse. Para regla también se ha definido la clase *OrdenRegla* que permite establecer un orden de ejecución (sucesor y predecesor) entre las diferentes reglas, esta característica será muy útil a la hora de definir el mapa de composición de las transformaciones. Además, una Regla está compuesta por un conjunto de n instancias de la metaclase *Dominio*. Esta relación de asociación permite que las reglas sean n:m, es decir, n dominios origen y m dominios destino (donde n y m son mayores o igual a 0). Por último, indicar que cada instancia de la clase Dominio tiene dos atributos (1) *esCheckable* especifica si se desea comprobar únicamente el metamodelo (*checkable* a verdadero) o si se desea modificar (*checkable* a falso) y (2) *metamodelo* de tipo String indica que el camino del metamodelo que está siendo referenciado.

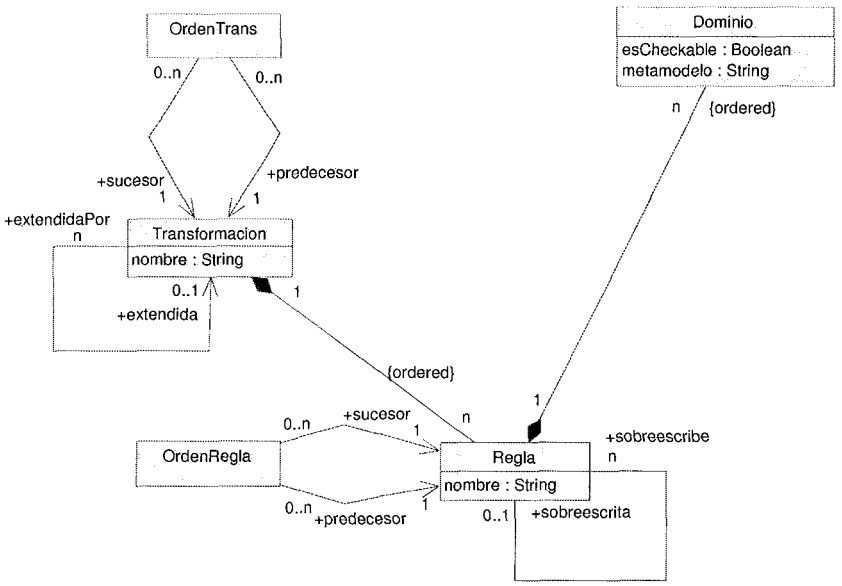

Figura. 58. Paquete Transformaciones del Metamodelo de UPT

El siguiente paquete del metamodelo de UPT se denomina *Relaciones* (ver Figura. 59). Su elemento principal es la *Relación* situada en la parte superior, es la unidad básica dentro de la especificación declarativa de UPT. Se puede apreciar que la clase Relación extiende de Regla heredando así sus propiedades y el vínculo con la clase Dominio. Cada metaclase *Relación* contiene un atributo *esRaiz* que indica si es raíz o no dicha *Relación*, es decir, si su valor es verdadero la relación es invocada directamente por la transformación y si es falso es invocada desde el Where de otra relación. Además contiene dos relaciones con la metaclase *Patrón* que corresponden a las expresiones *When* y *Where*. Por otro lado, se tiene que cada instancia de la clase *Patrón* está formada por un conjunto de Predicados que a su vez contienen una instancia de *ExpresionOCL*. Finalmente, la expresión contiene una referencia a las instancias de la clase *Variable* utilizadas, se almacena su nombre y su tipo que está situado en la clase *PatronClase*. En el lado derecho, en la cima se sitúa la clase *Dominio* especificada en el paquete *Transformaciones*, cada clase Dominio esta relacionada con un *PatronClase* denominado *variableRaiz* que representa la instancia de la metaclase que marca el punto de entrada sobre el metamodelo referenciado en el Dominio. Cada *PatronClase* tiene dos atributos, la clase que es una cadena que indica la metaclase referenciada y la variable que es de tipo *Variable*. Por último indicar que el *PatronClase* contiene además de sus atributos propios, un conjunto de elementos *PatronPropiedad* que le permiten introducir tanto propiedades de tipo simple como y propiedades que referencian a asociaciones con otras instancias *PatronClase*, referenciado mediante una asociación con PatronClase. Para las propiedades simples se indica el valor cuyo tipo una expresiónOCL y si la propiedad *esClave*, es decir, forma parte de la identificación del *PatronClase*. Por otro lado, el *PatronPropiedad*

puede ser un rol de una asociación, en este punto hay que destacar la modificación de UPT, donde se ha introducido la metaclase *PatronAsociacion* que permite establecer las relaciones de asociación entre clases, a diferencia de QVT que son entre objetos. Se incorpora a las relaciones de asociación además la información sobre el nombre del *rol*, su cardinalidad superior e inferior (*cardInf* y *cardSup*) y se indica si la asociación es composición o no (*esComposicion*).

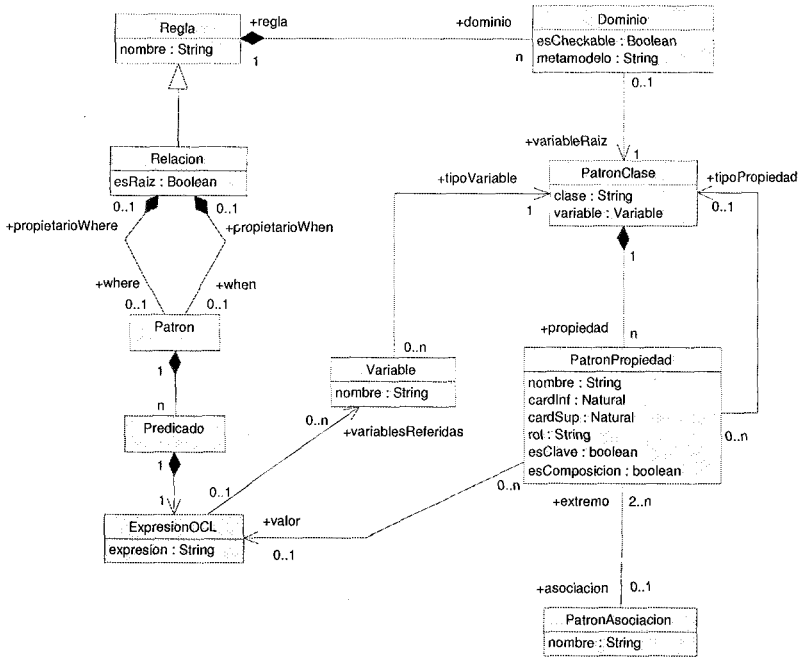

Figura. 59. Paquete Relaciones del Metamodelo de UPT

8.3.3 Definición de los Esterotipos de UPT

UPT realiza una adaptación de cada uno de los conceptos definidos por el metamodelo de nuestra propuesta al metamodelo del estándar UML [95]. Además, UPT necesita establecer restricciones y consultas, extendiendo para ello OCL en su especificación. Se ha seguido el alineamiento propuesto por la especificación OCL [93] que permite insertar las diferentes expresiones OCL en UML. Como UPT ha elegido el diagrama de clases para expresar las transformaciones, se debe buscar el alineamiento de cada uno de los conceptos con las metaclases del diagrama de clases de UML y con las de OCL respecto al modelo. Este alineamiento se describe con profundidad en el apéndice IV.

En las siguientes secciones, se hace uso del lenguaje UPT para la especificación de la transformación T1 definida dentro del proceso de desarrollo de WebSA.

8.4 Transformación T1: Fusión de los Modelos Funcionales con los Modelos de Arquitectura

La primera transformación establecida por el proceso de desarrollo de WebSA, es la transformación T1 (ver Figura. 5). T1 propone la fusión de los modelos de la vista funcional definidos dentro de las propuestas de la Ingeniería Web con los modelos de la vista de arquitectura definida por WebSA. Como resultado de esta fusión se obtiene la vista de integración, compuesta por el modelo de integración (MI). Esta transformación establece un cambio de abstracción dentro del proceso ya que los modelos origen están definidos en la fase de análisis y los modelos destino se definen en la fase de diseño. Sin embargo, tanto el origen como el destino son independientes de plataforma. Siguiendo la terminología definida por MDA, T1 es una transformación PIM a PIM.

T1 es una transformación compleja, es decir, está formada por una sucesión ordenada de transformaciones, que a su vez están compuestas por un conjunto de relaciones. Cada una de las transformaciones más pequeñas que componen T1 se trata de forma particular, permitiendo describir con detalle la transformación. Gracias a la capacidad expresiva de UPT, se puede establecer la ordenación y la composición de las diferentes transformaciones.

Por otro lado, T1 es parcialmente dependiente de los modelos definidos en la parte funcional. Es decir, se tiene que cambiar o instanciar determinadas transformaciones en función de la método de ingeniería Web que se utiliza para definir la navegación. Debido a que estas metodologías poseen modelos de navegación diferentes, se deben definir transformaciones distintas. Para solventarlo, se utiliza la capacidad de herencia de UPT para introducir las transformaciones dependientes sin afectar al resto de transformaciones independientes.

El objetivo de la transformación T1, es evitar que el analista manualmente realice el paso entre el análisis y el diseño con los consiguientes errores e inconsistencias que pueden producirse. T1 automatiza el cambio de abstracción entre dichas fases. Esto permite que exista una trazabilidad precisa entre todos sus elementos y evitar posibles errores introducidos por al intervención humana.

T1 presenta las siguientes propiedades:

- **Establecida de n a m Modelos**: Mas concretamente, parte de un conjunto de modelos origen (n=4), y se genera un modelo destino (m=1).
- **Definida entre Modelos Independientes de Plataforma**: Tanto las vistas funcionales y arquitectónicas origen como la vista de integración destino, no contienen elementos dependientes de una plataforma destino.
- **Compleja**: Es decir está compuesta por la sucesión de transformaciones ejecutadas en un determinado orden, y compuestas a su vez por un conjunto de relaciones.
- **Definida a Nivel de Metamodelo**: las reglas de transformación se establecen entre los elementos de los metamodelos origen y los elementos de los metamodelos destino.
- **Bidireccional**: debido a que UPT es un lenguaje de transformaciones declarativas con la propiedad de bidireccional.

- **Parcialmente Dependiente de la Funcionalidad**: existen transformaciones dependientes del método funcional utilizado y otras que son independientes a ellos.
- **Basada en Elementos y Patrones**: la definición de las reglas de transformación se realiza basándose en la existencia de ciertos elementos, que en ocasiones establecen patrones en los modelos origen. Estos patrones aportan más información a la hora establecer una transformación.
- **Parcialmente Genérica para Cualquier Dominio del Problema**: las transformaciones se definen a nivel de metamodelo y son genéricas para cualquier aplicación Web. Sin embargo, existen ocasiones que se necesita establecer reglas excepcionales ya sean a nivel funcional o arquitectónico. Para estos casos, se definen reglas que buscan la aparición de ciertas ocurrencias particulares para el problema, y si es así, sobrescriben a las reglas genéricas.

Debido a la complejidad de T1, se necesita realizar una descripción general, que defina los diferentes pasos o transformaciones más pequeñas que componen T1. Así, primero se especifica cuáles son las transformaciones que componen T1 y seguidamente se profundiza en las relaciones que componen cada una de ellas.

La Figura. 60 muestra el mapa de transformación de T1. Para ello, se hace uso de la propuesta definida por el lenguaje UPT, en la que se representa el orden de ejecución de las transformaciones.

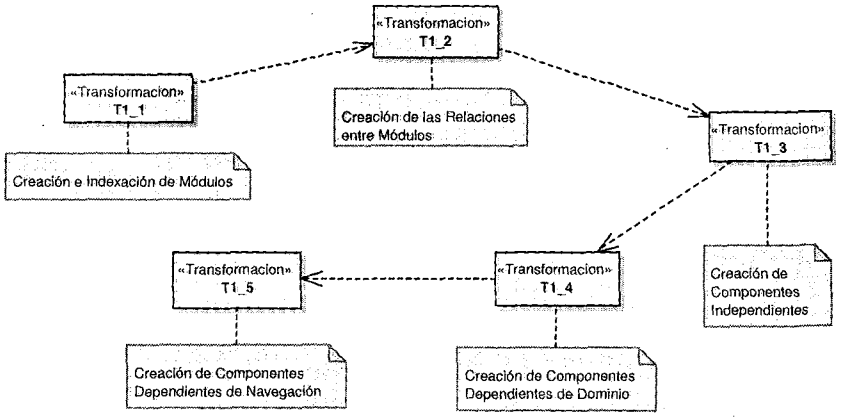

Figura. 60. Mapa de Transformaciones de T1

Siguiendo el orden de ejecución, las transformaciones que componen T1 son las siguientes:

1. **T1_1: Creación e Indexación de Módulos**: Inicialmente esta transformación crea un módulo en el modelo de integración por cada uno de los subsistemas definidos en el modelo de subsistemas. Una vez realizado, se indexa a cada uno de los componentes definidos en el modelo de configuración, según el tipo de componente, apuntado a los módulos previamente creados.

2. **T1_2: Creación de las Relaciones entre Módulos**: Basándose en la indexación de los componentes de configuración, se establecen cuáles son los puertos de los módulos y los conectores entre ellos.
3. **T1_3: Creación de Componentes Independientes**: Esta transformación comienza con la creación de los componentes de integración desde los componentes de configuración que son independientes de la funcionalidad, estos son AgenteUsuario, Almacén, CacheWeb, LibreriaFuncional, PaginaFuncional, PaginaEstilo, VistaLegada, Recurso y GestorConexiones. Con ello se crea un componente de integración por cada componente de configuración independiente, y además, se crean sus propiedades como atributos, operaciones, puertos y conexiones.
4. **T1_4: Creación de los Componentes Dependientes de Dominio**: En esta transformación los modelos origen son el modelo de configuración y el modelo de dominio y el destino es el modelo de integración. Estos modelos proporcionan la información necesaria para crear aquellos componentes de integración y sus propiedades que obtienen información desde la funcionalidad proporcionada por el domino. Los componentes de configuración implicados son DatosEntidad, EntidadWeb, ComponenteProceso, Vista y DAC.
5. **T1_5: Creación de los Componentes Dependientes de Navegación:** Transformación definida a partir del modelo de configuración y el modelo de navegación para crear los componentes de navegación en el modelo de integración. Se crearán tanto los componentes como sus propiedades. Los componentes que intervienen en estas transformaciones son PaginaEstatica, PaginaServidor, ControladorWeb, ObjetoWeb.

Seguidamente una vez definida la secuencia de transformaciones que componen T1, se describen cada una de las transformaciones en detalle. Para ello, se realiza un recorrido sobre el mapa de relaciones que contiene cada transformación y se explica que efecto tiene sobre los modelos origen y destino. Las principales relaciones de T1 especificadas con el lenguaje UPT, pueden consultarse en el apéndice V.

8.4.1 T1_1: Creación e Indexación de Módulos

La transformación T1 sigue una filosofía arriba-abajo, es decir, parte de los elementos de mayor granularidad, para ir bajando el nivel de detalle hasta los elementos más específicos. Así, T1_1 comienza el elemento más grande dentro del modelo de integración, el ModuloWeb. Para su creación, sigue la correspondencia de crear un ModuloWeb por cada uno de los subsistemas definidos en el modelo de subsistemas. Una vez realizada la creación de los módulos, se procede a ubicar los componentes del modelo de configuración en cada uno de los módulos creados. Para hacerlo, T1_1 se basa en la correspondencia que existe en la tipología de componentes respecto a la tipología de subsistemas.

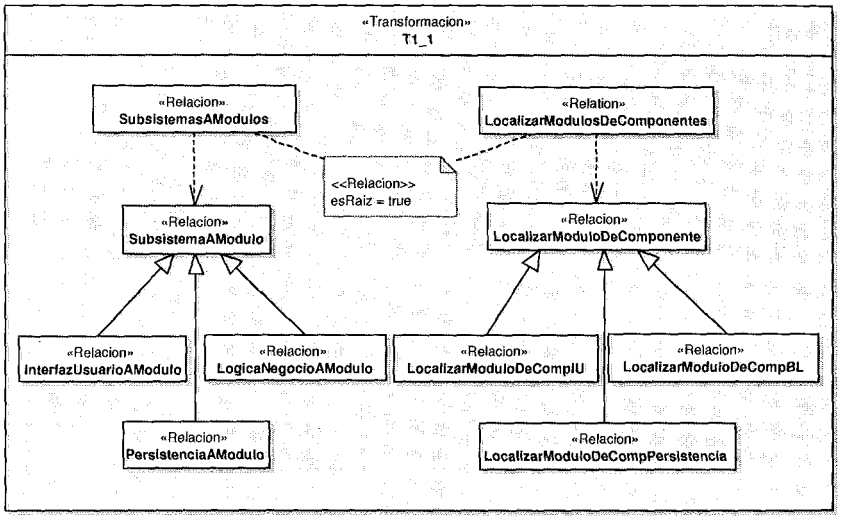

Figura. 61. Mapa de Composición de la Transformación T1_1

La Figura. 61 representa un mapa de relaciones simplificado de la transformación T1_1. Esta notación permite entender mejor las reglas de transformación o relaciones, con un modelo sencillo que muestra la herencia y la relación de orden entre las diferentes reglas. Las relaciones de herencia se definen cuando una regla sobrescribe a otra, mientras que la relación de dependencia se introduce cuando una regla en su ejecución invoca a otras reglas ya sea desde la cláusula Where o When.

La transformación T1_1 comienza la ejecución invocando a dos relaciones raíz. Primero se invoca a la relación *SubsistemasAModulos*, dicha la relación realiza la tarea de recorrer todas las instancias de SubsistemaWeb, y por cada una crear instancia de Módulo, (ver relación en la Figura. 93). Invocará a su vez por cada SubsistemaWeb a la relación *SubsistemaAModulo*, la cual está encargada de transformar las propiedades de un SubsistemaWeb a un Modulo. Sin embargo, no se utiliza directamente sino que es sobrescrita por otras relaciones que corresponden a los diferentes tipos de subsistemas definidos por el metamodelo de subsistemas (p.e. InterfazUsuario, Servidor, LogicaNegocio, etc.). En el Apéndice V se describen las principales reglas de transformación de T1_1.

8.4.2 T1_2: Creación de las Relaciones entre Módulos

La T1_2 es la segunda transformación dentro de la serie de transformaciones que compone T1. T1_2 tiene como objetivo establecer las relaciones o conexiones entre los diferentes módulos. Para ello, se sirve de los resultados obtenidos en la transformación anterior, es decir, las instancias de *ModuloWeb* creadas y la indexación de los componentes del modelo de configuración a los diferentes *ModuloWeb*. T1_2 tomará los casos donde los componentes de configuración estén relacionados entre sí y se indexen a ModulosWeb distintos y establecerá una relación entre los diferentes módulos.

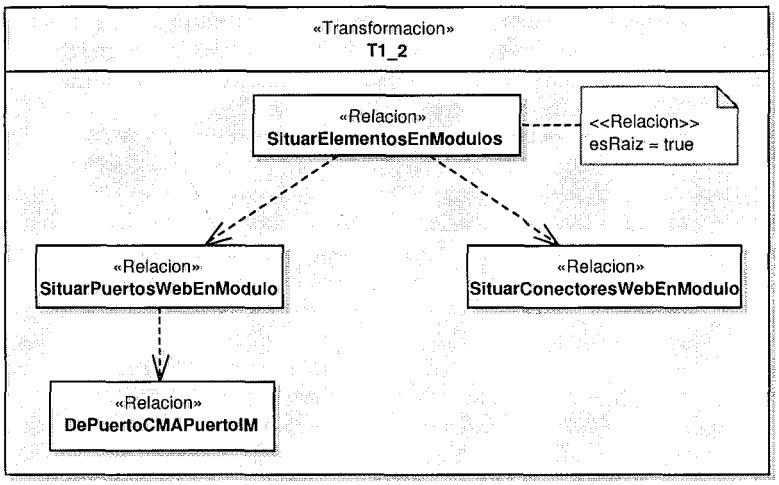

Figura. 62. Mapa de Composición de la Transformación T1_2

La Figura. 62 muestra el mapa de composición de T1_2, en el cual la transformación se vincula con la relación raíz *SituarElementosEnModulos* que comienza la ejecución invocando a las relaciones que sitúan los diferentes elementos de la conexión. Por un lado, *SituarConectorWebEnModulo*, y por otro lado, *SituarPuertoWebEnModulo*. Esta última relación invocará a la relación *DePuertoCMAPuertoIM*. En el Apéndice V se describen las principales reglas de transformación de T1_2.

8.4.3 T1_3: Creación de Componentes Independientes

T1_3 se inicia con la creación de los componentes de integración que son independientes de la funcionalidad y su colocación en los módulos. Es decir, aquellos componentes que únicamente tienen información procedente de la vista de arquitectura. Estos componentes tienen como característica mantener una correspondencia uno a uno entre un componente de integración y un componente de configuración. Son los siguientes tipos de componentes: *ControladorWeb, AgenteUsuario, Almacén, CacheWeb, LibreriaFuncional, PaginaFuncional, PaginaEstilo, VistaLegada, Recurso* y *GestorConexiones*.

Además, junto al propio componente se crean propiedades como atributos, operaciones y puertos. Y por último, las posibles relaciones entre los componentes de este tipo mediante interfaces y conectores. El resto de relaciones son creadas en posteriores transformaciones.

La Figura. 63 muestra el mapa de composición de la transformación T1_3. El punto de partida es la relación raíz *SituarCompWebCMIndyEnModulo*. Esta relación se encarga de situar los componentes de configuración independientes en el módulo al que apunta su propiedad *moduloDestino*. Seguidamente, la relación invoca a *CrearCompWebIMIndy* recibiendo el componente de configuración y el modulo correspondiente y procediendo a la creación del componente de integración dentro del módulo. Una vez creado el componente, se invoca a *SituarPropiedades-*

CompWebIMIndy que crea las propiedades del componente de integración a partir de las propiedades de configuración. A partir de las instancias de las propiedades se invocan a las relaciones encargadas de completar los valores de dichas propiedades, llamando a *CrearOperacionIMEnComp, CrearFinalConectorWeb, CrearInterfazWebOfertado* y *CrearInterfazWebRequerido.*

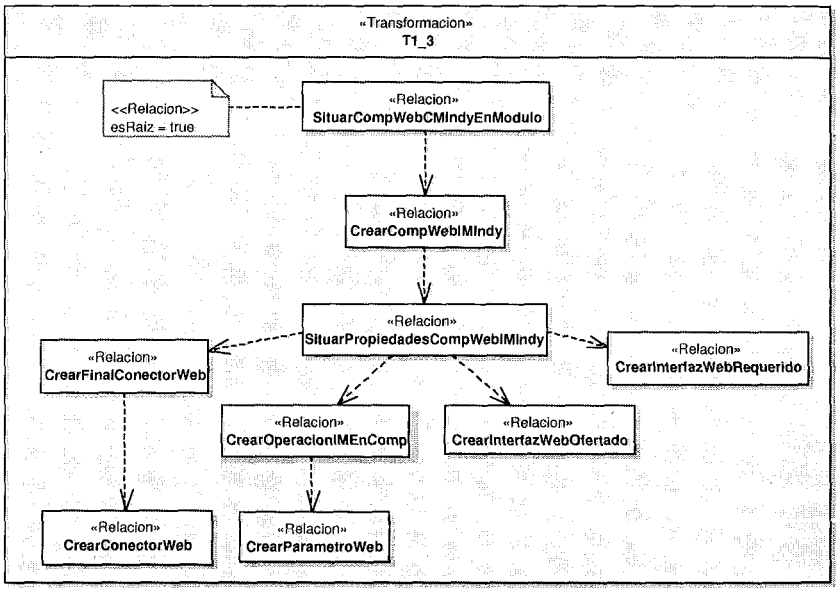

Figura. 63. Mapa de Composición de T1_3

Para finalizar, *CrearOperacionIMEnComp* invoca a la relación *CrearParametroWeb* que procede a crear cada uno de los parámetros. Y la relación *CrearFinalConectorWeb* invoca a *CrearConectorWeb* la cual establecerá la conexión con otro componente independiente.

En el apéndice V se muestran las relaciones de mayor relevancia o que pueden aportar aspectos nuevos de T1_3.

8.4.4 T1_4: Creación de los Componentes Dependientes de Dominio

La transformación T1_4 procede a la creación de los componentes de integración que están vinculados con la funcionalidad proporcionada por el dominio del problema. Se establece por lo tanto que los modelos origen son (1) el modelo de configuración junto con (2) el modelo de dominio y el destino es el (3) modelo de integración. T1_4 según MDA guide [87] es una transformación de tipo fusión (merge). Los componentes Web implicados son del tipo DatosEntidad, EntidadWeb, ComponenteProceso, Vista y CAD. Estos componentes son completados con información proveniente de las clases del dominio y por sus propiedades. Además, T1_4 procede a crear un componente de integración por cada clase encontrada en el modelo de dominio.

Seguidamente, se describe mediante el mapa de composición de la transformación T1_4 (ver Figura. 64). Este mapa describe las relaciones que fusionan la arquitectura y funcionalidad. La transformación T1_4 se inicia una relación raíz llamada *SituarCompWebCMDominioEnModulo* cuya función es la de situar en que ModuloWeb se genera el componente de integración a partir de la información que contiene CompWebCM. Además, inicia el recorrido por el conjunto de clases que contiene el modelo de dominio. Dicha relación invoca desde la cláusula Where a otra relación llamada *CrearCompWebIMDominio* que procede a la creación de la instancia de CompWebIM en el módulo correspondiente. A continuación una vez creado el componente, se procede a introducir sus propiedades, para ello se invoca a la relación *SituarPropiedadesCompWebIMDominio*, encargada de la creación del nombre del componente, junto a la invocación al conjunto de relaciones cada una de las cuáles procede a crear propiedades diferentes del componente. A continuación se describe la relación principal junto a las relaciones invocadas para completar la introducción de las propiedades en el componente de integración:

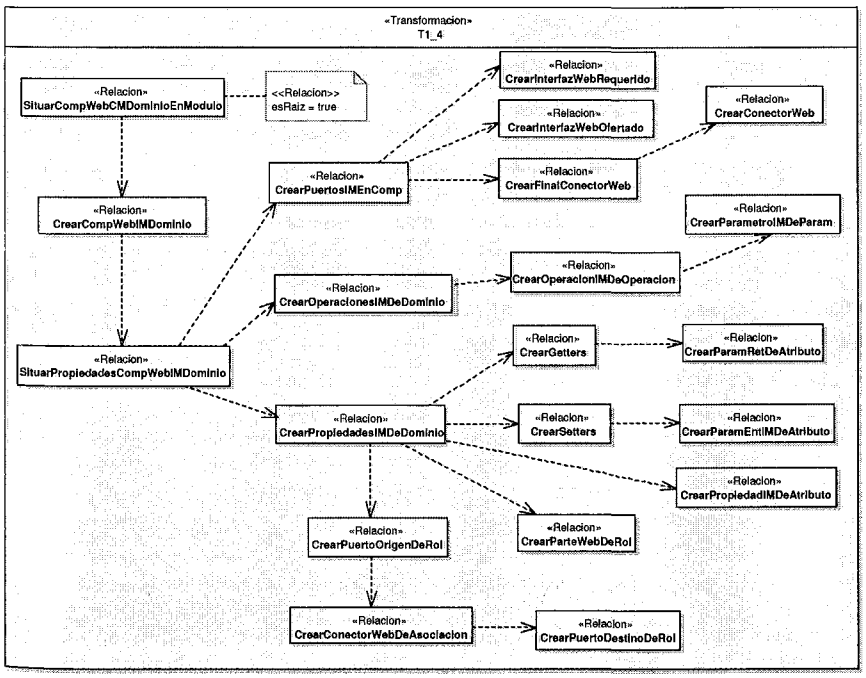

Figura. 64. Mapa de Composición de la Transformación T1_4

1. ***CrearPuertosIMEnComp*** cuya función es crear los puertos que el componente de configuración tenga para crear los mismos puertos en el componente de integración. Además, a partir de los puertos surgen tanto conectores como interfaces, así que invocará a *CrearInterfazWeb-Requerido* y *CrearInterfazOfertado* para crear los interfaces que surgen

del puerto. Para crear los conectores invocará a *CrearFinalConectorWeb* la cual se completa invocando a *CrearConectorWeb* para crear el conector completo y conectarlo a otro componente.

2. ***CrearOperacionesIMDeDominio*** crea un conjunto de elementos OperacionIM por cada una de las operaciones definidas por una clase de dominio, para ello invoca a las relaciones que crean una operación con sus propiedades *OperacionIMDeOperacion* y sus parámetros *CrearParametroIMDeParam*.
3. ***CrearPropiedadesIMDeDominio*** procede a crear las propiedades del componente a partir de los atributos del modelo de dominio. En el caso de que el componente de configuración indique que tiene propiedades (valor definido tienePropiedades) en el componente de integración se crearán sus setters y los getters, es decir, las operaciones que permiten escribir y obtener los valores de las diferentes propiedades. Las relaciones *CrearSetters* y *CrearGetters* invocarán a su vez a las relaciones que permiten crear sus parámetros de entrada y de salida. En el caso de que los atributos sean roles de una asociación, se invocará a la relación *CrearPuertoOrigenDeRol* que crea un PuertoWeb origen de la relación. Del PuertoWeb surgirá un ConectorWeb creado mediante la relación *CrearConectorWebDeAsociacion*. Finalmente se establecerá la conexión con el componente destino creando el puerto mediante la relación *CrearPuertoDestinoDeRol*.

La transformación T1_4, como se puede apreciar en el mapa de la Figura. 64 está constituida por 21 relaciones, por ello se aborda únicamente la descripción de las relaciones más importantes o que introducen relaciones con aspectos no tratados hasta ahora. El resto de relaciones pueden consultarse en el apéndice V.

8.4.5 T1_5: Creación de los Componentes Dependientes de Navegación

La transformación T1_5 procede a la creación y ubicación en módulos de los componentes de integración que están vinculados con la funcionalidad proporcionada por la navegación. Los modelos origen son (1) el modelo de configuración junto con (2) el modelo de navegación y el modelo destino es el modelo de integración. El modelo de navegación, como se ha indicado en el capítulo 4, puede ser definido por las aproximaciones OO-H como en Meliá & Gómez [75] y UWE como en Meliá et al. [78]. Las transformaciones mostradas en este capítulo se centran en la integración con el modelo de navegación llamado DAN definido por OO-H [19]. El modelo de navegación de OO-H proporciona la información necesaria para crear aquellos componentes de integración y las propiedades que obtienen información desde la funcionalidad proporcionada por la navegación, se trata por lo tanto una transformación de tipo fusión (merge). Los componentes de configuración son PaginaEstatica, PaginaServidor y ObjetoWeb. A la hora de proceder a su creación en el modelo destino, no solamente se completa los componentes de integración sino que se creará un componente de integración por cada instancia de ClaseNavegacional del modelo de navegación. Seguidamente se describe a través del mapa de composición de la transformación T1_5 (ver Figura. 65), pueden apreciarse las relaciones que realizan la tarea de fusionar arquitectura y la navegación.

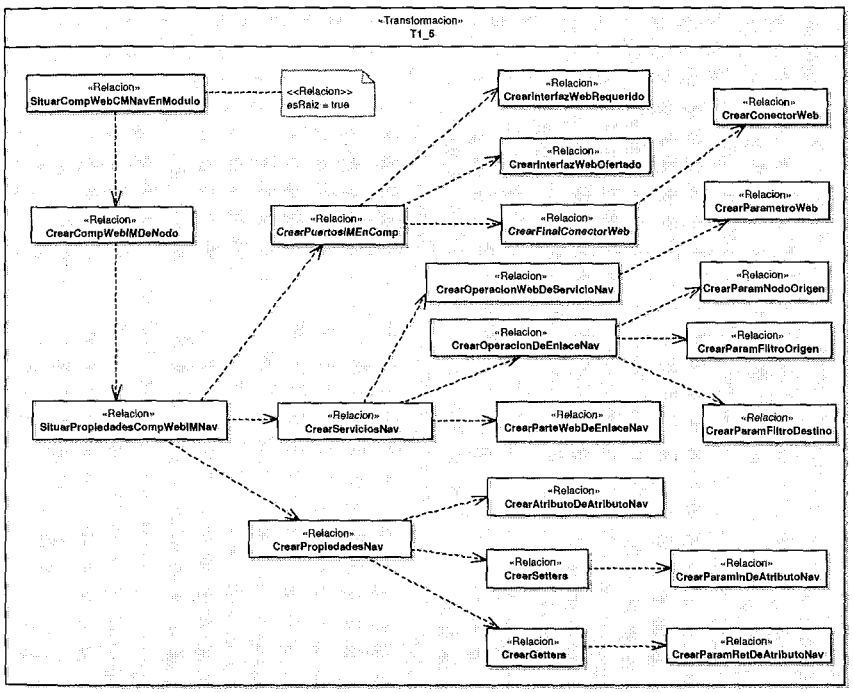

Figura. 65. Mapa de Composición de la Transformación T1_5

Se inicia la transformación T1_5 la relación raíz *SituarCompWebCMNavEnModulo* cuya función es la de situar en que modulo se genera el componente a partir de la información que contiene CompWebCM, y a su vez, iniciar el recorrido por el conjunto nodos que contiene el modelo de navegación de OOH. Dicha relación invoca desde su parte Where a la relación *CrearCompWebIMDeNodo* que procede a la creación de las instancias de CompWebIM a partir de las instancias de ClaseNavegacion y Colección. Una vez creado el componente que representa a la clase de navegación, se introducen sus propiedades, para ello se invoca a la relación *SituarPropiedadesCompWebIMNav*, encargada de la creación del nombre del componente, junto a la invocación al conjunto de relaciones cada una de las cuáles procede a crear propiedades diferentes del componente. A continuación se describe la relación principal junto a las relaciones invocadas para completar la introducción de las propiedades en el componente de integración:

1. ***CrearPuertosIMEnComp,*** cuya función es crear los puertos que el componente de configuración tenga para crear los mismos puertos en el componente de integración. Además, a partir de los puertos surgen tanto conectores como interfaces, así que invocará a *CrearInterfazWeb-Requerido* y *CrearInterfazOfertado* para crear los interfaces que surgen del puerto. Para crear los conectores invocará a *CrearFinalConectorWeb* la cual se completa invocando a *CrearConectorWeb* para crear el conector completo y conectarlo a otro componente. Estas relaciones al ser

independientes de la funcionalidad son las mismas que las indicadas en la transformación T1_4.

2. ***CrearServiciosNav,*** encargada de crear las propiedades de comportamiento del componente, es decir, las instancias de OperacionWeb. Para ello, (1) invoca a ***CrearOperacionWebDeServicioNav*** que como su nombre indica, crea un OperacionWeb desde el ServicioNavegacional definido por OOH, esta relación a su vez invoca a ***CrearParametroWeb*** para crear un ParametroWeb desde cada ParametroNavegacional. A continuación, (2) crea un OperacionWeb a partir de un EnlaceNavegacional mediante la relación ***CrearOperacionWebDeEnlaceNav*** (ver con detalle en la Figura. 111). Esta relación invocará a las relaciones ***CrearParamNodoOrigen, CrearParamFiltroOrigen*** y ***CrearParam-FiltroDestino,*** las cuáles crean un ParametroWeb para propiedad del EnlaceNavegacional. (3) crea una ParteWeb desde un EnlaceNavegacional mediante la relación ***CrearParteWebDeEnlaceNav,*** que permite componer componentes cuando el EnlaceNavegacional tiene la propiedad esMismoNodo=true.
3. ***CrearPropiedadesNav,*** procede a la creación de las propiedades estáticas del componente y a las operaciones de acceso a ellas, a partir de los atributos navegacionales definidos en el modelo de navegación de OOH. Para ello, primero invoca a (1) ***CrearAtributoDeAtributoNav*** que procede a la creación de un atributo en el componente de integración a partir de un atributo de la clase Navegacional. A continuación, (2), invoca a ***CrearSetters*** que creará un OperacionWeb que permite la escritura sobre el atributo del componente de integración. Además, esta relación invoca a ***CrearParamInDeAtributoNav*** que crea los parámetros de entrada de la operación Setter a partir de los atributos. Por último, (3) invoca a ***CrearGetters*** que crea un OperacionWeb que permite la lectura sobre el atributo del componente de integración. Además, esta relación invoca a ***CrearParamRetDeAtributoNav*** que crea el parámetro de retorno del setter a partir del atributo.

La transformación T1_5, como muestra Figura. 65 contiene por 22 relaciones, por ello en este capítulo se realiza la descripción de las relaciones más importantes o que introducen relaciones con características no tratadas. El resto de relaciones pueden consultarse en el apéndice V.

8.5 Transformación T2: Integración de los Aspectos Dependientes de Plataforma

La segunda transformación establecida por el proceso de desarrollo de WebSA, es la llamada transformación T2 (ver Fig. Figura. 5). T2 propone la integración de los aspectos dependientes de la plataforma sobre el modelo de integración (MI) definido en la fase de diseño. La integración se realiza mediante la definición de transformaciones modelo a texto definidas en la propuesta MOFScript [94], que ha sido enviado como respuesta a la propuesta de estándar de la OMG *MOF to Text Transformation Language* [91]. Estas transformaciones realizan un recorrido de MI,

en el cual por cada uno de los componentes especificados en el diseño se obtiene el código correspondiente de las diferentes plataformas J2EE y .NET. Esta transformación siguiendo la terminología definida en MDA Guide [87] es PIM a PSM. Por un lado, el modelo origen MI, es un modelo independiente de la plataforma destino. Por otro lado, el resultado de T2 es un PSM representado directamente por la implementación de las diferentes plataformas.

T2 es una transformación compuesta por reglas de transformación. Las reglas de transformación en MOFScript son llamadas TextMappingOperation, y contienen un recorrido por el metamodelo MOF del modelo origen y el correspondiente mapeo al código destino.

La transformación T2 es dependiente de la plataforma destino. Así, el proceso de WebSA define diferentes transformaciones T2 (T2', T2'', etc.) para cada una de las plataformas destino. Sin embargo, hay un conjunto de transformaciones que constituyen el núcleo de T2 que pretende que sea común para cualquiera de las plataformas.

El principal objetivo de T2 es culminar el proceso de desarrollo de WebSA, convirtiendo el modelo de integración definido en la fase de diseño en la representación de la implementación final. Se completa así un proceso automático que permite que se obtenga el código de una aplicación Web y tan solo los servicios de lógica de negocio deben ser codificados por el programador para poseer una implementación completa.

Las propiedades más importantes de la transformación T2 son las siguientes:

- Establecida de n a m modelos. Donde los modelos origen son n = 3. Se parte del modelo de integración para generar los componentes de implementación. Y para completar los almacenes de información, se necesita la información procedente del modelo de dominio y el modelo de navegación. Por su lado, los modelos destino son m=2 que son las plataformas destino J2EE y .NET.
- PIM a PSM. Definida entre modelos independientes de plataforma en el origen, tanto el modelo de integración, navegación y de dominio son independientes de plataforma. El destino es un PSM representado por la implementación en una plataforma concreta.
- Compleja: Es decir está compuesta por la sucesión de transformaciones ejecutadas en un determinado orden y compuestas a su vez por un conjunto de reglas de transformación modelo a texto.
- Definida a nivel de metamodelo: las reglas de transformación se establecen mediante el recorrido de los elementos de los metamodelos origen y la generación de los elementos destino.
- Unidireccional: MOFScript es un lenguaje de transformaciones que establece la conversión modelo a texto, pero no su inversa.
- Parcialmente dependiente de la funcionalidad: existen transformaciones dependientes del método funcional (ya sea OO-H o UWE) y otras que son independientes a ellos.
- Parcialmente genérica para cualquier dominio del problema: las transformaciones al definirse a nivel de metamodelo son genéricas para cualquier dominio del problema. Sin embargo, existen ocasiones que se necesita establecer reglas excepcionales ya sean a nivel funcional o

arquitectónico. Para estos casos, se definen reglas que buscan la aparición de ciertas ocurrencias particulares para este problema, y si es así, sobrescriben a las reglas genéricas.

A continuación, se describe brevemente el lenguaje de transformaciones MOFScript utilizado para la especificación de la transformación T2.

8.5.1 MOFScript: Lenguaje de Transformaciones Modelo a Texto

MOFScript [94] propone la definición de un lenguaje de transformaciones modelo a texto basadas en el lenguaje imperativo OperacionalMappings del estándar QVT [91]. Para ello, MOFScript ha extendido las metaclases definidas en QVT introduciendo las capacidades necesarias para el tratamiento de texto. Para poder entender las transformaciones especificadas en MOFScript, es necesario conocer sus conceptos principales:

- **TextTransformation**: extiende la metaclase OperationalTransformation de QVT. TextTransformation es el contenedor del conjunto de reglas de transformación específicas de modelo a texto.
- **TextMappingOperation**: representa el concepto de regla de transformación dentro de MOFScript. Forma parte de un TextTransformation. Extiende la metaclase ImperativeOperation de QVT.
- **TextMappingEntryOperation**: es una regla de entrada que define un punto de entrada para la TextTransformation. Extiende la clase EntryOperation de QVT.
- **TextMappingDescriptor:** define un descriptor global de propiedades para transformaciones. Implementa básicamente un patrón singleton que contiene propiedades configurables MOF que controlan aspectos de las transformaciones modelo a texto.
- **TextOperationBody:** representa el cuerpo de un TextMappingOperation. Su contenido está basado en especializaciones de OCLExpression que se verá a continuación.
- **OutputExpression:** representa una expresión que genera una salida de texto hacia un dispositivo, normalmente un fichero o la consola. Es una especialización de OCLExpression.
- **EscapedOutputExpression:** es una especialización de OutputExpression que permite a texto escapado ser parte de la salida textual. El texto escapado es texto embebido en una regla como texto estándar el cual es enviado a la salida. Es el mismo mecanismo usado por los lenguajes de script como JSP, ASP, y lenguajes de plantillas como Velocity.

El resto de elementos definidos en MOFScript están enfocados en la posibilidad de permitir el tratamiento de texto mediante la definición de Tipos simples como String, Integer, Real, Boolean; estructuras de almacenamiento como List, Dictionary y Tuple; Iteradores para recorrer estructuras como ForEachExp; sentencias de control como IfExp; y por último, elementos que permiten la trazabilidad y el mantenimiento del código introducido manualmente por el programador. Para profundizar más en cada uno de estos elementos consultar la especificación de MOFScript [94].

A continuación, se describen las reglas de modelo a texto que constituyen a la transformación T2.

8.5.2 Estructura Principal de la Transformación T2

La transformación T2 está constituida por un conjunto de reglas de transformación modelo a texto que realizan el recorrido por cada uno de los elementos que contiene el modelo MI para obtener la implementación final de la aplicación Web. Como se ha definido en el capítulo 7, MI es un modelo de componentes en el que se han fusionado los aspectos de arquitectura con los aspectos funcionales. Por un lado, tanto los elementos ModuloWeb como los CompWebIM tienen información sobre el subtipo que representa. Por ejemplo, el ModuloWeb puede representar una capa "Presentación" o una capa "Servidor" y el CompWebIM puede ser del tipo "PaginaServidora" o "Almacén". Esta información, proporciona la posibilidad de establecer una transformación a código mucho más precisa para la plataforma destino. Por otro lado, cada uno de los CompWebIM contiene un conjunto de propiedades tanto estáticas como de comportamiento que permiten generar la funcionalidad necesaria a cada uno de los componentes.

En la Figura. 66 puede apreciarse el conjunto de reglas de transformación de texto que constituyen T2. Para su descripción se sigue la misma notación utilizada para T1, basándose en un diagrama de clases para representar las reglas que constituyen la transformación.

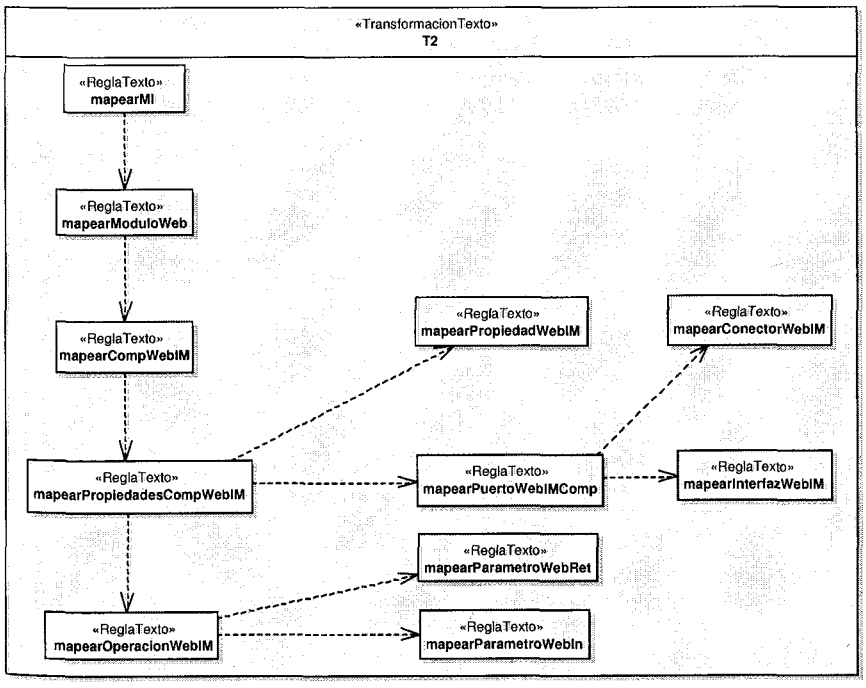

Figura. 66. Mapa de Reglas Principales de la Transformación T2

El mapa comienza con una clase que representa la transformación T2. La transformación T2 referencia en primera instancia a una regla de tipo

TextMappingEntryOperation llamada *mapearMI*, que marca el punto de entrada de T2. Seguidamente, *mapearMI* está constituida por un conjunto de reglas de tipo TextMappingOperation llamada *mapearModuloWeb*. Esta regla se encarga de transformar un *ModuloWeb* en el código correspondiente en la plataforma destino. Además, cada *ModuloWeb* está constituido por un conjunto de componentes CompWebIM, así que invocará a la regla *mapearCompWebIM* por cada uno de los componentes que contiene. Para mapear un CompWebIM se recorren cada uno de los elementos que lo constituyen, así la siguiente regla invocada se denomina *mapearPropiedadesCompWebIM*, esta regla se encarga de invocar a cada una de las propiedades de CompWebIM. Primero invoca a *mapearPropiedadWebIM*, encarga de generar el código asociado a la propiedad estática PropiedadWeb. A continuación invoca a *mapearPuertoWebIM*, esta regla generará el código asociado al puerto y a su vez de los elementos ConectorWebIM e InterfazWebIM que parten de él, para ello invoca a *mapearConectorWebIM* y *mapearInterfazWebIM*. Por último, se transforman a código las propiedades de comportamiento del componente, para ello se invoca a *mapearOperacionWebIM* que contiene un conjunto de parámetros que se transforman a texto mediante *mapearParametroWebRet* para los parámetros de retorno y *mapearParametroWebIn* para los parámetros de entrada.

Sin embargo, este mapa no cubre los casos de los diferentes tipos de ModuloWeb y CompWebIM que puede contener un modelo MI. Para ello, se ha definido mediante el mecanismo de herencia, un conjunto de reglas de transformación cada una de las cuáles es específica para cada uno de los tipos.

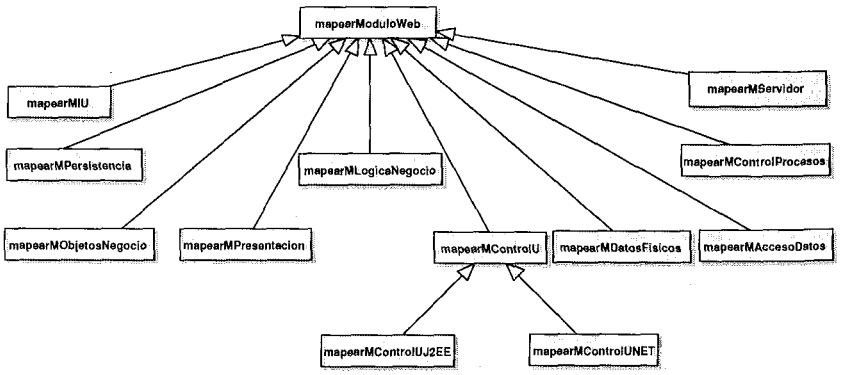

Figura. 67. Mapa Simplificado de Reglas de ModuloWeb

En la Figura. 67 se muestra el mapa de las diferentes reglas definidas para la transformación a código de un ModuloWeb. Consiste en una jerarquía de herencia donde la clase raíz es mapearModuloWeb que está definida en el mapa principal de T2 (ver Figura. 66). De la regla mapearModuloWeb extienden cada uno de los subtipos correspondientes a cada una de las capas que puede representar un módulo, así se tiene *mapearMIU*, que transforma a código un ModuloWeb que representa la Interfaz de Usuario, *mapearPersistencia* que transforma a un ModuloWeb que representa la capa de persistencia y así para todos los casos.

Una vez se ha definido una relación por cada una de las capas y subcapas que representa cada ModuloWeb, a su vez se extiende cada regla especializándose en cada una de las plataformas a las que se transforma. Por ejemplo, en mapearMControlU se especializa en mapearMControlUJ2EE para transformar en la plataforma J2EE y mapearMControlUNET para transformar a la plataforma .NET.

Siguiendo con la especialización de las reglas de transformación se continúa con la especialización de CompWebIM. Como ya es sabido cada uno de los componentes definidos en WebSA se basan en una jerarquía de tipos definidos dentro del dominio de las aplicaciones Web. Esto permite que en T2 pueda establecerse un mapeo más preciso de un componente al código que representa dentro de una aplicación Web. En la Figura. 68 se representa un mapa simplificado de las diferentes reglas que transforman los tipos de CompWebIM a código. Como se aprecia la regla raíz de la jerarquía es mapearCompWebIM de la cual extienden un conjunto de reglas, una por cada uno de los diferentes tipos de componentes definidos en WebSA, por ejemplo mapearCPaginaServidora se encarga de transformar en el código que representa a un componente de tipo PaginaServidora.

Además, algunas de las reglas especializadas en un determinado tipo de componente ha de especializarse para transformase en el código específico de una determinada plataforma. Por ejemplo, mapearCLibreriaFuncional se transforma en código java cuando esta regla se especializa en mapearCLibreriaFuncionalJ2EE y en código C# cuando se especializa en la regla mapearCLibreriaFuncionalNET.

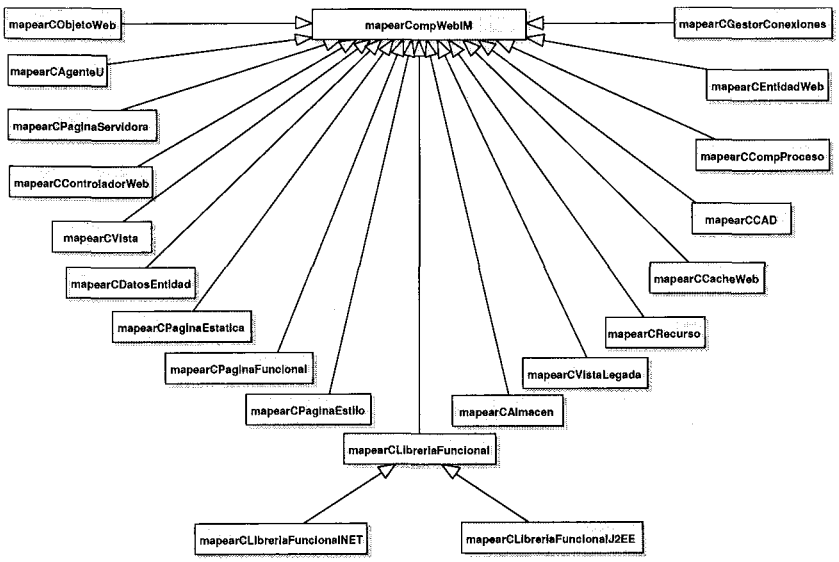

Figura. 68. Mapa Simplificado de las Reglas de CompWebIM

El resto de reglas que generan la información de las propiedades de un CompWebIM se especializan en cada una de las plataformas destino, por ejemplo

mapearPropiedadWebIM se especializa en mapearPropiedadWebIMJ2EE y en mapearPropiedadWebIMNET.

Es importante destacar que estas reglas de transformación son las que la aproximación WebSA proporciona por defecto, pero que pueden extenderse posteriormente añadiendo nuevos tipos de componentes Web o nuevos tipos de plataformas destino.

A continuación, siguiendo la estructura de T2 se muestran las reglas de transformación de T2 más importantes siguiendo la notación del estándar MOFScript.

8.5.3 Reglas de transformación de T2 en MOFScript

La especificación de las reglas de transformación de T2 mediante el lenguaje de transformaciones MOFScript, supone un esfuerzo de adaptación de la estructura de la transformación de T2 al propio lenguaje MOFScript. La principal adaptación se debe fundamentalmente a que MOFScript como mecanismo de herencia entre reglas únicamente soporta la sobreescritura y no soporta la extensión. Esto supone tener que modificar el código cuando se realiza uso de la herencia, ya que no puede introducirse código común en las reglas genéricas que posteriormente sea completado por las reglas especializadas, sino que se introduce de forma completa en la especializada.

A continuación se mostrarán aquellas reglas que sean más importantes por su papel en T2 y aquellas que aportan características a destacar. Siguiendo el mapa de reglas principales de T2 (ver Figura. 66) parte de la definición de la transformación de texto T2, que comienza con la declaración de la propia transformación que recibe como parámetro la ruta al modelo de integración dentro del metamodelo de WebSA. Al modelo de integración se le asigna la variable mi, la cual permite que el modelo de integración pueda ser referenciado en el resto de reglas. Al inicio de la transformación se establecen un conjunto de propiedades que son comunes para todas las reglas de transformación y cuyos valores son configurables para cada ejecución. Los valores establecen la plataforma destino, en este ejemplo se ha fijado el valor "J2EE", a continuación se establece el nombre del paquete y el directorio del paquete org.WebSA, la extensión de los ficheros que coloca el valor ".java" y por último el autor.

La primera regla de transformación (ver Figura. 69) que invoca T2 es mapearMI, que consiste en una TextMappingEntryOperation, es decir, la regla que es el punto de entrada de la transformación y recibe el nombre de main. En el cuerpo de esta regla, se recorren todos los elementos ModuloWeb que contiene MI y para cada uno de ellos se invoca a mapearModuloWeb.

```
texttransformation T2 (in
mi:WebSA.ModelosComponentes.ModeloIntegracion.MI)
property plataforma = "J2EE"
property nombre_paquete = "org.WebSA"
property dir_paquete = "org/WebSA"
property extFichSer = ".java"
property extFichWeb = ".jsp"
property author = "Santiago Meliá"
mi::main (){
          self.miModulos->forEach (p:mi.ModuloWeb){
              p.mapearModuloWeb()
          }
}
```

Figura. 69. Regla de Transformación de Inicio de T2

La siguiente regla de transformación es mapearModuloWeb (ver Figura. 70) que como su nombre indica consiste en convertir en código una instancia de ModuloWeb. La tarea principal de la relación es discernir en que tipo de capa hace referencia el ModuloWeb (IU, Servidor, LogicaNegocio, etc.) y a partir de esa información invocar a cada uno de las reglas específicas de cada una de las capas (mapearMIU, mapearMServidor, etc.).

```
mi.ModuloWeb::mapearModuloWeb () {
var nombre = self.getNombre()
var capa = self.capa()
if (capa = "IU")   self.mapearMIU()
else if (capa = "Servidor") self.mapearMServidor()
else if (capa = "LogicaNegocio")
          self.mapearMLogicaNegocio()
…. //se realiza lo mismo para todas las capas
}
```

Figura. 70. Regla de Texto mapearModuloWeb

Siguiendo con el código de T2, la siguiente regla de transformación invocada en función de la capa del ModuloWeb (p.e. Servidor) sería *mapearMServidor* (ver Figura. 71) . En el presente ejemplo presentado para mostrar el código MOFScript de T2 la plataforma seleccionada es J2EE. Entonces, en mapearMServidor se invoca a la

regla mapearMServidorJ2EE. Seguidamente, se invoca a mapearPuertoServidor por cada uno de los puertos que contiene el presente módulo. Por último, la regla sirve también de punto de entrada para el recorrido de los diferentes componentes que contiene el módulo.

```
mi.ModuloWeb::mapearMServidor () {
 if (plataforma = "J2EE"){
          self.mapearMServidorJ2EE()
 else if (plataforma = ".NET")
          self.mapearMServidorNET()
 }
self.puertos->forEach(c:mi.PuertoWebIM) {
          c.mapearPuertoServidor(nombre)
}
//se invoca a la transformación a código de los
componentes del módulo
self.componentes->forEach(c:uml.CompWebIM) {
          c.mapearCompWebIM()
} }
```

Figura. 71. Regla de Texto mapearMServidorJ2EE

A continuación, se muestra la regla *mapearMServidorJ2EE* (ver Figura. 72). Esta regla contiene la generación de texto para los ficheros encargados de realizar la compilación de los componentes de servidor y el ensamblado de estos componentes. Para ello, se obtienen dos ficheros, uno que compila los componentes del servidor, y otro que ensambla los componentes ya compilados en un fichero comprimido con la extensión .jar.

```
mi.ModuloWeb::mapearMServidorJ2EE() {
   // Contiene código necesario para compilar y
   // ensamblar todos los componentes del servidor
   file (dir_paquete + self.getNombre()+ "_compilar"
            + getNombre() + ".bat")
   <% set CPATH=.;%CLASSPATH% %>
    self.componentes->forEach(c:uml.CompWebIM) {
            println(c.codigoCompilacion())
   file (dir_paquete + self.getNombre() + "_ensamblar"
            + getNombre() + ".bat")
   <%jar cf %> self.getNombre() <%.jar %> dir_paquete
   <%\componentesServidor\*.class%>
   <%jar uf %> self.getNombre() <%.jar %> dir_paquete
   <%\descriptoresServidor\*.class%>
}
```

Figura. 72. Regla de Texto mapearMServidorJ2EE

Las siguientes reglas son las encargadas de la transformación de los diferentes tipos de CompWebIM a texto siendo las que mayor contenido aportan a la obtención de código a partir del modelo MI de WebSA. La invocación a dichas reglas comienza con la invocación realizada desde la regla *mapearModuloWeb* a la regla *mapearCompWebIM* (ver Figura. 73). Esta regla genérica se encarga de discernir sobre de que tipo de componente se trata e invoca a una regla de transformación específica para ese tipo. Además, la regla recibe como parámetro el tipo del módulo sobre el que se genera el componente, esto implica en algunos tipos de componente (p.e. DatosEntidad) que la generación sea diferente. Por lo tanto, en ese caso el tipoModulo es enviado como parámetro a la regla de transformación *mapearDatosEntidad.*

```
mi.CompWebIM::mapearCompWebIM (tipoModulo:String) {
var nombre = self.getNombre()
var tipo = self.getTipoComponente()
if (tipo = "Almacen") self.mapearCAlmacen()
else if (tipo = "CacheWeb") self.mapearCCacheWeb()
else if (tipo = "DatosEntidad")
          self.mapearDatosEntidad(tipoModulo)
else if (tipo = "VistaLegada")
          self.mapearVistaLegada()
else if (tipo = "Recurso") self.mapearRecurso()
// ... se invocan al resto de tipos que completan los 19
tipos de componentes Web }
```

Figura. 73. Regla de Texto mapearCompWebIM

Una vez elegido el tipo de componente la transformación debe situarse en la plataforma sobre la cual se convertirá a código. Por ejemplo, el tipo de componente es PaginaServidora y para este tipo de componente existen tantas reglas como plataformas destino, *mapearPaginaServidoraNET* que convierte a texto una PaginaServidora para la plataforma .NET, *mapearPaginaServidoraJ2EE*: convierte a texto una PaginaServidora para la plataforma J2EE, etc.

A continuación, se muestra la regla de transformación *mapearPaginaServidoraJ2EE* (ver Figura. 74).

La regla mapearPaginaServidoraJ2EE transforma una componente PaginaServidora del modelo MI en un componente JSP de la plataforma J2EE. La generación de la página comienza por la cabecera se indica que se trata de una página de tipo JSP, y a continuación se cargan los componentes a los cuáles referencia desde dicha página. Para ello, se invoca a la función *obtenerComponentesRelacionados*, que obtendrá aquellos CompWebIM que están relacionados con el actual componente a través de uno de los PuertoWeb. Seguidamente se genera el texto HTML de la propia página JSP en cuyo interior contendrá enlaces, las propiedades y las invocaciones a la lógica de negocio.

Para la generación las propiedades que contiene una PaginaServidora, primero se procede a recorrer todos los elementos OperacionWeb que contiene. Existen dos tipos de OperacionWeb principalmente, aquellos cuyo tipoComportamiento ="Navegación" y por lo tanto consisten enlaces de la propia página, y aquellos cuyo tipoComportamiento = "Dominio" y por lo tanto, son invocaciones de operaciones mediante POST a la lógica de negocio. Cada uno de estas operaciones contiene un conjunto de elementos ParametroWeb que generarán cada uno de ellos componentes de entrada que permitirán al usuario introducir los valores desde el navegador.

```
mi.CompWebIM::mapearPaginaServidoraJ2EE (){
 var tipoComp = self.getTipoComponente()
 var tieneAtributos = tipoComp.getTieneAtributos()
 var tipoAtributos = tipoComp.tipoAtributos()
 file (self.getNombre().toLower() + extFichWeb)
 <% <%@ page language="java" %> %>
 //Se obtienen los componentes relacionados
 var comp = self.obtenerComponentesRelacionados()
 comp->forEach (c){
          <% <jsp:useBean id="%>c.getNombre()
          <%" class="%> dir_paquete + "."+
          self.getModulo()+ "."+ c.getNombre()
          <% scope="page"%>
          <% </jsp:useBean> %>
 }
 <% <html><head><title>%> self.getNombre() <%
          </title> %>
 <% <body> %>
 self.serviciosWeb->forEach (s:mi.OperacionWeb){
          // Enlaces a otras paginas
          if (s.getTipoComportamiento() = "navegacion")
   {
           var nodo =   s.getParamNodoOrigen()
            <a href="%> nodo.getNombre()+ extFichWeb
            <%"> %> nodo.getNombre() <%</a>%>
  }else{ //Invocaciones a la lógica
            <%form name="%> s.getNombre()
            <%" method="POST" action="ejecutar%>
          extFichWeb + s.getNombre()<%"%>
           s.parametrosWeb->forEach (p:mi.ParametroWeb)
                p.generarParametroPaginaServidoraJ2EE()
  } } }
```

Figura. 74. Regla de Texto mapearPaginaServidoraJ2EE

Siguiendo con la descripción de las reglas que transforman los componentes en texto, a continuación se muestra la regla que transforma a código un componente DatosEntidad para la plataforma .NET. concretamente en el lenguaje C#.

```
mi.CompWebIM::mapearDatosEntidadNET (tipoModulo:String){
 var tipoComp = self.getTipoComponente()
 var tieneAtributos = tipoComp.getTieneAtributos()
 var tipoAtributos = tipoComp.tipoAtributos()
 file (self.getNombre().toLower() + extFichWeb)
 <% using System;
  using System.Collections;
  namespace %> self.getModulo().getNombre()<%
  {%>
 <% public class %> self.getNombre() <%
  {%>
  //Se mapea a las propiedades del propio componente
  self.getPropiedades()->forEach
          (p:mi.PropiedadesWebIM){
    p.mapearDEntidadPropiedadNET()
  }
  //Se mapean los roles y sus metodos con otros
          componentes
  var comp = self.obtenerComponentesRelacionados()
  comp->forEach (c){
           c.mapearRolNET()
           c.mapearMetodosAccesoRolNET()
  }
  //Se mapean los servicios del componente
  self.getServicios()->forEach
          (s:mi.ServiciosWebIM){
            p.mapearDEntidadOperacionWebNET()
  }
```

8.6 Conclusiones

Se ha presentado en este capítulo los artefactos para la formalización y la trazabilidad entre modelos e implementación. Este nuevo tipo de artefacto introducido por la Ingeniería dirigida por Modelos se denomina transformación. WebSA a lo largo de su proceso define 2 transformaciones complejas denominadas T1 y T2, que automatizan los cambios de abstracción de análisis a diseño y de diseño a implementación respectivamente.

T1 es una transformación modelo a modelo, que para su definición requiere de un lenguaje de transformaciones que exprese la conversión de los elementos de un modelo en los elementos de otro modelo, todo ello desde sus metamodelos. Para ello, en un inicio se utilizó el lenguaje de transformaciones modelo a modelo estándar QVT. QVT permite expresar la transformación T1 de WebSA (Meliá & Gómez [71]), sin embargo, no permite materializarlas. Por ello, se ha optado por la definición de un nuevo lenguaje de transformaciones modelo a modelo denominado UPT (UML Profile for Tranformations) que redefine la forma de expresar las transformaciones por QVT, mejorando su expresividad y interoperabilidad, además de dar soporte para ejecutar las reglas de transformación.

T1 está constituida por 5 transformaciones cada una de las cuáles realiza un papel dentro del proceso de conversión de los modelos de análisis funcional y arquitectónico al modelo de integración. Concretamente de T1_1 a T1_3 son transformaciones que se centran en la creación de los módulos, comunicación entre los módulos y componentes independientes de funcionalidad. T1_4 crea los componentes que contienen información de dominio. Y T1_5 crea los componentes que contienen información de navegación. Las transformaciones contienen un conjunto de reglas de transformación que hace un total aproximadamente de 80 reglas. Es importante indicar, que el proceso de definición de transformaciones no es un proceso ya finalizado, sino que está en constante crecimiento e incluso que permite cierta personalización en aquellas reglas de transformación donde el usuario considere que está satisfecho por las transformaciones ya definidas.

Por su parte, la transformación T2 establece una conversión del modelo de integración a la implementación de la aplicación Web en las plataformas J2EE y .NET. Para su definición, se ha utilizado el lenguaje estándar MOFScript que realiza una lectura del metamodelo de WebSA y procede a ejecutar las reglas que escriben el código asociado. El proceso fundamental de T2 es el recorrido de los diferentes módulos que contiene el modelo de integración y dentro del módulo contiene un conjunto de componentes cada uno de los cuáles pertenece a un tipo determinado de componente definido por la tipología de WebSA. Entonces cada tipo de componente tiene establecido una regla de transformación en T2 que lo materializa en el código de una determinada plataforma.

En el siguiente capítulo se presenta la herramienta MDA que permite dar soporte al proceso de WebSA.

Volver al índice/Tornar a l'índex

CAPÍTULO 9

Una Herramienta MDA para Aplicaciones Web: WebTE

"The essence of Model Driven Architecture is its potential for automated model transformation."

[Anneke Kleppe]

9.1 Motivación

Según el estudio de Lewis & Wrage [66] las herramientas actuales de desarrollo basadas en MDA solamente implementan una parte de los conceptos que este estándar define. Destacando que casi todas se centran en generar código desde modelos lo cual es sólo uno de los aspectos de MDA. Sin embargo, estas herramientas fallan a la hora de adquirir ventajas sobre otros aspectos importantes como el uso de modelos y transformaciones que permitan ser compartidos con éxito entre las herramientas sin perder información.

Las herramientas MDA únicamente soportan el intercambio de modelos definidos mediante el estándar UML. Sin embargo, el nivel de interoperabilidad deseable requiere que las herramientas permitan intercambiar también transformaciones. Esto conduce a una nueva generación de herramientas que está todavía por definir. De hecho, la definición de transformaciones es específica de una herramienta, es decir, un desarrollador que necesite definir una transformación en una herramienta diferente debe hacerlo desde cero. QVT trata de cubrir esta carencia, sin embargo, presenta un problema fundamental, las principales herramientas de modelado Together 2006 [117], Racional XDE [99], etc. todavía no han incorporado a QVT Relations y a su notación gráfica como parte del conjunto de modelos que proporcionan. Únicamente Together presenta la posibilidad de definir QVT en su versión imperativa (QVT Operacional Mappings). Por lo tanto, QVT no es una solución al problema actual que presentan las transformaciones de modelos.

Otro aspecto importante a considerar en las herramientas basadas en MDA, es que requieren un gran esfuerzo inicial para la configuración y definición de las transformaciones para una plataforma. Este esfuerzo es compensado únicamente cuando las transformaciones son aplicadas a múltiples aplicaciones. Sin embargo, las plataformas evolucionan continuamente lo que requiere una sustitución de las transformaciones en algunos casos o la extensión de las transformaciones definidas. Esta capacidad de extensibilidad de las transformaciones debe ser accesible y sencilla para que sea realizada por los desarrolladores que han adquirido la herramienta. Sin embargo, en la mayoría de las herramientas MDA actuales es un aspecto complicado de abordar que únicamente se lleva a cabo por los propios desarrolladores de la herramienta, lo que exige en muchos casos la compra de módulos nuevos o incluso de una nueva versión de la herramienta.

En este trabajo, para la implementación de WebSA, se presenta la herramienta WebTE[28] que se basa en tres aspectos fundamentales:

1. El soporte para la especificación de los modelos y las transformaciones que propone WebSA actualmente.
2. La extensibilidad para que puedan añadirse y modificarse tanto los modelos como las transformaciones.
3. La interoperabilidad entre diferentes herramientas para que puedan definirse los modelos y las transformaciones desde diferentes herramientas.

Como se ha descrito en los capítulos previos, WebSA define sus modelos en UML, sus transformaciones modelo a modelo siguiendo el lenguaje UPT y las transformaciones modelo a texto con MOFScript. Sin embargo, WebTE no proporciona la interfaz gráfica para la definición de estos artefactos. WebTE es básicamente una aplicación Web que a través de las páginas Web permite la introducción tanto de los modelos como de las transformaciones mediante ficheros de texto que contendrán en algunos casos el estándar XMI que representa a los modelos y en otros casos a las plantillas (transformaciones de modelo a texto). Por lo tanto, esta herramienta permite definir los modelos desde cualquier herramienta UML comercial, alcanzando así una mayor interoperabilidad.

Seguidamente se especifica la arquitectura que configura de la herramienta WebTE.

9.2 Descripción de la Herramienta WebTE

La herramienta WebTE se ha implementado según los estándares proporcionados por la OMG (UML, XMI, MOF y OCL). Básicamente por dos motivos. Primero, por optimizar el esfuerzo de implementación, ya que permite la utilización de COTS (components off-the-shelf) como parsers, compiladores, clases y frameworks que son definidos para estos estándares (p.e JMI [52], MDR [79], etc.). Y segundo, por

[28] WebTE: WebSA Transformation Engine (acrónimo ingles de Motor de Transformaciones Web).

facilitar la interoperabilidad entre cualquier herramienta estándar que tenga soporte de UML y exportación al formato XMI.

WebTE es una aplicación Web que combina la gestión de la información almacenándola en una BBDD y además contiene la referencia a dos componentes que se encargan de realizar las tareas de transformación. Por un lado, el transformador UPT que se encarga de la implementación y ejecución de las transformaciones modelo a modelo. Por otro lado, el transformador a texto Velocity-Text implementa las reglas de transformación definidas en MOFScript mediante el uso de plantillas definidas mediante el framework Velocity [123].

A continuación se describe la aplicación WebTE utilizando los propios modelos definidos por la propuesta WebSA.

9.2.1 La Arquitectura de WebTE

Se realiza una descripción de la arquitectura general de WebTE comenzando por el Modelo de Subsistemas (ver capítulo 5) que proporciona la visión más global de la aplicación.

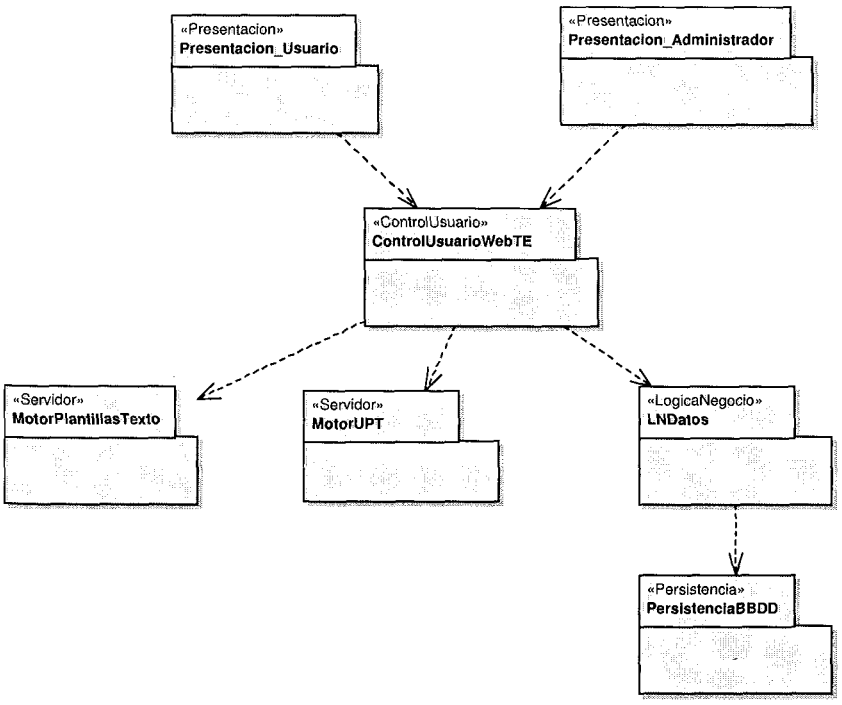

Figura. 75. Modelo de Subsistemas de WebTE

La Figura. 75 muestra la composición en diferentes elementos SubsistemaWeb que representa un estilo arquitectónico basado en capas para representar la aplicación WebTE. En ella, en la parte superior existen dos subsistemas de tipo Presentación: *Presentacion_Usuario* que está enfocado al usuario final que va a introducir los

modelos y algunas transformaciones nuevas para poder obtener el código de las aplicaciones a partir de los modelos. Y *Presentacion_Administrador* es la vista que proporciona el control absoluto sobre toda la funcionalidad que proporciona la herramienta. Ambas vistas de presentación, comparten un SubsistemaWeb de tipo ControlUsuario llamado *ControlUsuarioWebTE* que va a recibir todas las peticiones por parte del cliente Web, y las reenvía a los diferentes módulos encargados de realizar el procesamiento de servidor. El procesamiento de servidor está divido en 3 SubsistemasWeb: (1) *MotorPlantillaTexto*, es un SubsistemaWeb de tipo Servidor que recibe las plantillas nuevas por parte del cliente, y ejecuta dichas plantillas devolviendo el código final de la aplicación, (2) *MotorUPT*, es un SubsistemaWeb de tipo Servidor que recibe como entrada los XMI de los modelos y las transformaciones UPT, y devuelve otro XMI con el modelo obtenido. Y por último, (3) el *Subsistema LNDatos* de tipo LogicaNegocio que contiene la lógica de negocio que procesa los datos que el usuario va introduciendo a la aplicación, y los almacena el Subsistema *PersistenciaBBDD* que contiene los componentes necesarios para almacenar la información en una base de datos.

El segundo modelo de arquitectura de WebSA para describir la WebTE es el Modelo de Configuración (descrito en el capítulo 6) representado en la Figura. 76. El presente modelo MC representa la arquitectura de la aplicación WebTE mediante un estilo arquitectónico basado en componentes Web. En la parte superior de la Figura. 76 se aprecia como la aplicación proporciona como interfaz de usuario un AgenteUsuario llamado *Navegador* cuyo tipo de dispositivo es un NavegadorWeb. Este navegador muestra un conjunto de componentes *PaginasWeb* de tipo PaginaServidora. Los componentes *PaginasWeb* invocan o al controladorWeb llamado *Controladores* cuando tienen que invocar a uno de los servicios de la lógica o al componente *Façade* cuando quiere recuperar datos de modelo y mostrarlos. Respecto a la parte de lógica de negocio de la aplicación Web contiene los componentes necesarios para realizar las tareas almacenamiento y recuperación de la información en la BBDD, utilizando para ello el patrón *Façade* descrito en el capítulo 6 (ver Figura. 41) mediante el cual la aplicación Web está preparada para su funcionamiento en Internet con una alto número de clientes. Por otro lado, WebTE desde el componente *Controlador* invoca a tres componentes que representan los componentes que posibilitan la implementación de las transformaciones. Por un lado, el *MetamodeloJMI* que representa al metamodelo MOF y que dota de la posibilidad de cargar y generar ficheros XMI. Además, el controlador invoca a los dos componentes encargados de la implementación y ejecución de las transformaciones, son los componentes el *TransformadorUPT* y el *TransformadorTexto*. Estos componentes serán invocados en aquellas ocasiones donde se requiera de la realización de las transformaciones de modelo a modelo y de modelo a texto respectivamente.

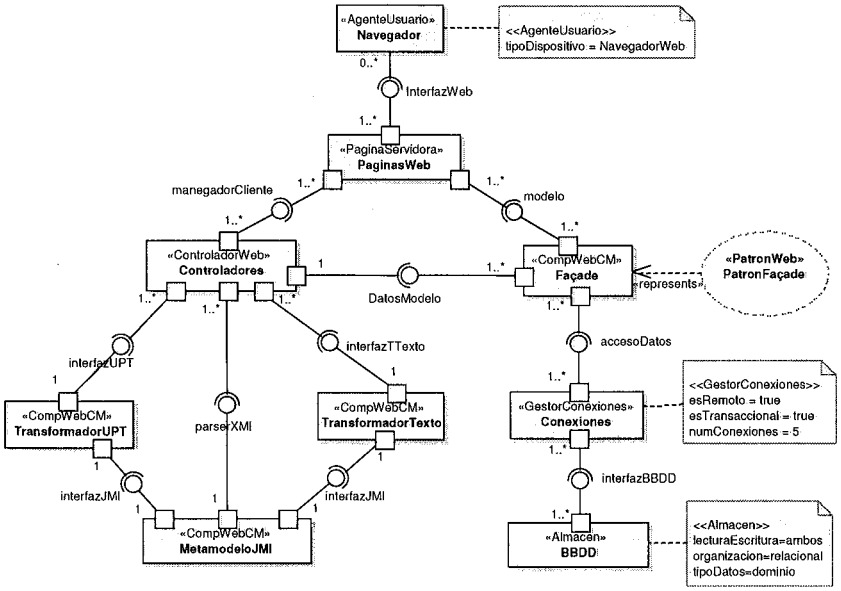

Figura. 76. Modelo de Configuración de WebTE

Por otro lado, el componente *MetamodeloJMI* ofrece a cada uno de los componentes de transformaciones una interfaz de consulta llamada *interfazJMI* que permite en todo momento consultar las instancias de las metaclases en JMI. Recordar que JMI implementa la interfaz MOF en Java (para repasar ver capítulo 3).

Una vez descrita la arquitectura general de WebTE, a continuación se procede a la descripción la funcionalidad que proporciona esta aplicación al usuario. Para ello, se muestra tanto el modelo de dominio como el modelo de navegación, utilizando para ello la notación proporcionada por la aproximación OOH.

9.2.2 La Funcionalidad de WebTE

Para realizar una descripción completa de la aplicación Web WebTE, es necesario especificar cual es la funcionalidad que proporciona. Para ello, se ha utilizado la aproximación OO-H [19]. El primer modelo que especifica la funcionalidad es el modelo de dominio. En la Figura. 77 se aprecian las clases y las relaciones que componen el modelo. Comienza la descripción con la clase Usuario que representa a la persona que se identifica en la aplicación y que contiene un conjunto de instancias *AplicacionWeb* que se dispone a definir y a transformar en una implementación final. La clase *AplicacionWeb* está asociada por un conjunto de instancias de la clase *Transformación*, y a su vez la *Transformación* puede pertenecer a más de una *AplicacionWeb*, esto permite reutilizar las transformaciones para más de un sistema. Además, la *AplicaciónWeb* tiene un conjunto de instancias *Modelo* que representa a los modelos que definen dicha Aplicación tanto los de entrada como los obtenidos por las transformaciones.

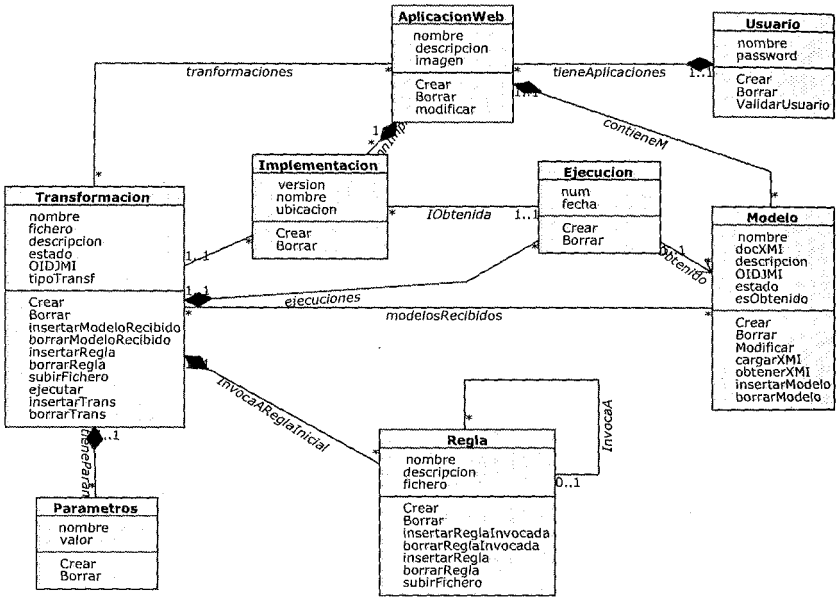

Figura. 77. Modelo de Dominio de WebTE

La *AplicaciónWeb* contiene también a conjunto de instancias *Implementación* que serán obtenidas como resultado de las transformaciones de texto. La siguiente clase de importancia para el sistema es *Transformación* que contiene los atributos y operaciones necesarios para introducir una transformación en forma de un fichero de texto, ejecutar dicha transformación, y asociarle los servicios o implementaciones obtenidas de su ejecución. Una *Transformación* de cuyo atributo *tipoTransf="Texto"* contiene un conjunto de *Reglas* que serán mantenidas en pos de un mantenimiento de los ficheros que están asociados a cada una de estas reglas. Además, una *Transformación* de tipo texto contiene un conjunto de *Parámetros* necesarios para realizar la correspondiente transformación a texto.

Otra clase de gran importancia para el sistema WebTE es *Modelo*, la cual contiene los atributos necesarios para el mantenimiento de los valores del *Modelo* tanto a nivel de fichero como en el repositorio *JMI*, del cual contendrá una referencia *OIDJMI* que identifica unívocamente al *Modelo*. Un *Modelo* tendrá a su vez la funcionalidad de realizar una *cargaXMI* desde un fichero y obtenerXMI para obtener el fichero desde el repositorio JMI. Además, un *Modelo* está asociado con un conjunto de instancias *Transformación* y viceversa, con lo que el *Modelo* puede utilizarse para más de una *Transformación*.

Como mecanismo para almacenar una historia de las ejecuciones de las transformaciones, se define la clase *Ejecución*. En función del resultado obtenido de la *Ejecución*, puede tener asociado un *Modelo* (si es una transformación modelo a modelo) o una *Implementación* (si es una transformación modelo a texto). Además, almacena la hora y el número que identifica la *Ejecución*.

La última clase es Implementación, que referencia al código obtenido a partir de una *Transformación*. La clase *Implementación* principalmente almacena el número de versión, un nombre que puede indicar el usuario y la ubicación donde se ha situado dentro del servidor.

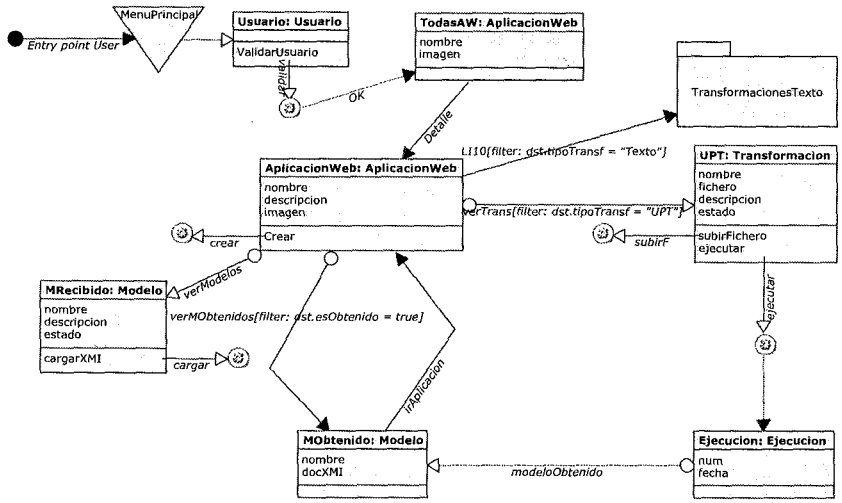

Figura. 78. Modelo de Navegación Principal de WebTE

El siguiente modelo funcional es el modelo de navegación que permite representar la interfaz hipermedial que proporciona la aplicación WebTE. El modelo de navegación de la herramienta está divido en 2 diagramas. El diagrama de navegación principal donde se sitúa el inicio de la aplicación Web. La Figura. 78 muestra el inicio de la aplicación Web en una página de inicio en la cual se solicita la introducción del *nombre* y el *password* del usuario que desea utilizar los servicios de la herramienta. Si el usuario está correctamente validado comienza la introducción de los elementos mediante el acceso a las aplicaciones Web del usuario. El usuario puede crear una nueva aplicación Web invocando a la operación *crear*. Desde la clase navegacional *AplicacionWeb* se puede navegar a los modelos recibidos (*MRecibidos)* como entrada por el usuario (*modelo.esObtenido = false*), además el usuario puede proceder a la carga de nuevos modelos invocando enlace de servicio *cargarXMI*.

En el diagrama mostrado en la Figura. 78 la navegación principal asociada a las transformaciones modelo a modelo. Para realizarla, el usuario desde el nodo *AplicacionWeb* puede navegar para visualizar las *Transformaciones* llamada *UPT*, para ello se filtran aquellas transformaciones cuyo *tipoTransf = "UPT"*. A continuación, el usuario puede invocar subir una nueva transformación invocando a la operación *subirFichero*, o lanzar la operación *ejecutar*, con el cual creará un nuevo objeto de la clase *Ejecución*, que tendrá asociado a su vez uno o más instancias de *Modelo* obtenidas por la ejecución de la transformación.

Si el usuario desea lanzar transformaciones de texto, desde el nodo *AplicaciónWeb* navega por el enlace dirigido al destino navegacional *TransformacionesTexto* y cuyo contenido se muestra en la Figura. 79

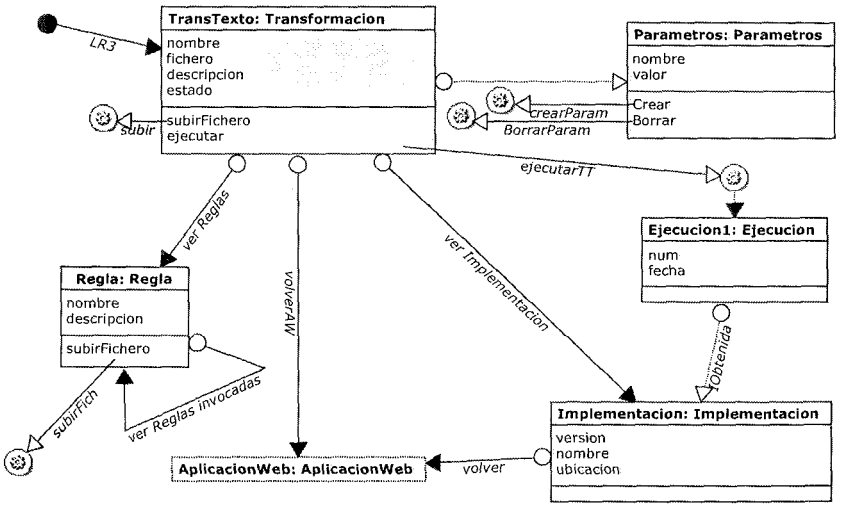

Figura. 79. Modelo de Navegación de las Transformaciones de Texto

La navegación para las introducción de las transformaciones de texto comienza por un punto de entrada situado en el nodo *TransTexto*, el cual muestra los atributos de la transformación de texto, el esto y oferta dos operaciones que pueden ser invocadas desde la página. Además, en la misma página se muestran los parámetros asociados a la transformación. El nodo *Parámetro* permite *Crear* un nuevo Parámetro o *Borrar* un parámetro. Por otro lado, el usuario desde la transformación puede invocar a *subirFichero* que cargaría el fichero de la transformación de texto, o puede invocar a *ejecutar* que crearía una instancia *Ejecución* que tiene asociada la implementación obtenida a partir de la transformación. La *Implementación* muestra la versión, el *nombre* y la *ubicación* donde está situada.

Otra tarea que puede realizarse sobre las transformaciones es la edición de cada una de sus reglas, para ello desde el nodo *TransTexto* se puede navegar a las reglas que dicha transformación contiene. Una vez situados sobre una regla, puede ser actualizada mediante la carga del fichero que contiene los datos de la regla, o se puede navegar a las reglas que son invocadas desde ella.

Una vez se ha terminado el trabajo con las transformaciones de texto, desde *TransTexto* y desde *Implementación* puede volverse a la página de la *AplicaciónWeb*.

Seguidamente se muestra alguna de las páginas Web que constituye la interfaz de usuario proporcionada por la aplicación WebTE.

9.2.3 La Interfaz de Usuario de WebTE

Para terminar la descripción de la herramienta se procede a describir las pantallas más importantes de la aplicación. La primera pantalla que se presenta en la Figura. 80 es la página de inicio de WebTE, donde se le solicita al usuario que introduzca su login[29] y su password.

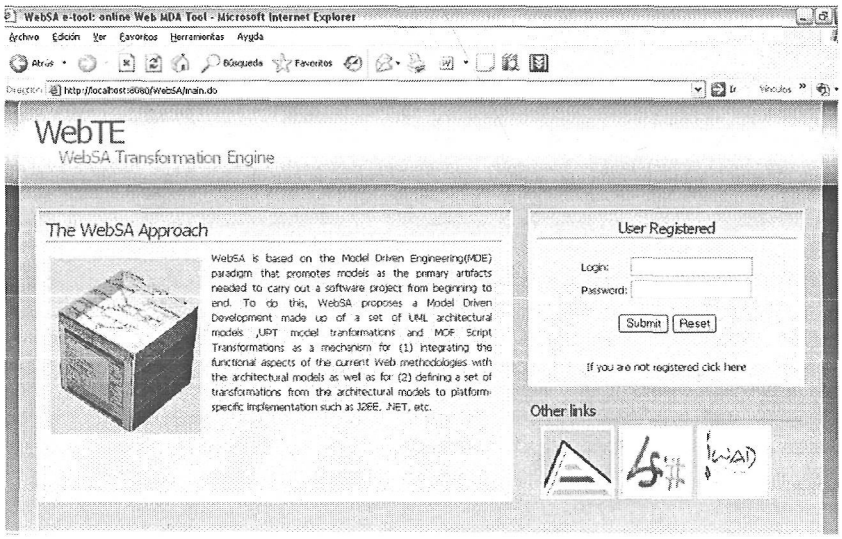

Figura. 80. Pagina Inicio de WebTE

En caso de ser incorrecto le muestra un mensaje de error y continua en la misma página, si es correcto entonces procede a mostrar la siguiente página donde se presentan las diferentes aplicaciones Web que han sido introducidas por el usuario para su desarrollo mediante la aproximación WebSA (Ver Figura. 81)

[29] Login: Palabra inglesa que indica la introducción de la Identificación del usuario.

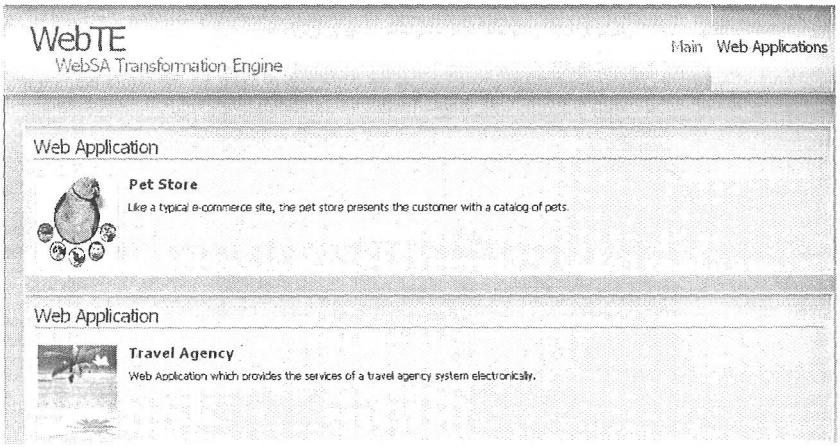

Figura. 81. Página de Aplicaciones Web de WebTE

Cada una de las aplicaciones Web muestra una imagen que la identifica, junto con un nombre y una breve descripción, en la Figura. 81 se muestran las aplicaciones Petstore y Travel Agency que han servido de ejemplo en la presente tesis.

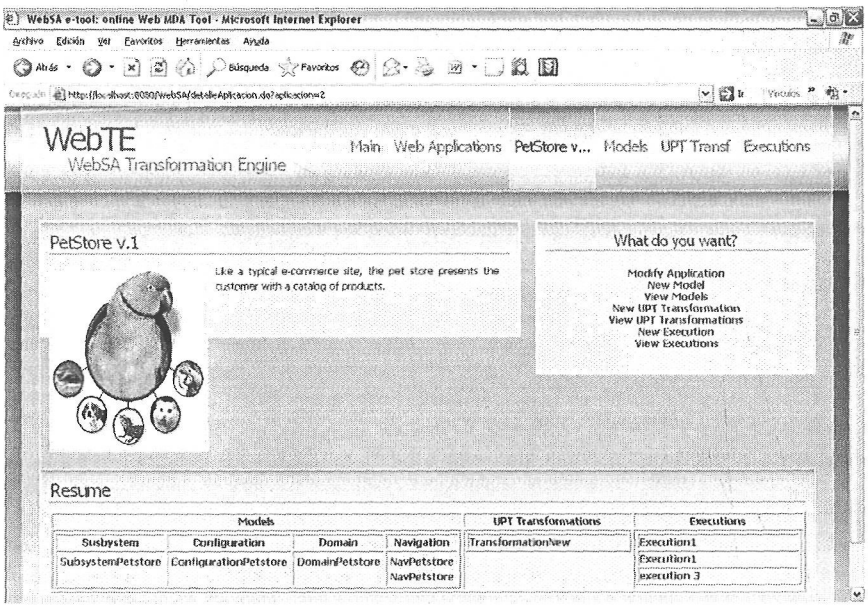

Figura. 82. Pagina Principal de Petstore

A continuación, el usuario selecciona una de las aplicaciones Web, mostrándose una página específica para cada aplicación, en la Figura. 83 muestra la pantalla donde el usuario ha seleccionado la aplicación Petstore. Navega entonces a la página específica para la aplicación Petstore donde aparece una imagen más grande de la aplicación, una descripción y un resumen de los modelos, transformaciones y ejecuciones de las transformaciones han sido almacenadas hasta ahora.

Además, en el lado derecho puede apreciarse una tabla donde se indican las diferentes acciones que el usuario puede realizar como: *Modify Application* (Modificar la aplicación), *New Model* (nuevo modelo), *View models* (ver los modelos), *new UPT Transformation* (ver las transformaciones UPT), *view UPT Transformations* (ver las transformaciones UPT), *New Execution* (nueva ejecución de las transformaciones), *View Execution* (ver las ejecuciones de las transformaciones).

Seguidamente se realiza un ciclo completo y el usuario introduce todos los artefactos asociados con la aplicación Web. El siguiente paso es la introducción de los modelos de entrada asociados con las transformaciones. Para ello, se puede ir a la página de los modelos donde se introducen nuevos modelos para los diferentes tipos que obtiene la herramienta WebTE (modelo de subsistema, modelo de configuración, modelo de dominio y modelo de navegación de OOH).

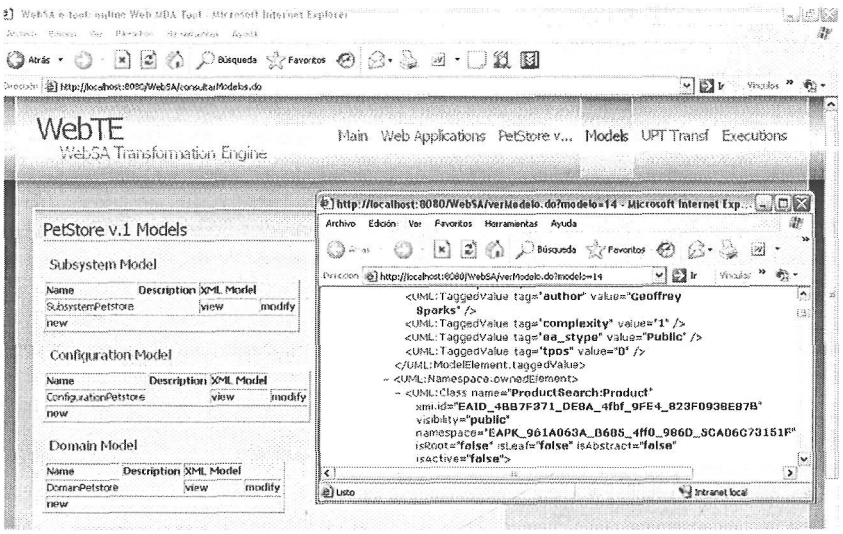

Figura. 83. Pagina de los Modelos de Petstore

Cada uno de estos modelos es subido al sistema mediante un upload de un fichero XMI que el usuario previamente ha modelado en una herramienta UML. Una vez introducidos los ficheros se pulsa el enlace new situado debajo del último modelo introducido, y se lanza la operación *cargarXMI* para cada uno de los modelos. Entonces los ficheros son subidos al metamodelo JMI desde donde la aplicación WebTE podrá consultarlos para realizar las siguientes transformaciones. Como se aprecia en la Figura. 83, a continuación se puede dar al enlace *view* (ver) que permite

consultar el XMI del modelo o al enlace *modify* (modificar) que permite modificar el modelo introducido en el sistema.

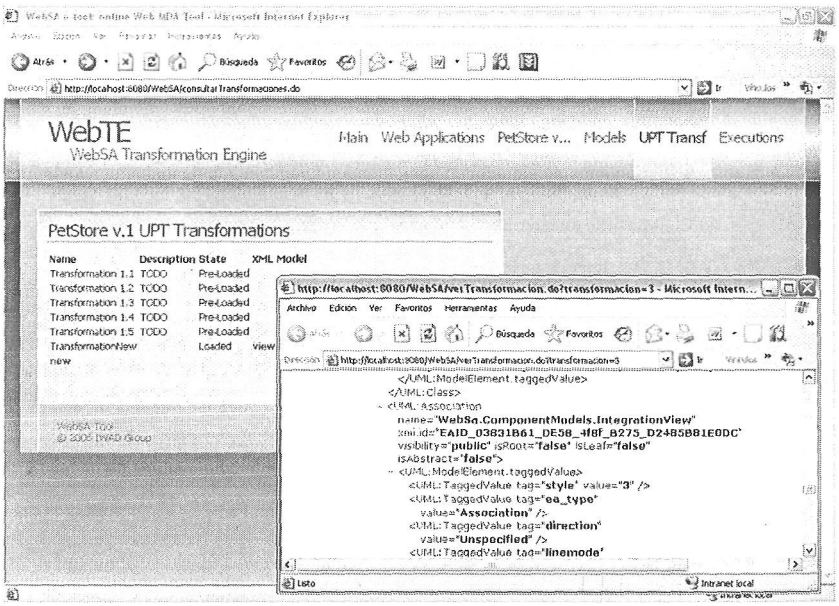

Figura. 84. Pagina de las Transformaciones UPT de Petstore

La Figura. 84 muestra la página de las transformaciones UPT modelo a modelo que están asociadas a la aplicación Petstore. En la izquierda de la página se muestran las transformaciones que constituyen la transformación T1 de WebSA y el estado en el que se encuentran, indicando con Pre-Loaded que se encuentran cargadas en el repositorio JMI. El usuario puede decidir por dos posibles servicios, por un lado puede cargar más transformaciones en este caso que sean específicas de la aplicación Petstore. Estas transformaciones extienden la transformación T1 introduciendo características específicas. Por ejemplo, se supone que se desea especificar que la clase Carrito de Petstore no tenga persistencia en BBDD y solo se almacene en Sesión Web. Para ello, se introduce una transformación que cuando encuentre la clase carrito no genere los componentes necesarios para dotarle de persistencia.

Una vez introducidas las transformaciones que el usuario considere conveniente, se procede a la ejecución de la transformación T1 con lo que se obtiene como salida el modelo de integración (ver capítulo 7). La aplicación WebTE navega hasta la página de las ejecuciones de la transformación T1 (ver Figura. 85) donde se muestra el resultado que serán diferentes modelos de integración en formato XMI que son obtenidos a partir del metamodelo JMI. En la Pagina de ejecución se muestran las diferentes ejecuciones realizadas mediante enlaces, en este caso solamente contiene una ejecución. Si se pulsa el enlace aparece una ventana flotante que muestra el XML del fichero XMI que representa al modelo de integración.

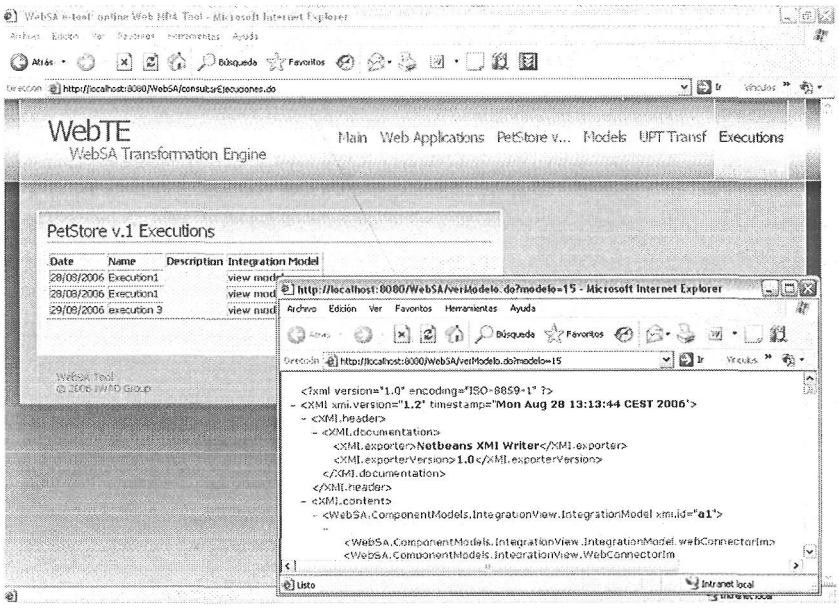

Figura. 85. Página de Ejecución de Transformación T1

Una vez ejecutada la transformación T1, el siguiente paso a realizar por la aplicación WebTE es la introducción y ejecución de las transformación de modelo a texto T2 (ver Sección 8.5) que permite convertir el modelo de integración en la implementación de la aplicación. Primero comenzamos con la pagina de creación de la transformación de texto en la que el usuario elige el nombre de la transformación y establece cual será la plataforma elegida entre J2EE y .NET. Una vez elegida se crearán un conjunto de reglas que constituyen en núcleo fundamental de la conversión a la plataforma y que se generan de forma específica para cada uno de los usuarios. Esto permite que un usuario pueda modificar las reglas introduciendo características propias a su generación e incluso añadiendo llamadas a otras reglas que el mismo puede introducir.

En la Figura. 86 se aprecia una tabla en la que se muestran las diferentes reglas que contiene la transformación T2 llamada por el usuario *TransformationJAVA.* En este caso, las reglas que aparecen son las generadas por defecto o aquellas que el usuario ha introducido. Para cada una de ellas, el usuario puede visualizarlas dándole al enlace *Template,* o modificarlas (*modify)* o borrarlas (*delete)*.

.

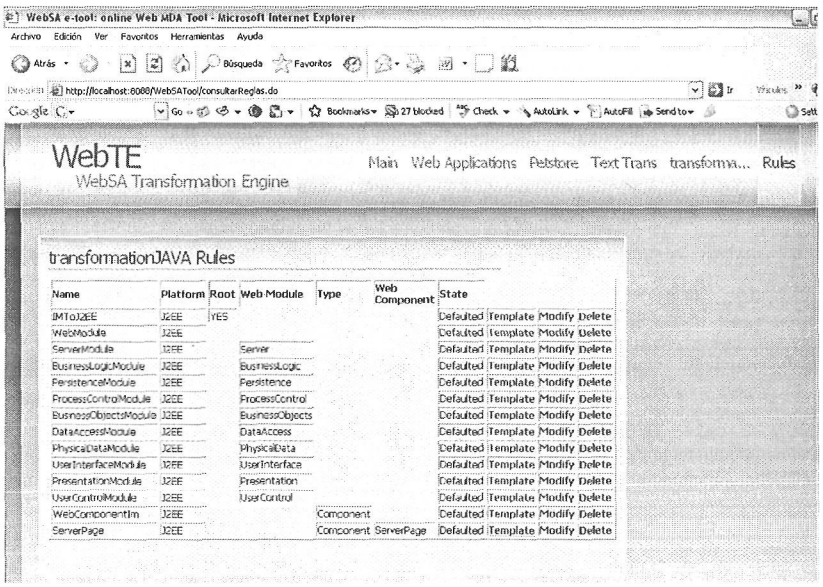

Figura. 86. Ejemplo de Reglas de una Tranformación de Texto T2

Una vez introducidas las reglas que el usuario considere oportuno, a continuación procede a la ejecución de la transformación T2 que tendrá como resultado una implementación con el código de la aplicaciónWeb. En la Figura. 87 puede apreciarse como la aplicación muestra en la izquierda las diferentes implementaciones obtenidas a partir de la ejecución de la transformación T2. Para obtener cada una de ellas, el usuario pulsa sobre el enlace implementación y le aparece un diálogo que le indica que se le devuelve en un fichero .zip el conjunto de ficheros generados. En el caso de que la generación no hubiese sido del agrado del usuario, comenzaría un proceso de modificación de las reglas de modelo a modelo y modelo a texto para obtener así un resultado más satisfactorio.

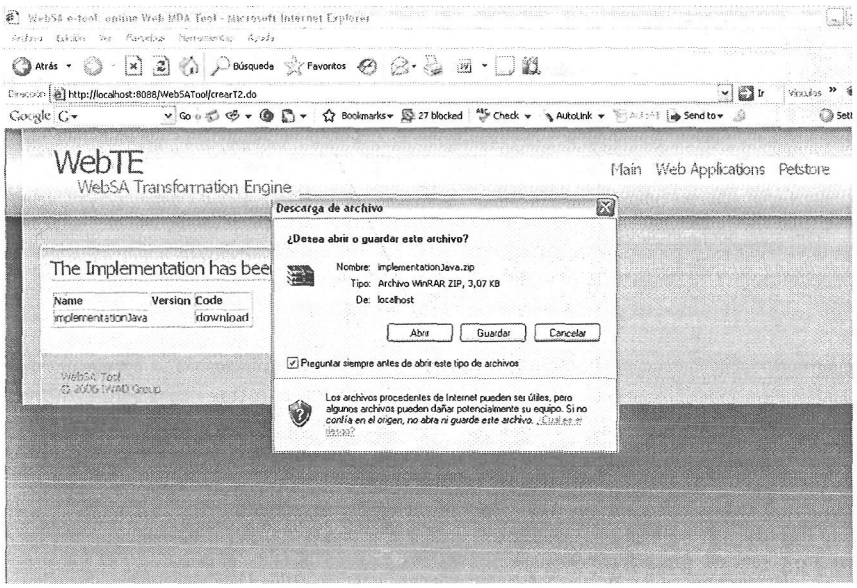

Figura. 87. Página para la Obtención de la Implementación

En las siguientes secciones se profundizan en los dos módulos que permiten la generación y ejecución de las transformaciones basadas en metamodelos. La siguiente sección comienza con el módulo de transformaciones.

9.3 Módulo de Transformaciones UPT

Como se ha especificado en la arquitectura de la aplicación WebTE, concretamente en el modelo de configuración (ver Figura. 76) existen tres componentes principales que posibilitan la realización de las transformaciones, de los cuáles únicamente dos son los encargados de las transformaciones en UPT. Por un lado, el MetamodeloJMI que consiste en la representación del metamodelo de WebSA y UPT en Java. Además, proporciona la interfaz necesaria para consultar e instanciar las metaclases y dotar de la posibilidad de cargar y exportar modelos expresados mediante el estándar XMI. Por otro lado, el Transformador UPT proporciona por un lado un generador de código de las transformaciones UPT en Java, cuya implementación consiste básicamente en la consulta de las metaclases en los metamodelos origen y la creación de las nuevas instancias de metaclases en el metamodelo destino. Posteriormente, una vez creado el código se encarga de su compilación y posterior ejecución.

Seguidamente se especifica cual es la estructura interna que constituye al transformador UPT.

9.3.1 Estructura Interna del Transformador UPT

Para entender mejor como funciona el componente TransformadorUPT internamente es necesario especificar su estructura interna y su comportamiento. Y para ello, se muestra tanto la parte estática mediante el modelo de estructura compuesta de UML 2.0 [95] y como la parte dinámica mediante el modelo de secuencia.

Comenzando por la descripción de la parte estática del TransformadorUPT, en Figura. 88 se describe mediante el modelo de estructura compuesta cuáles son los componentes que lo configuran y sus relaciones. Para ello, en la parte superior se muestra la configuración de TransformadorUPT y los componentes que contiene son representados mediante Partes y en la parte inferior se representan cada uno de los componentes con sus atributos, operaciones y puertos. Comenzando la descripción por la parte superior, el componente punto de acceso al TransformadorUPT es *Ejecución*. Ejecución es el encargado de llevar el hilo conductor de la ejecución de las transformaciones y es el punto de acceso con el exterior. Ejecución oferta la *InterfazUPT* para que sea invocado el TransformadorUPT por el cliente, y además invoca a *InterfazJMI* comunicándose con el metamodelo JMI-MOF para poder obtener su información. Además, el componente Ejecución como se aprecia en la parte inferior almacena los modelos requeridos y obtenidos para la transformación, además de las transformaciones que se van a invocar en una determinada ejecución. El componente Ejecución invoca a su vez dos tipos de lectores *LectorModeloUML* y *LectorTransformacionUPT* que se encargan de leer un ficheroXMI y transformarlo en las clases JMI que los representan para que puedan ser consultados posteriormente. Cuando son conocidas las transformaciones el componente Ejecución invoca al componente *GenerarCodigoUPT* que se encarga de generar el código java a partir de las transformaciones UPT cargadas en el metamodelo UPT, contando además con un compilador de OCL que traduzca las expresiones OCL de las transformaciones. Por último, el componente Ejecución se comunica con un conjunto de componentes *TransformacionUPTJava* los cuáles contienen la implementación Java de UPT encargada de consultar y escribir los metamodelos JMI que indique la transformación.

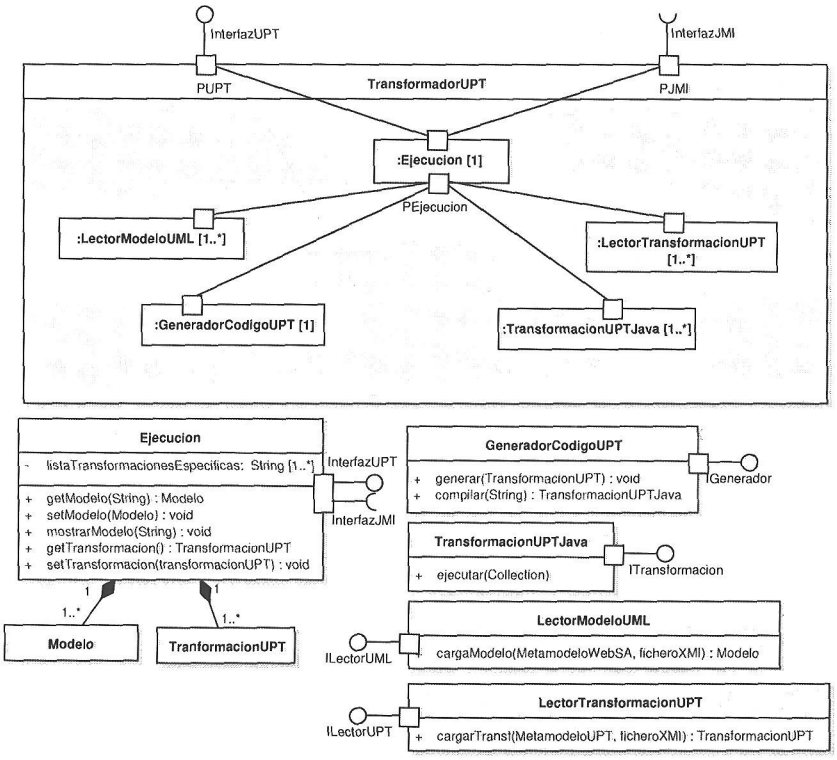

Figura. 88. Modelo de Estructura Compuesta del Transformador UPT

En la Figura. 89 se describe el comportamiento dinámico de los componentes internos del TransformadorUPT mediante un modelo de secuencia en el que participan las diferentes partes descritas en el modelo de estructura compuesta. El inicio del proceso de transformación UPT comienza desde el componente Ejecución en el cual se establece un bucle llamado *Carga Modelos* donde se cargan todos los modelos origen de la transformación a partir del XMI y se almacenan en el componente Ejecución. Seguidamente comienza el bucle *Ejecución Transformaciones* en el cual se realizarán todas las transformaciones UPT requeridas por el usuario. El primer paso es el envío del mensaje cargarTransf al LectorTranformacionesUPT que devolverá una TransformaciónUPT que será almacenada en el objeto Ejecución mediante el mensaje setTransformacion. A continuación, se invoca a generar y compilar del componente GenerarCodigoUPT pasándole los objetos TransformacionUPT. Este devolverá la clase TransformacionUPTJava que la implementa. Finalmente, a TransformacionUPTJava se le envía el mensaje generar y devolverá el modelo destino, que en el caso de WebSA es el modelo Integración. Por último el componente Ejecución almacena el modelo de Integración y posteriormente lo muestra a la interfaz de usuario.

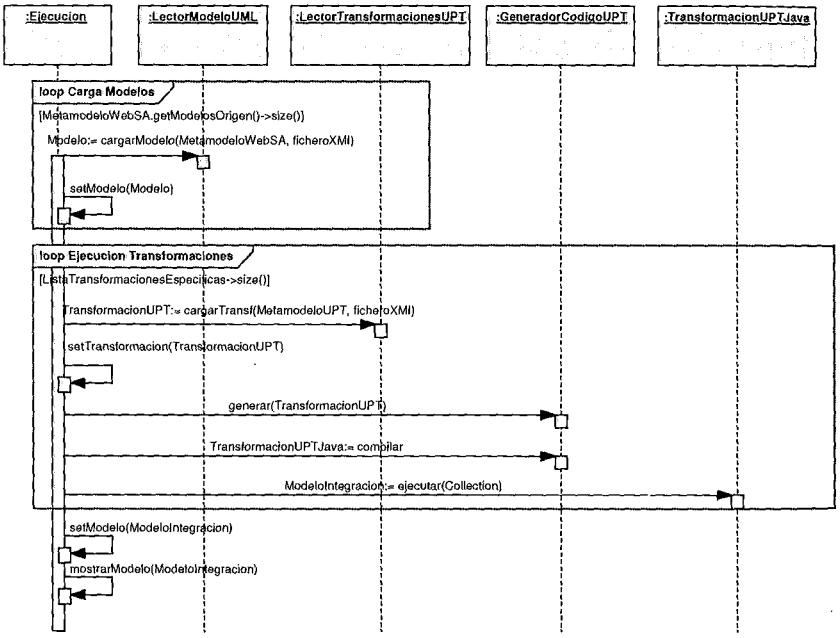

Figura. 89. Modelo de Secuencia del Transformador UPT

A continuación, se especifica cual es el proceso de creación y ejecución de las transformaciones UPT en la herramienta.

9.3.2 El Proceso de Implementación y Ejecución del Transformador UPT

El modo de proceder del Transformador UPT es el siguiente: se especifican los modelos de WebSA y transformaciones desde alguna herramienta UML que los exporte a ficheros en formato XMI. Las representaciones XMI de los modelos y de las transformaciones sirven de punto de entrada para que los cargue en el MetamodeloJMI. A partir de aquí, comienza el proceso de implementación y ejecución de las transformaciones por parte del Transformador UPT que conducirá a la generación XMI del modelo destino.

A continuación se describe paso a paso el proceso de implementación y ejecución que se sigue para las transformaciones UPT. La Figura. 90 describe de forma genérica cual es este proceso, ya que no se centra en los modelos que se especifican en WebSA sino que puede ser utilizado por cualquier aproximación. Con esta manera, no se descarta que en un futuro el Transformador UPT pueda ser utilizado por otras aproximaciones MDA que utilicen transformaciones basadas en metamodelos.

La descripción se realiza de forma gradual coincidiendo cada uno de los pasos con los números que aparecen en la Figura. 90:

1. Se definen los metamodelos MOF que sean el origen y el destino de las transformaciones a definir. Esta tarea se realiza utilizando una herramienta UML que permita la conversión de los metamodelos a XMI para MOF. Los

metamodelos definidos en XMI se convierten en un metamodelo JMI (Java Metamodel Interface) [97]. JMI permite dos cosas (1) el manejo de las instancias de las clases de un metamodelo MOF, es decir, permite manejar los modelos que son complacientes con los metamodelos definidos, (2) proporciona los parsers para la lectura y escritura desde XMI-UML que representen a los modelos. Este es un paso semiautomático, proporcionado por el framework MDR de Netbeans [79].

2. Se definen los modelos UML siguiendo la notación para las transformaciones UPT desde cualquier herramienta UML que tenga la generación en XMI para UML y se obtengan los ficheros XMI correspondientes. La herramienta procede a la carga de los ficheros XMI y crea las instancias en el metamodelo JMI de UPT.
3. A partir del metamodelo de UPT se obtienen las transformaciones en código Java y se compilan. Aquí termina la fase de definición de metamodelos y transformaciones en la herramienta. Ahora comienza el proceso de ejecución de las transformaciones UPT.

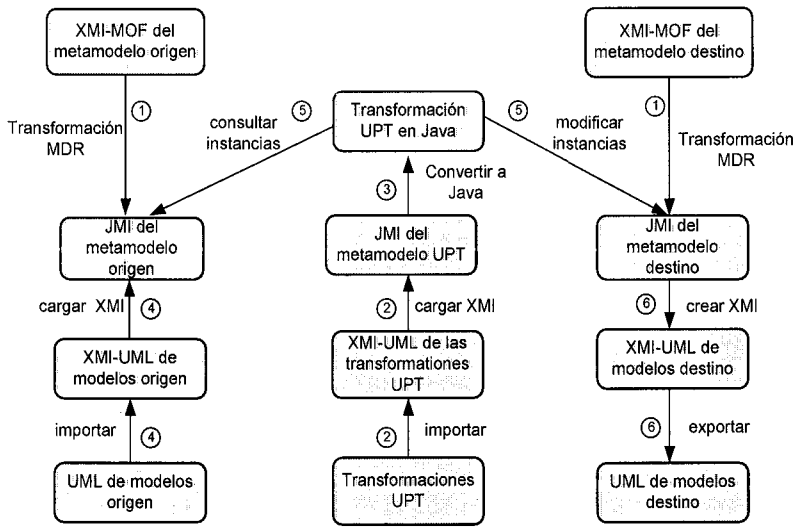

Figura. 90. El Proceso de Implementación y Ejecución del Transformador UPT

4. Se procede a la definición de los modelos origen en una herramienta UML, a partir de la cual se generarán los XMI para UML. El XMI será leído por los parsers para JMI, y de esta manera nuestros metamodelos en JMI serán cargados con los modelos introducidos.
5. Se ejecutan las transformaciones para convertir las instancias del metamodelo JMI origen en instancias en el metamodelo JMI destino.
6. Por último, se procede a convertir las instancias del metamodelo destino en XMI mediante los escritores que JMI proporciona para XMI.

En su estado actual, el transformador UPT presenta dos restricciones: (1) los metamodelos origen y destino de las transformaciones deben estar definidos en MOF.

(2) Los modelos origen de las transformaciones deben poder expresarse en una herramienta que los exporte a XMI.

9.4 Módulo de Transformaciones de Modelo a Texto

Dentro de las funcionalidades proporcionadas por la herramienta, la transformación de modelo a texto es fundamental para la obtención del código final de la aplicación. Esta tarea es desarrollada por el módulo llamado MotorPlantillasTexto (ver Figura. 75) que contiene básicamente 2 componentes principales descritos en el modelo de configuración (ver Figura. 76): (1) El MetamodeloJMI que realiza la tarea de cargar el metamodelo origen de la transformación el cual es obtenido a partir de la ejecución de la transformación T1, y además dota de una interfaz que permite consultar las instancias de las metaclases del metamodelo. (2) El TranformadorTexto que consiste en un entorno de ejecución de plantillas Velocity [123] que implementan las transformaciones modelo a texto que han sido especificadas mediante el estándar MOFScript.

Concretamete el componente TransformadorTexto está basado en la implementación proporcionada por el framework Texen [113], que consiste en una utilidad de proposito general para la generación de cualquier tipo de texto basada en las plantillas de Velocity. Texen permite la definición de dos tipos de plantillas:

- Plantillas Control: que establecen el hilo conductor de la generación e invocan a otras plantillas para su ejecución.
- Plantillas Trabajadoras: estas plantillas serían las hojas de la generación y que simplemente se encargar de generar texto.

Además, Texen tiene la posibilidad de definir un conjunto de variables de contexto que pueden ser utilizadas a través de todas las plantillas de Velocity.

El modo de proceder del TransformadorTexto es el siguiente: se especifican los modelos que sean origen de las transformaciones desde una herramienta UML que los exporte a ficheros en formato XMI. Se introducen las transformaciones de texto mediante plantillas que implementan las reglas de transformación que se han especificado en MOFScript pero adaptándolas a la notación de Velocity. A partir de aquí, comienza el proceso de implementación y ejecución de las transformaciones por parte del Transformador de Texto que conducirá a la generación de la implementación del modelo destino.

A continuación se describe paso a paso el proceso de implementación y ejecución que se sigue para las transformaciones de texto. La Figura. 91 describe de forma genérica cual es este proceso, ya que no se centra en los modelos que se especifican en WebSA sino que puede ser utilizado por cualquier aproximación. De esta manera, no se descarta que en un futuro el Transformador de Texto pueda ser utilizado por otras aproximaciones MDA que utilicen transformaciones de texto basadas en metamodelos MOF.

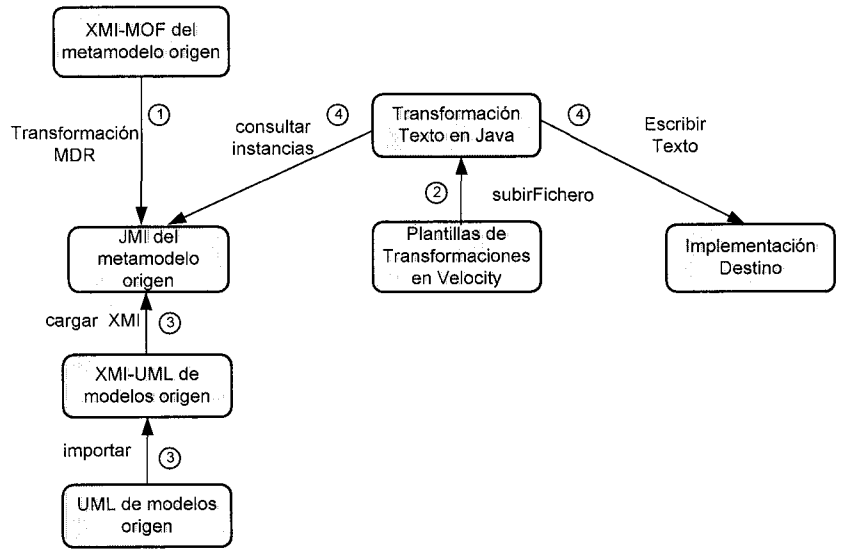

Figura. 91. Proceso de Implementación y Ejecución de Transformaciones de Texto

La descripción se realiza de forma gradual coincidiendo cada uno de los pasos con los números que aparecen en la Figura. 91:

1. Se definen los metamodelos MOF que sean el origen de las transformaciones a definir. Esta tarea se realiza utilizando una herramienta UML que permita la conversión de los metamodelos a XMI para MOF. Los metamodelos definidos en XMI se convierten en un metamodelo JMI. Este es un paso semiautomático realizado por el framework MDR de Netbeans.
2. Se definen las plantillas en Velocity que representan las diferentes reglas de las transformaciones de texto. Estas plantillas son subidas al motor de transformaciones para que puedan ser ejecutadas. Ahora ya puede comenzar el proceso de ejecución de las transformaciones de Texto.
3. Se procede a la definición de los modelos origen en una herramienta UML a partir de la cual se generarán los XMI para UML. El XMI será leído por los parsers para JMI, y de esta manera nuestros metamodelos en JMI serán cargados con los modelos introducidos.
4. Se ejecutan las transformaciones de texto para convertir las instancias del metamodelo JMI origen en la implementación destino.

9.5 Conclusiones

En este capítulo se presenta la especificación de la herramienta que posibilita la materialización de nuestra aproximación, que se ha denominado WebSA Transformation Engine (WebTE). WebTE se centra en dar soporte a la carga de los metamodelos de modelos y transformaciones que propone WebSA, dotar de extensibilidad a las transformaciones para que puedan añadirse y/o modificarse y

proporcionar de interoperabilidad entre diferentes herramientas UML para que puedan definirse e intercambiarse modelos y las transformaciones.

Para una descripción más completa, se definen los modelos que representan la herramienta WebTE mediante WebSA. Es decir, los modelos funcionales de dominio y navegación, y los modelos de arquitectura MS y MC. Además, se presentan la interfaz de usuario formada por las páginas Web que permiten la gestión con la aplicación.

Sin embargo, la parte más compleja de la aplicación son los dos módulos de transformaciones: el motor de transformaciones UPT y el motor de plantillas. Por un lado, el motor de transformaciones modelo a modelo definido para el lenguaje UPT, permite la especificación de las transformaciones mediante un modelo grafico UML que es introducido en la herramienta mediante XMI. Por otro lado, el motor de plantillas se basa en el lenguaje Velócity, permite especificar las transformaciones modelo a texto utilizando una notación similar a la proporcionada por MOFScript.

Es importante destacar que el trabajo sobre la herramienta no ha finalizado. Se han de mejorar aspectos de automatización del proceso de conversión de XMI-MOF a JMI, lo que permitiría que rápidamente introducir metamodelos de otras aproximaciones funcionales como UWE, WebML, W2000, etc.

Volver al índice/Tornar a l'índex

PARTE IV

Conclusiones y Apéndices

La parte final de la tesis muestra las conclusiones del presente trabajo, sus ventajas, desventajas y los trabajos futuros. Finalmente, se muestran los cinco apéndices donde se sitúan los aspectos más técnicos del presente trabajo.

CAPÍTULO 10

Conclusiones y Trabajos Futuros

10.1 Conclusiones Finales

Es un hecho comúnmente aceptado que las aplicaciones Web poseen un conjunto de rasgos diferenciadores que las distinguen del resto de aplicaciones software. Entre ellos destacan la necesidad de una puesta en marcha veloz, su evolución orgánica que requiere de procesos ágiles de mantenimiento, y el uso de interfaces donde la navegación es un aspecto preponderante.

Sin embargo, la propia evolución de las aplicaciones Web en los últimos años ha hecho necesario tratar otros aspectos no funcionales como son la utilización de componentes distribuidos en la parte servidora, la necesidad de conectar con sistemas legados o la necesidad de dotar de escalabilidad para dar soporte a un mayor número de clientes. Esto ha hecho necesario replantear los procesos de desarrollo e introducir nuevos artefactos que ayuden a la captura de estos nuevos aspectos.

En este contexto, esta tesis ha presentado una nueva propuesta de desarrollo Web llamada WebSA que pretende ser continuista con los trabajos funcionales definidos hasta ahora, pero que ha intentando mejorar sus procesos de desarrollo mediante la introducción de nuevos artefactos para el tratamiento específico de la arquitectura software que complementan al conjunto de artefactos propuestos por cualquier aproximación funcional Web. Además, estos artefactos se han integrado en un proceso dirigido por modelos que permite establecer una trazabilidad completa y obtener una aceleración del proceso mediante el uso de las transformaciones de modelo y de texto.

Para la consecución de sus objetivos, WebSA ha partido de trabajos previos provenientes de la investigación en las siguientes disciplinas: la Ingeniería Web, la Arquitectura del Software para Web y el desarrollo dirigido por Modelos.

A continuación se resumen las principales características que distinguen a WebSA de otras aproximaciones de modelado hipermedial, sus principales aportaciones al campo de la Ingeniería Web y las ventajas que proporciona su uso.

10.1.1 Características de WebSA

WebSA posee las siguientes características diferenciadoras:

1. El proceso de desarrollo de WebSA es una instancia del proceso de desarrollo unificado (UP) [50], donde las fases y workflows se han adaptados al desarrollo Web. Este proceso está además dirigido por la Arquitectura del Software para Web donde los artefactos de arquitectura son considerados ciudadanos de primera categoría.
2. Los artefactos del proceso WebSA se definen siguiendo el paradigma de ingeniería dirigida por modelos MDE [55], donde todos los modelos y las transformaciones son computables. Esto proporciona una trazabilidad precisa desde el análisis hasta la implementación y permite acelerar el proceso de desarrollo.
3. WebSA utiliza los estándares MDA [89]. La especificación de cada uno de los modelos del proceso WebSA esta realizada mediante un metamodelo MOF [88]. Este hecho permite establecer restricciones a la hora de especificar dichos modelos, vincular los diferentes modelos y obtener la información necesaria cuando se definen las transformaciones. Por otro lado, los modelos de WebSA especifican su notación mediante un perfil UML 2.0. Esto asegura su compatibilidad con UML y permite definirlos en cualquier herramienta UML. La definición de las transformaciones se realiza mediante UPT [76], que permite transformar modelos utilizando una notación UML, a partir de cualquier modelo que tenga su metamodelo definido en MOF. Por último, las transformaciones modelo a texto se basan en la propuesta de estándar MOFScript [94].
4. WebSA es capaz de considerar no sólo requisitos funcionales (que dirigen la definición de los modelos funcionales) sino también requisitos no funcionales a través de las fuerzas que dirigen los patrones de arquitectura definidos en los diferentes modelos de arquitectura.
5. WebSA permite un alto grado de reutilización. La definición de los modelos de arquitectura y de los modelos funcionales de una forma independiente permite la reutilización de la arquitectura para diferentes dominios de aplicación. Además, esta separación dota al método de la posibilidad de aplicar a un mismo dominio del problema diferentes arquitecturas. Por otro lado, la representación de patrones de arquitectura y diseño mediante WebSA permite que dichos patrones también sean reutilizables entre sistemas desde la fase de análisis.
6. WebSA es una aproximación funcionalmente genérica, es decir, complementa otras aproximaciones Web actuales como UWE [59], OO-H [19], WebML [20], incorporándoles los modelos de arquitectura y las transformaciones para poder obtener una solución más completa. La única restricción que debe cumplir cualquier aproximación funcional para ser complementada mediante WebSA es formalizar sus modelos mediante un metamodelo MOF.

Una vez caracterizada nuestra aproximación, seguidamente se presentan las principales aportaciones realizadas por el presente trabajo.

10.1.2 Aportaciones de WebSA

A continuación se repasan aquellos conceptos presentados por la presente tesis que son considerados como aportaciones propias y novedosas. Se han intentado resumir mostrando únicamente los aspectos más relevantes:

1. Definición de un proceso de desarrollo para las aplicaciones Web instancia del proceso unificado (UP) y adecuado al desarrollo dirigido por modelos (MDE).
2. Definición de un estilo arquitectónico denominado modelo de subsistemas (MS) Web basado en un proceso de descomposición dirigido por los patrones de distribución que indican cuál es la separación idónea en diferentes subsistemas de la aplicación Web.
3. Adaptación de las fuerzas que inducen a la aplicación de los diferentes patrones de distribución al dominio de las aplicaciones Web.
4. Formalización de MS mediante un metamodelo MOF y estandarización de su notación a través de su representación mediante un perfil UML 2.0.
5. Definición de un estilo arquitectónico denominado modelo de configuración (MC) basado en el componente Web como unidad arquitectónica. El MC, mediante un conjunto de componentes Web y sus conectores, permite representar la arquitectura de la aplicación Web de una forma genérica, para cualquier tipo de dominio.
6. Definición de una topología de componentes Web basadas en elementos de arquitectura utilizados en otras aproximaciones como WAM [24], EDOC [96], REST [31] y a partir del propio estudio de variabilidades de la arquitectura dentro del dominio de las aplicaciones Web.
7. Definición de un perfil UML para MC basado en la extensión del modelo de composición estructurada definido en UML 2.0.
8. Definición de un modelo de arquitectura denominado modelo de integración (MI) que representa presenta un modelado de la aplicación Web cercano a la implementación final, en la que se representan tanto los aspectos funcionales como los aspectos de arquitectura, pero que abstrae los aspectos puramente dependientes de la plataforma.
9. Definición de un metamodelo de componentes que unifica los elementos de arquitectura definidos en MC y los elementos definidos en el modelo de integración (MI).
10. Definición de un perfil UML para MI basado en la extensión del modelo de composición estructurada definido en UML 2.0.
11. Definición de un lenguaje de transformaciones llamado UPT (UML Profile for Transformations) que permite establecer las conversiones modelo a modelo utilizando una notación basada en un perfil sobre el diagrama de clases de UML.
12. Definición de la transformación denominada T1 que propone la fusión de los modelos funcionales definidos por otras propuestas de la ingeniería Web con los modelos de la vista de arquitectura definida por WebSA. Como resultado de T1 se obtiene el modelo de integración (MI). T1 es una transformación formada por 5 subtransformaciones y 80 reglas.
13. Definición de la transformación T2 que, a partir de MI, establece la integración de los aspectos dependientes de plataforma para obtener la

implementación final. La integración se realiza mediante la definición de transformaciones modelo a texto definidas en la propuesta del estándar de la OMG MOFScript [94].

14. Implementación de la herramienta WebTE, que cubre tres aspectos: (1) dar soporte a la carga de los metamodelos y transformaciones que propone WebSA, (2) dotar de extensibilidad a las transformaciones para que puedan añadirse y/o modificarse y (3) proporcionar interoperabilidad entre herramientas para que puedan intercambiar los modelos y las transformaciones entre las diferentes herramientas.

Una vez indicadas las características y aportaciones de este trabajo, queda por señalar qué ventajas proporciona su utilización.

10.1.3 Principales Ventajas de WebSA

A continuación, se presentan las ventajas que se obtienen por la adopción de la aproximación WebSA para el desarrollo de aplicaciones Web.

Estas ventajas se han dividido atendiendo a la característica de WebSA que las proporciona.

En primer lugar, la definición proporcionada por WebSA de la arquitectura del software en el desarrollo Web contribuye a:

1. Pretende la mejora de la satisfacción del cliente gracias a la consideración de los requisitos funcionales y no funcionales a través de la representación de la arquitectura del software en las fases tempranas.
2. Aumento de la gama de aplicaciones que son abordables mediante una metodología de desarrollo hipermedial. WebSA abre el campo al modelado de aplicaciones Web que dan servicio a un gran número de clientes en Internet, con una parte servidora basada en componentes distribuidos, comunicación con sistemas legados, diversos almacenes de datos, etc.
3. Tanto la parte funcional como la arquitectura se pueden definir en forma independiente. Permitiendo que los actores encargados de cada perspectiva puedan trabajar de forma concurrente, ahorrando tiempo en la definición del análisis. Además, se podrá realizar una reutilización de cada uno de los modelos definidos en cada una de las vistas. Por un lado, se reutiliza la misma vista de arquitectura para diferentes dominios y aplicaciones. Y por otro lado, se aplica el mismo problema o vista funcional para diferentes arquitecturas.
4. Eliminación del gap existente entre los modelos funcionales y la implementación final de la aplicación.
5. Mejora del mantenimiento de las aplicaciones Web. Una mejor documentación de la arquitectura permite mejorar la comunicación entre los modeladores y programadores a través de la representación de los componentes de la aplicación.
6. Reducción del esfuerzo basándose en las mejores prácticas proporcionadas por la definición de patrones de arquitectura y diseño.

Por otro lado, las ventajas obtenidas a partir de la definición de un proceso dirigido por modelos para las aplicaciones Web son las siguientes:

7. Reducción del trabajo y aceleración del proceso de desarrollo Web gracias a la automatización de los mapeos que van desde los modelos de análisis hasta la implementación.
8. Reducción del número de errores generados dentro del proceso de desarrollo gracias a la menor intervención humana.
9. Completa trazabilidad en los mapeos propuestos desde el análisis, pasando por el diseño y la implementación. La trazabilidad se obtiene mediante el modelado de las transformaciones.

Por último, las ventajas obtenidas por basarse en los estándares MDA para definir los artefactos son las siguientes:

10. Integración con modelos funcionales proporcionados por otras aproximaciones basadas en MOF para su formalización.
11. Posibilidad de utilizar herramientas comerciales de modelado UML para representar los diferentes modelos y transformaciones.
12. Reutilización de frameworks, parsers y componentes que permiten reducir el tiempo de realización de la herramienta WebTE.
13. Interoperabilidad de WebTE, al permitir la utilización de modelos y transformaciones realizadas en otras herramientas.

Una vez presentadas las principales propiedades de WebSA, en la siguiente sección se señalan las principales líneas de trabajo abiertas por esta tesis.

10.2 Principales Restricciones y Limitaciones de WebSA

A continuación, se presentan cuáles son las restricciones y limitaciones que presenta actualmente la aproximación WebSA en el desarrollo de aplicaciones Web.

1. La interfaz de usuario de las aplicaciones Web obtenidas es mejorable puesto que el modelo de presentación todavía no se ha integrado dentro del proceso de WebSA.
2. La lógica de negocio no se obtiene de forma completa puesto que no se integra con un modelo de proceso que permita definir transacciones entre los diferentes elementos. Esto exige un trabajo manual por parte de los desarrolladores.
3. Aumento en la formación de los analistas que deben representar los modelos de arquitectura en el flujo de trabajo de análisis. O disponer de expertos en Arquitectura Web dentro del equipo de desarrollo.
4. Aumento de formación en los desarrolladores que exige un conocimiento de las transformaciones de modelo a modelo, y específicamente de la notación UPT utilizada para su representación.
5. Aumento de la formación para el tratamiento de las transformaciones de texto definidas para obtener el código de las plataformas a partir del modelo de integración.

10.3 Producción Científica

El trabajo ha sido contrastado por el resto de la comunidad científica mediante la publicación y exposición de los resultados a través de los congresos y revistas de carácter nacional e internacional. Las publicaciones se centran principalmente en la presentación de WebSA, del lenguaje de transformaciones UPT, y de la mejora del método OO-H.

- **Artículos en Revistas Internacionales**

Meliá S., Gomez J. **The WebSA Approach: Applying Model Driven Engineering to Web Applications**. Journal of Web Engineering ©Rinton Press. Vol. 5, No. 2, pp. 121-149. ISSN: 1540-9589. http://www.rintonpress.com/journals/jwe. June 2006

Meliá S., Gomez J. **Applying Transformations to Model Driven Development of Web applications.** Perspectives in Conceptual Modeling. Lecture Notes in Computer Science © Springer-Verlag. Volume 3770/2005. ISSN: 0302-9743. JCR-2005 Impact Factor: 0,402. 1st International Workshop on Best Practices of UML. (ER, 2005), Austria, October 2005

Meliá S., Gomez J., Koch N. **Improving Web Design Methods with Architecture Modeling**. E-Commerce and Web Technologies. Lecture Notes in Computer Science © Springer-Verlag. Volume 3590/2005. ISSN: 0302-9743. JCR-2005 Impact Factor: 0,402. 6th International Conference on Electronic Commerce and Web Technologies (ECWeb, 2005). Copenhagen, Denmark, August 2005

Meliá S., Kraus A., Koch N. **MDA Transformations Applied to Web Application Development**. Web Engineering. Lecture Notes in Computer Science. © Springer-Verlag. Volume 3579/2005. ISSN: 0302-9743. JCR-2005 Impact Factor: 0,402. 5th International Conference on Web Engineering (ICWE'05). Sydney, Australia, July 2005

Meliá S., Cachero C. **An MDA Approach for the Development of Web Applications**. Web Engineering. Lecture Notes in Computer Science. © Springer-Verlag. Vol. 3140/2004. ISSN: 0302-9743. JCR-2004 Impact Factor: 0,513. 4th International Conference on Web Engineering (ICWE'04). Munich, Germany, July 2004

Koch N., A. Kraus, Cachero C., Meliá S. **Integration of Business Processes in Web Application Models**. Journal of Web Engineering © Rinton Press. Vol. 3, No 1. pp. 022-049 ISSN: 1540-9589. http://www.rintonpress.com/ journals/jwe, June 2004

- **Contribuciones en Congresos Internacionales**

Meliá S., Gomez J. **UPT: A Graphical Transformation Language based on a UML Profile**. Proceedings of European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA 2006). 2nd European Conference on Model Driven Architecture (EC-MDA 2006). July 2006

Meliá S., Gomez J. **Applying WebSA to a case study: A travel agency system**. 1st International Workshop on Model-Driven Web Engineering (ICWE, 2006). Sydney, Australia, July 2005

Meliá S., Cachero C., Gómez J. **Using MDA in Web Software Applications.** 2nd International Workshop in Generative Techniques in the context of MDA. OOPSLA 2003. Anaheim, California, EEUU, October 2003

Koch N., A. Kraus, Cachero C., Meliá S.. **Modeling Web Business Processes with OO-H and UWE**. Proceedings of 3rd International Workshop on Web Oriented Software Technology. Oviedo, Spain. July 2003

- **Contribuciones en Congresos Nacionales**

Meliá, S., Gómez, J., Serrano, J.L., Mazón, J-N. **Un perfil UML para la definición de un lenguaje gráfico de transformaciones basado en QVT**. XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2006). Sitges, Barcelona. October 2006

10.4 Trabajos Futuros

Las áreas en las que se ha fundamentado la presente tesis son muy recientes y están en constante evolución. De este modo, durante la elaboración de este trabajo han sido constantes las apariciones de nuevos estándares, lenguajes y propuestas que se han ido incorporando a la idea original.

Las áreas de investigación y su evolución van a fijar cuáles son los aspectos en los que se tiene que seguir invirtiendo esfuerzos para mejorar la propuesta. En primer lugar, por lo que respecta a aspectos relativos a la Arquitectura del Software, quedan propuestos como trabajos futuros:

- La introducción de la vista de Arquitectura Dinámica mediante la utilización de modelos estándares UML. Esta vista permitirá la representación de la coreografía de los componentes y su comportamiento interno mediante diagramas de estado. La idoneidad del diagrama de estructura compuesta para relacionarse con la parte dinámica mediante diagramas de secuencia hace que sea viable su pronta incorporación.
- El perfeccionamiento e introducción de nuevos tipos de componentes Web según se vaya identificando su necesidad.
- La elaboración de una librería de patrones de arquitectura modelados con la propuesta del modelo de configuración que puedan ser reutilizados para diferentes proyectos.

- La vinculación de los patrones de arquitectura con los requisitos no funcionales. De este modo, mediante un mecanismo guiado, sería posible elegir que patrón o patrones son más adecuados para un conjunto determinado de requisitos.

Por lo que respecta al área de investigación de la Ingeniería Web, se abren las siguientes oportunidades de ampliación:

- Incorporación de la vista de requisitos no funcionales para Web, que permitirá que pueda inferirse una arquitectura por defecto a partir de ellos.
- Consideración de la vista de presentación a partir de propuestas como OO-H, UWE, WebML o W2000, que permita la mejora de la calidad en las interfaces Web obtenidas y reduzca el trabajo realizado por el diseñador de interfaz.
- Definición de una vista de proceso que aborde la lógica de negocio de la aplicación. Esta vista permitirá obtener una lógica de negocio completa y reducir al mínimo el trabajo manual de los programadores.
- Incorporación de las vistas de personalización y usuario, que permitirá la obtención de aplicaciones Web adaptables.
- Incorporación de una vista que permita modelar almacenes de datos, de manera que se facilite la explotación remota de este tipo de almacenes. Además, la incorporación del análisis y minería de datos en las aplicaciones Web para determinadas facetas como el estudios sobre audiencias, aspectos de calidad, etc. permitiría la modificación estática y/o dinámica de la aplicación Web en función de los resultados obtenidos.

Otra área de trabajo que se encuentra en continua evolución por su juventud es la ingeniería dirigida por modelos. En esta área se prevé que los mayores esfuerzos se dirijan en un futuro hacia:

- La incorporación a la transformación T1 de nuevas reglas de transformación definidas en UPT para la incorporación de los todos los modelos funcionales y vistas propuestas por otras aproximaciones como UWE, WebML y W2000.
- La incorporación a la transformación T2 de nuevas reglas de transformación en MOFScript que permitan la transformación a otras plataformas Web como PHP, Coldfusion, etc.
- El mapeo del lenguaje de transformaciones declarativo UPT al lenguaje imperativo Operational Mappings de QVT. Esto permitirá formalizar el salto que existe entre la definición de una regla de transformación UPT y su implementación en un lenguaje imperativo como Java o C#.
- La definición de un repositorio de metamodelos de aproximaciones de Ingeniería Web y de transformaciones que puedan ser compartidos entre los diferentes grupos de investigación y así perfeccionarse con el tiempo.

Por último, se prevé seguir trabajando en la herramienta WebTE con el fin de mejorar los siguientes aspectos:

- Mejorar la carga de las transformaciones UPT desde las diferentes versiones XMI generadas por cualquier herramienta UML.
- Mejorar la carga de los modelos de arquitectura y funcionales para que puedan definirse desde cualquier herramienta UML.

- Asegurar la trazabilidad completa de los errores producidos en la carga de los modelos en los metamodelos JMI.
- Permitir la generación automática de los metamodelos JMI a partir de los metamodelos MOF. Esto permitirá poder utilizar rápidamente los modelos funcionales de diferentes aproximaciones y considerarlos en el proceso de desarrollo de WebSA.
- Mejorar la trazabilidad de la transformación de modelo a texto introduciendo aspectos como trazas entre el modelo y el código generado, seguimiento de los segmentos generados, protección de los segmentos que hayan sido modificados a mano, etc.

Volver al índice/Tornar a l'índex

APÉNDICE I

Perfil UML 2.0 del Modelo de Subsistemas

Se presenta la especificación completa del perfil UML 2.0 definido para el modelo de Subsistemas, complementado así la presentación del perfil introducida en el capítulo 5. Para poder describir el perfil, es necesario navegar a través de la nueva relación *extension* que proporciona el mecanismo de perfil de UML 2.0. Para ello, se define la siguiente operación OCL *isStereotyped*:

isStereotyped (stereotypeName:String) : Boolean;
self.extension-> exists (x | x.ownedEnd.type = stereotypeName)

A la hora de presentar la información de cada uno de los estereotipos se dispone de una plantilla que especifica su nombre, descripción, atributos o valores definidos, restricciones, la semántica que tiene asociada el estereotipo y su notación.

Nombre: SubsistemaWeb
Descripción SubsistemaWeb extiende a la metaclase UML "*Class::Kernel::Package*" (ver Figura. 18). Este concepto es un estereotipo abstracto que no puede ser instanciado. Es el elemento raíz de la jerarquía de estereotipos para cada uno de los subsistemas.
Restricciones **context** PerfilModeloSubsistemas::SubsistemaWeb -- *Un SubsistemaWeb está solamente relacionado con otros elementos SubsistemaWeb* **inv**: self. basePackage.supplierDependency->forAll (s|s.client->forAll((sc|sc.oclIsTypeOf ("SubsistemaWeb")) and self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf ("SubsistemaWeb)

Nombre: EnlaceSubsistema
Descripción Un EnlaceSubsistema extiende la metaclase Dependency de UML. "*Classes::Dependencies::Dependency*" (ver Figura. 18).
Restricciones No tiene restricciones adicionales

Nombre: InterfazUsuario
Descripción El estereotipo *InterfazUsuario* se define como una especialización del estereotipo *SubsistemaWeb.*
Restricciones **context** PerfilModeloSubsistemas::InterfazUsuario *-- Una InterfazUsuario es hijo de SubsistemaWeb* **inv**: self.parent.isStereotyped("SubsistemaWeb") *-- Puede estar relacionado como cliente de Servidor, LogicaNegocio o ControlNegocio. Y es proveedor de otro estereotipo InterfazUsuario.* **inv:** self. basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (Servidor)) or self.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (LogicaNegocio)) or self.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (ControlNegocio)) **inv:** self. basePackage.supplierDependency->forAll (s|s.client->forAll((sc|sc.oclIsTypeOf (InterfazUsuario))

Nombre: Servidor
Descripción El estereotipo *Servidor* se define como una especialización del estereotipo *SubsistemaWeb.*
Restricciones **context** PerfilModeloSubsistemas::Servidor *-- Un Servidor es hijo de SubsistemaWeb* **inv**: self.parent.isStereotyped("SubsistemaWeb") *-- El Servidor puede estar relacionado como proveedor de Presentacion ,InterfazUsuario o Servidor. Y como cliente de otro Servidor.* **inv:** self. basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (Servidor)) **inv:** self.basePackage.supplierDependency->forAll (s|s.client-> forAll ((sc|sc.oclIsTypeOf (Presentacion)) or self.basePackage.supplierDependency->forAll (s|s.client->forAll((sc| sc.oclIsTypeOf (InterfazUsuario)) or self.basePackage.supplierDependency->forAll (s|s.client->forAl ((sc|sc.oclIsTypeOf (Servidor))

Nombre: Presentacion

Descripción
El estereotipo *Presentación* se define como una especialización del estereotipo *SubsistemaWeb.*

Restricciones
context PerfilModeloSubsistemas::Presentación
-- *Presentación es hija de SubsistemaWeb*
inv: self.parent.isStereotyped("SubsistemaWeb")
-- *Puede estar relacionado como cliente de Servidor, LogicaNegocio o ControlNegocio. Y es proveedor de otro estereotipo Presentacion.*
inv: self. basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (Servidor)) or self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (LogicaNegocio)) or self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (ControlNegocio))
inv: self.basePackage.supplierDependency->forAll (s|s.client->forAl ((sc|sc.oclIsTypeOf (Presentacion))

Nombre: LogicaNegocio

Descripción
El estereotipo *LogicaNegocio* se define como una especialización del estereotipo *Servidor.*

Restricciones
context PerfilModeloSubsistemas::LogicaNegocio
-- *Un Servidor es hijo de Servidor*
inv: self.parent.isStereotyped ("Servidor")
-- *La LogicaNegocio puede estar relacionado como cliente con Persistencia, AccesoDatos y LogicaNegocio. Y como proveedor de LogicaNegocio.*
inv: self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (LogicaNegocio)) or self.basePackage.clientDependency->forAll(i|i.supplier->forAll ((is|is.oclIsTypeOf (Persistencia)) or self.basePackage.clientDependency->forAll (p|p.supplier->forAll (ps|ps.oclIsTypeOf (AccesoDatos))
inv:self.basePackage.supplierDependency->forAll(s|s.client->forAll((sc|sc.oclIsTypeOf (LogicaNegocio))

Nombre: Persistencia

Descripción
El estereotipo *Persistencia* se define como una especialización del estereotipo *Servidor.*

Restricciones
context PerfilModeloSubsistemas::Persistencia
-- *Un Servidor es hijo de Servidor*
inv: self.parent.isStereotyped ("Servidor")
-- *El Servidor puede estar relacionado como cliente de Persistencia. Y como proveedor de LogicaNegocio y Persistencia.*
inv: self. basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (Persistencia))
inv:self.basePackage.supplierDependency->forAll(s|s.client->forAll((sc| sc.oclIsTypeOf(Persistencia))

Nombre: ControlUsuario
Descripción El estereotipo *ControlUsuario* se define como una especialización del estereotipo *SubsistemaWeb.*
Restricciones **context** PerfilModeloSubsistemas::ControlUsuario *-- Una InterfazUsuario es hija de SubsistemaWeb* **inv**: self.parent.isStereotyped("SubsistemaWeb") *-- Puede estar relacionado como cliente de Servidor, LogicaNegocio o ControlUsuario. Y es proveedor de otro estereotipo Presentacion y ControlUsuario.* **inv:** self. basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (Servidor)) or self. basePackage.clientDependency->forAll(n|n.supplier->forAll ((ns|ns.oclIsTypeOf (LogicaNegocio)) or self. basePackage.clientDependency->forAll(i|i.supplier->forall ((is|is.oclIsTypeOf (ControlUsuario)) **inv:** self. basePackage.supplierDependency->forAll (s|s.client->forAl ((sc|sc.oclIsTypeOf (ControlUsuario)) or self. basePackage.supplierDependency->forAll (p|p.client->forAl ((pc|pc.oclIsTypeOf (Presentacion))

Nombre: ControlProcesos
Descripción El estereotipo *ControlProcesos* se define como una especialización del estereotipo *LogicaNegocio.*
Restricciones **context** PerfilModeloSubsistemas::ControlProcesos *-- Un ControlProcesos es hijo de LogicaNegocio* **inv**: self.parent.isStereotyped("LogicaNegocio") *-- Puede estar relacionado como cliente de ObjetosNegocio o ControlProcesos. Y es proveedor de otro estereotipo ControlProcesos.* **inv:** self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (ObjetosNegocio)) or self.basePackage.clientDependency->forAll(p|p.supplier->forAll ((ps|ps.oclIsTypeOf (ControlProcesos)) **inv:** self.basePackage.supplierDependency->forAll (s|s.client->forAl ((sc|sc.oclIsTypeOf (ControlProcesos))

Nombre: ObjetosNegocio
Descripción El estereotipo *ObjetosNegocio* se define como una especialización del estereotipo *LogicaNegocio.*
Restricciones **context** PerfilModelSubsistema::ObjetosNegocio *-- Un ObjetosNegocio es hijo de LogicaNegocio.* **inv**: self.parent.isStereotyped("LogicaNegocio") *-- Puede estar relacionado como cliente de ObjetosNegocio o ControlProcesos. Y es proveedor de otro estereotipo ControlProcesos.* **inv:** self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (ObjetosNegocio)) or self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf

(ControlProcesos)) **inv:** self.basePackage.supplierDependency->forAll (s|s.client->forAl ((sc|sc.oclIsTypeOf (ControlProcesos)) or self. basePackage.supplierDependency->forAll (o|o.client->forAl ((oc|oc.oclIsTypeOf (ObjetosNegocio))

Nombre: AccesoDatos
Descripción El estereotipo *AccesoDatos* se define como una especialización del estereotipo *Persistencia.*
Restricciones **context** PerfilModeloSubsistemas::AccesoDatos *-- Un ObjetosNegocio es hijo de Persistencia.* **inv**: self.parent.isStereotyped("Persistencia") *-- Puede estar relacionado como cliente de DatosFisicos o AccesoDatos.* *Y es proveedor de otro estereotipo AccesoDatos.* **inv:** self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (DatosFisicos)) or self. basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (AccesoDatos)) **inv:** self.basePackage.supplierDependency->forAll (s|s.client->forAl ((sc|sc.oclIsTypeOf (AccesoDatos))

Nombre: DatosFisicos
Descripción El estereotipo *DatosFisicos* se define como una especialización del estereotipo *Persistencia.*
Restricciones **context** PerfilModeloSubsistemas::DatosFisicos *-- Un DatosFisicos es hijo de Persistencia.* **inv** self.parent.isStereotyped("Persistencia") *-- Puede estar relacionado como cliente de DatosFisicos.* *Y es proveedor de AccesoDatos o DatosFisicos.* **inv:** self.basePackage.clientDependency->forAll(c|c.supplier->forAll ((cs|cs.oclIsTypeOf (DatosFisicos)) **inv:** self.basePackage.supplierDependency->forAll (s|s.client->forAll ((sc|sc.oclIsTypeOf (AccesoDatos)) or self.basePackage.supplierDependency->forAll(d|d.client->forAll((dc|dc.oclIsTypeOf (DatosFisicos))

Volver al índice/Tornar a l'índex

APÉNDICE II

Topología de Componentes Web

En este apéndice se describe una topología de 19 tipos de componentes que han sido identificados dentro de la familia de aplicaciones Web. Estos tipos de componentes se utilizan para la definición de los modelos de Configuración y de Integración. A continuación se realiza una descripción de los diferentes tipos de componentes en función si se trata de tipos comunes que pueden vivir en cualquier parte del sistema, o tipos específicos de un determinado tipo de subsistema o capa lógica. Cada tipo de componente podrá tener asociado un conjunto de propiedades y operaciones.

A2.1 Tipos de Componentes Comunes

Se considera a estos tipos de componentes como comunes para los diferentes subsistemas, debido a que los componentes poseen la misma tarea o función en cada uno de los subsistemas. Así, se consigue reducir el número tipos distintos de componentes Web, haciendo más sencillo el modelado de MC. Sin embargo, su ubicación si tiene repercusión en la fase de implementación, ya que dependiendo del subsistema en la que esté situado el componente, su implementación puede ser variar. En la Figura. 29 representa la estructura jerarquica de los tipos de componentes comunes dentro del metamodelo MC.

- **ComponenteWeb:** representa la forma básica de la encapsulación, proporcionando las características que permiten configurar la arquitectura Web en el modelo MC. Dentro de las diferentes instancias de MC, el arquitecto puede utilizar este tipo genérico en 2 casos: (1) cuando desee definir un componente que represente a un patrón de arquitectura, o (2) en aquellos componentes atípicos, que no pueden catalogarse como comunes dentro de la arquitectura Web. Debido a su genericidad, puede contener cualquier tipo de propiedad o de operación, ya sea de control o de funcionalidad. El ComponenteWeb es la clase raiz sobre la que se definen todas las propiedades a nivel de metamodelo.
- **DatosEntidad:** basado en EDOC [96], representa una entidad funcional definida por una clase del modelo de dominio. En este caso, este componente

puede estar localizado en cualquier subsistema de nuestra aplicación Web. DatosEntidad podrá contener las siguientes características funcionales:

- **tieneIdClase**: atributo booleano que indica que dicha entidad posee uno o más atributos que identifican a la clase de dominio.
- **tieneAtributos**: atributo booleano que indica que DatosEntidad posee los atributos de la clase de dominio.
- **tipoAtributos**: atributo cadena que indica el modelo funcional de procedencia de los atributos. Sus valores son: (1) *navegación,* (2), *proceso* y (3) *dominio* (valor por defecto).
- **tienePropiedades:** atributo cadena que indica si la clase tiene propiedades de acceso público a los atributos. Sus valores pueden ser (1) *lectura,* si la propiedad permite leer los atributos, (2) *escritura*, si las propiedades permiten escribir en los atributos, (3) *lect-esc*, si las propiedades permite leer y escribir en los atributos, (4) *ninguno,* si no posee propiedades (valor por defecto).

- **Vista Legada:** Componente que representa una puerta de acceso a una funcionalidad proporcionada o requerida por un sistema externo. Tiene los siguientes propiedades:
  - **invocación**: atributo que puede ser (1) asíncrono o (2) sincrono según el tipo de llamadas que admite el sistema legado.
  - **codificación:** atributo de control que tiene los siguientes valores: (1) *des*, si la vista descodifica la llamada desde el sistema legado al sistema, (2) *cod*, si la vista codifica la invocación del sistema hacia el sistema legado, (3) *descod*, si hace ambas tareas, (4) *ninguno*, si no hace ninguna tarea.
- **Almacén:** Componente que mantiene y gestiona un conjunto de información de forma persistente y permite que otros componentes dispongan de ella. Tiene 3 atributos:
  - **acceso**: atributo de control que indica cual es el tipo de acciones que se puede realizar sobre el almacén. Sus valores son: (1) *lectura*, (2) *escritura* y (3) *lect-esc*: lectura y escritura.
  - **tipoOrganización**: atributo de control que puede ser de tres tipos: (1) *Relacional*: Base de Datos Relacional, compuesta por un conjunto de tablas relacionadas. (2) *OrientadoObjetos*: Base de Datos Orientada a Objetos, más sofisticada que el relacional puesto que no solo permite el tratamiento de los datos, sino que proporcionan un lenguaje propietario que posibilita la definición de lógica de negocio a este nivel. (3)*FicherosPlanos*: ficheros físicos, ya sea con un formato propio o estándar (XML).
  - **tipoDatos**: atributo funcional que indica que el tipo de datos puede ser: (1) *dominio,* (2) *navegación y* (3) *proceso.*
- **Componente de Acceso a Datos (CAD):** Componente cuya funcionalidad es la de acceder a los datos persistentes del sistema, realizando el mapeo de datos a entidades y viceversa, independizando así a las componentes del dominio del tipo de persistencia que el sistema utiliza. Las operaciones de control que pueden asociarse a este elemento son:

  - **insertar**: operación que se encarga de introducir en el almacén los datos de una clase de dominio.
  - **modificar**: operación que se encarga de modificar los datos de una clase de dominio.
  - **borrar**: operación que se encarga de borrar los datos de una clase de dominio
  - **consultarTodo**: operación que se encarga de consultar todos los registros relacionados con una clase.
  - **consultarFiltro:** operación que se encarga de realizar una consulta filtrada. Filtro es expresado en OCL.
  - **consultarID:** operación que se encarga de devolver un registro de la clase por su identificación.
- **Librería de Funciones:** Componente que contiene una colección de funciones estructuradas y accesibles por el resto de componentes del sistema. Puede contener el conjunto de operaciones de control o de funcionalidad que el modelador disponga.
- **CacheWeb**: Componente que mantiene la información durante un periodo de tiempo determinado y dentro de un ámbito datos. Permite reducir el número de interacciones de una capa con su inferior, mejorando la eficiencia y la escalabilidad. Tiene como atributo:
  - **alcance**: atributo de control que puede tener 4 valores, *sesión*: mantiene la información para la sesión de un determinado usuario. *Aplicación*: componente que mantiene la información común para toda la aplicación desde que inicia hasta que termina. *Usuario*: mantiene la información para un determinado usuario. Y *petición*: mantiene la información solo durante el tiempo de la petición del cliente.
- **Recurso:** Componente que contiene un elemento utilizado para la presentación de una página. Tiene un atributo:
  - **tipo**: atributo de control que puede tener los valores: *imagen, sonido, video, datos* según el tipo de datos que almacena.

A2.2 Tipos de Componentes Específicos

Son aquellos componentes que por su tarea específica se encuentran emparejados con uno o más subsistemas. Esta vinculación entre el componente y subsistema tiene importancia porque dicha información es tomada por la transformación T1 para integrar los modelos de subsistemas y configuración.

Por ejemplo, el componente *PaginaEstática* puede estar contenido en un subsistema *InterfazUsuario o Presentación*. Sin embargo, nunca se puede contener en un subsistema *Servidor* o *Persistencia* ya que realiza una tarea relacionada con la interfaz Web.

Por otro lado, un componente que se asocia a un subsistema de una determinada capa siempre se puede asociar a otro subsistema de una capa más genérica. Por ejemplo, si se indica que el componente *PaginaServidora* se asocia al subsistema

ControlUsuario, también se puede asociar al subsistema más genérico *InterfazUsuario*.

A2.2.1 Interfaz de Usuario

Capa genérica encargada de dotar de la funcionalidad necesaria para la interacción entre el usuario y la aplicación. En función de la aplicación del patrón de distribución, puede contener el subsistema *InterfazUsuario* cuando se aplica el patrón de distribución *Interfaz Remoto* o puede estar dividida en los subsistemas *Presentación* y *ControlUsuario,* cuando se aplica el patrón *Presentación Distribuida.* A la hora de elegir cuáles son los diferentes componentes Web de interfaz, WebSA se ha basado en el trabajo de Conallen [24].

En la Figura. 30 representa la estructura jerarquica de los tipos de Interfaz de Usuario dentro del metamodelo MC.

Antes de cubrir los componentes por cada subsistema, se mostrará el componente AgenteUsuario, el cual se puede asociar de forma indistinta al subsistema InterfazUsuario o Presentación.

- **AgenteUsuario:** componente que recibe las peticiones del cliente, reenviándolas al servidor y convirtiéndose en el último receptor de la respuesta del servidor, para a continuación mostrársela al cliente. El ejemplo más conocido es el navegador Web, que provee acceso al servicio y renderiza las respuestas de acuerdo a las necesidades de la aplicación. Tiene un atributo:
  - **tipoDispositivo,** atributo de tecnología que puede tener los valores *Ordenador, PDA, TV o Mobil,* según el tipo de dispositivo que se conecta.

A2.2.1.1 Subsistema Interfaz Usuario

Subsistema que proporciona un cliente pesado, es decir, con una interfaz gráfico rico y permite un procesamiento desconectado. El componente que puede asociarse es:

- **objetoWeb**: Componente que es ejecutado en el cliente proporcionando la funcionalidad tanto de presentación como de control de diálogo. Normalmente se ejecuta en el navegador mediante la instalación de programas (plugins), los ejemplos más típicos son los Applets Java, los componentes Flash y los ActiveX.

A continuación, se presentan los elementos que pueden asociarse a cada una de las subcapas de Presentación y Control de usuario, (igualmente pueden utilizarse en la capa Interfaz de Usuario).

A2.2.1.2 Subsistema Presentación

Este subsistema proporciona un cliente ligero, fácilmente distribuible, que no permite el procesamiento desconectado. En este subsistema va a aparecer los siguientes componentes Web:

- **paginaEstática:** Componente que se ejecuta en el navegador y contiene el contenido necesario para presentar la información al usuario y realizar las invocaciones al servidor. Puede contener referencias a otros componentes

como PaginaEstilo y PaginaFuncional. Ejemplos de implementación son las paginas HTML, XHTML, WML, etc.

- **paginaEstilo:** Componente que únicamente mantiene información referente a la presentación de la interfaz de usuario. Ejemplos de implementación de este componente son las páginas CSS, XSL.
- **paginaFuncional:** Componente que mantiene funcionalidad para la interacción del usuario con la interfaz de la aplicación. Dicha funcionalidad puede ser validación, presentación, navegación, etc. Puede ser referenciado por un componente cliente. Ejemplo de implementación son las paginas Javascript y Actionscript. Contiene un atributo:
  - **tipoTarea**: atributo de funcionalidad, que indica el tipo de tarea que realiza dicha página. Puede ser: (1) *validación,* (2) *presentación o* (3) *invocación.*

A2.2.1.3 Subsistema de ControlUsuario

Este subsistema llamado ControlUsuario se encarga de realizar el procesamiento de la funcionalidad de presentación, tanto en lo que se refiere a su navegación, a su presentación, y en las invocaciones que se realicen a la capa de la lógica. Basándose en la separación funcional de procesamiento, entrada y salida que hace el patrón MVC [17] y en WAM [24], en la capa ControlUsuario se incorporan elementos comunes dentro de este subsistema

Los principales tipos identificados son:

- **PáginaServidora:** Componente que realiza el procesamiento de la pagina Web mediante código ejecutado en la parte servidora, normalmente por un motor que ejecuta el código de la página y le devuelve el contenido al servidor Web. Este componente puede contener información referente a la presentación, navegación y dominio. Con lo que podrá tener asociados atributos navegación y dominio. Ejemplo de implementación son las ASP, JSP, PHP, Coldfusion, etc. Sus propiedades son:
  - **tieneAtributos**: atributo booleano que indica que PaginaServidora posee los atributos de la clase de dominio.
  - **tipoAtributos**: atributo cadena que indica el modelo funcional de procedencia de los atributos. Sus valores son: (1) *navegación,* (2), *proceso* y (3) *dominio* (valor por defecto).
- **ControladorWeb:** Componente que se encarga de recibir las peticiones realizadas por el usuario, reenviarlas a la lógica de negocio y establecer la reacción de la interfaz. La funcionalidad que aporta es la de enviar las peticiones de servicios a la lógica de negocio, y redireccionar las peticiones de navegación a páginas servidoras. Puede tener asociados entre otros las siguientes operaciones de control:
  - **get:** operación de control que se activa mediante la llamada GET desde un navegador. Puede tener dos tipos de parámetros: un *enlaceServicio*: recibe los parámetros de un servicio de navegación o un *enlaceNavegación*: recibe los parámetros de un enlace de navegación.

  - **post:** operación de control que se activa mediante la llamada POST desde un navegador. Como parámetro puede tener un *enlaceServicio* que recibe los parámetros de un servicio de navegación o un *enlaceNavegación* recibiendo los parámetros de un enlace de navegación.
- **Vista:** Componente obtiene la información del modelo de navegación y muestra la información al cliente introduciendo aspectos del formateo de salida y de personalización. Independizando los aspectos de interfaz de usuario del dominio de la aplicación. Puede tener las siguientes propiedades:
  - **tipoPersonalización**: atributo cadena que indica que tipo de personalización realiza la Vista. Sus valores son: (1) *contenido:* que restringe el contenido de navegación mostrado por la vista *y* (2) *formato*: formatea la información mostrada y (3) *ninguno:* si muestra la información como llega de dominio (valor por defecto).
  - **tieneIdClase**: atributo booleano que indica que dicha entidad posee uno o más atributos que identifican a la clase de dominio. Por defecto es verdadero.
  - **tieneAtributos**: atributo booleano que indica que DatosEntidad posee los atributos de la clase de dominio. Por defecto es verdadero.
  - **tipoAtributos**: atributo cadena que indica el modelo funcional de procedencia de los atributos. Sus valores son: (1) *navegación (*valor por defecto*)*, (2), *proceso* y (3) *dominio.*
  - **tienePropiedades:** atributo cadena que indica si la clase tiene propiedades de acceso público a los atributos. Sus valores pueden ser (1) *lectura,* si la propiedad permite leer los atributos, (2) *escritura*, si las propiedades permiten escribir en los atributos, (3) *lect-esc*, si las propiedades permite leer y escribir en los atributos, (4) *ninguno,* si no posee propiedades (valor por defecto).

A2.2.2 Lógica de Negocio

Parte del sistema que resuelve las reglas de negocio especificadas para el dominio del problema. Este subsistema puede a su vez estar divido en dos subsistemas: ControlProcesos y ObjetosNegocio. Ambas partes aparecen especificadas por la vista conceptual de sistema y se van a distribuir en un conjunto de componentes que en función de los requisitos de calidad van a sugerir la utilización de computación distribuida o no. Los componentes identificados en este subsistema están basados en elementos definidos en los perfiles del estándar de la OMG EDOC (Enterprise Distributed Object Computing) [96]. En la Figura. 31 representa la estructura jerarquica con los tipos de componentes de servidor dentro del metamodelo MC.

A2.2.2.1 Subsistema de Procesamiento de Control

Parte del sistema se encarga de atender las peticiones que se reciben del cliente, convirtiéndolas en procesos que se realizarán de forma transaccional o no y de forma síncrona o asíncrona. Aquí se utilizarán las siguientes combinaciones de componentes:

- **ComponenteProceso:** Componente funcional que representa a una unidad de procesamiento activa. Este componente puede proporcionar acceso a operaciones o implementar las mismas operaciones mediante los datos provenientes de las Entidades. ComponenteProceso contiene los siguientes atributos de control:
  - **tieneEstado**: atributo booleano que indica si el componente almacena un estado específico de una sesión a través de las interacciones. El mecanismo de control de dicha sesión queda fuera del alcance de WebSA. Por defecto es falso.
  - **esTransactional**: atributo booleano que indica si las operaciones realizadas se realizan de forma transaccional o no. Por defecto el valor es verdadero.
  - **esDistribuido**: atributo booleano que indica si el componente es alcanzable a través de la red. Por defecto el valor es verdadero.
  - **esCompartido**: atributo booleano que indica si el componente puede ser accedido por múltiples sesiones. Por defecto es verdadero.
  - **esSincrono**: atributo booleano que indica si el componente recibe peticiones síncronas o asíncronas. Por defecto es sincrono.

  Por otro lado, puede contener un conjunto de atributos y operaciones funcionales asociados con el modelo de dominio.

A2.2.2.2 Subsistema de Objetos de negocio

En este subsistema residen los elementos que contienen la representación de los objetos definidos en el dominio del problema. En nuestra aproximación aparecen definidos en la vista conceptual. Estos componentes pueden ser del tipo:

- **Entidad:** Componente que representa un concepto del dominio de la aplicación en la capa de lógica de negocio, esto supone que en algunos casos incorpora los mismos datos que un componente DatosEntidad, pero además provee la funcionalidad de comportamiento derivada del modelo de dominio. Sus atributos de control son:
  - **esCompartido**: atributo booleano que indica si una Entidad puede ser compartida por múltiples, concurrentes transacciones o usuarios. Por defecto es verdadero.
  - **esPersistente**: atributo booleano que indica si dicha Entidad tiene un mecanismo de almacenamiento asociado, incorporado por el entorno. Por defecto es falso.
  - **esSegura**: atributo booleano que indica si dicha Entidad tiene un mecanismo de seguridad otorgado por el entorno. Por defecto es falso.
  - **esDistribuible**: atributo booleano que indica si dicha Entidad es accesible desde la red, esto implica que disponga de una interfaz remota. Si no es distribuible tiene que ser accedida desde otra entidad que si lo sea. Por defecto es verdadero.
  - **tieneIdClase**: atributo booleano que indica que dicha entidad posee uno o más atributos que identifican a la clase de dominio. Por defecto es verdadero.

- **tieneAtributos**: atributo booleano que indica que DatosEntidad posee los atributos de la clase de dominio. Por defecto es verdadero.
- **tipoAtributos**: atributo cadena que indica el modelo funcional de procedencia de los atributos. Sus valores son: (1) *navegación*, (2), *proceso* y (3) *dominio* (valor por defecto).
- **tienePropiedades:** atributo cadena que indica si la clase tiene propiedades de acceso público a los atributos. Sus valores pueden ser (1) *lectura*, si la propiedad permite leer los atributos, (2) *escritura*, si las propiedades permiten escribir en los atributos, (3) *lect-esc*, si las propiedades permite leer y escribir en los atributos, (4) *ninguno*, si no posee propiedades (valor por defecto).

Por otro lado, puede contener entre otros, las siguientes operaciones de control:

- **Cargar:** operación que indica que el componente Entidad se comunica con la persistencia realizando una consulta que recupere el estado de la Entidad.
- **Almacenar:** operación que indica que el componente Entidad se comunica con la persistencia realizando un almacenamiento del estado de la Entidad.

A2.2.3 Persistencia

Parte del sistema encargada del dotarle de persistencia al sistema de información. Los componentes propios de esta parte del sistema son:

- **GestorConexiones:** Componente que proporciona una o más conexiones con un almacén de datos o un sistema legado. Va a mantener una serie de propiedades de control como son:
  - **cantidad:** indica la cantidad de conexiones que establece la fuente de datos con un sistema de almacenamiento o un sistema legado.
  - **conexionReutilizable:** si es falso, indica que cada conexión se crea cada vez que la solicita el usuario. En caso contrario, es obtenida al inicio y reutilizada posteriormente.
  - **esTransaccional**: Dicha fuente de datos tiene la posibilidad de admitir transacciones con el almacén de datos. Por defecto es verdadero.
  - **esSegura**: Necesita validación para el establecimiento de dicha conexión. Por defecto es verdadero.

esRemota: permite que la fuente de datos acceda a un almacén de datos a través de la red de forma remota. Por defecto es verdadero

APÉNDICE III

Perfil UML 2.0 del Modelo de Configuración

Se presenta la especificación completa del perfil UML 2.0 definido para el modelo de configuración, complementado así la presentación del perfil introducida en el capítulo 6. Como muestra la Figura. 34, el perfil de MC está formado a su vez, por un conjunto de perfiles cada uno de los cuáles contiene los estereotipos de forma estructurada a la disposición con la que se estableció el metamodelo que involucra a los modelos MC y MI. Siguiendo el orden de dependencia entre los diferentes perfiles, el primer Perfil que se define es el de Núcleo de Componentes común, que presenta los estereotipos de los elementos comunes a los dos modelos, a partir de entonces se define el resto de perfiles que contienen los estereotipos específicos del modelo MC.

Para poder describir cada perfil, es necesario navegar a través de la nueva relación *extension* que proporciona el mecanismo de perfil de UML 2.0. Para ello, se define la siguiente operación OCL *isStereotyped*:

isStereotyped (stereotypeName:String) : Boolean;
self.extension-> exists (x | x.ownedEnd.type = stereotypeName)

A la hora de presentar la información de cada uno de los estereotipos se dispone de una plantilla que presenta su nombre, descripción, atributos o valores definidos, restricciones, la semántica que tiene asociada el estereotipo y su notación.

A3.1 Perfil de Núcleo de Componentes Común

Como su nombre indica, especifica aquellos conceptos que se han definido en el paquete Núcleo del metamodelo de componentes y que son comunes para los dos modelos MC y MI. Los estereotipos definidos en este perfil extienden directamente de los elementos del modelo de estructura compuesta de UML 2.0, estableciendo las

restricciones necesarias para el establecimiento únicamente de las relaciones entre los elementos que forman parte de los modelos de WebSA.

Nombre: ComponenteWeb
Descripción ComponenteWeb extiende de la metaclase de UML "*CompositeStructures::StructuredClasses::Class*".
Atributos No tiene atributos adicionales.
Restricciones **context** PerfilMC::PerfilNucleoComponentes::ComponenteWeb *-- Un ComponenteWeb no tiene propiedades estáticas (atributos)* **inv**: self.baseClass.attributes->isEmpty() *--Los Interfaces de un ComponenteWeb deben ser instancias de InterfazWeb* and self.baseClass.implementation->forAll(i|i.contract.oclIsTypeOf (InterfazWeb)) *--Todos sus conectores deben ser instancia de ConectorWeb* and self.baseClass.ownedConnector->forAll (c| c.oclIsTypeOf(ConectorWeb)) *-- Todos sus puertos deben ser instancia de PuertoWeb* and self.baseClass.ownedPort->forAll (p | p.oclIsTypeOf(PuertoWeb))
Semántica Un ComponenteWeb es la unidad arquitectónica que representa a una abstracción de uno o más componentes software que pertenecen a una aplicación Web.
Notación ComponenteWeb mantiene la notación del structure class de UML.

Nombre: ConectorWeb
Descripción ConectorWeb extiende de la metaclase UML "*CompositeStructures::InternalStructures::Connector*" (ver Figura. 22 y Figura. 23).
Atributos No tiene atributos adicionales
Restricciones **context** PerfilMC::PerfilNucleoComponentes::ConectorWeb *-- Un ConectorWeb está relacionado mediante un FinalConnectorWeb.* **inv**: self.baseConnector.end-> forAll (e | e.isStereotyped("FinalConectorWeb"))
Semática ConectorWeb especifica un enlace que permite la comunicación en el sistema

entre dos o más elementos ComponenteWeb y/o ParteWeb
Notación Un ConectorWeb tiene dos notaciones, (1) por un lado un ConectorWeb une directamente a los puertos de los componentes Figura. 22 o la comunicación conlos puertos se realiza mediante InterfazWeb Figura. 23

Nombre: FinalConectorWeb
Descripción Un FinalConectorWeb extiende de la metaclase de UML "*CompositeStructure::InternalStructures::ConnectorEnd*" (ver Figura. 22) .
Atributos FinalConectorWeb tiene dos atributos o valores definidos: (1) *cardSuperior* que especifica el límite inferior de elementos que pueden estar conectados con el FinalConectorWeb. (2) *cardSuperior* que especifica el límite superior de elementos que pueden estar conectados a este elemento.
Restricciones **context** PerfilMC::PerfilNucleoComponentes::FinalConectorWeb *-- A WebConnectorEnd is related to a WebPort or a WebPart.* **inv**: self.baseConnectorEnd.role.isStereotyped("PuertoWeb")or self.baseConnectorEnd.partWithPort.isStereotyped("ParteWeb")
Semántica FinalConectorWeb representa un extreme del conector que une al conector con un PuertoWeb o con una ParteWeb
Notation Se distingue por mostrar las cardinalidades superiores e inferiores asociadas cada extremo. En el caso de que no se especificara dicha cardinalidad superior o inferior, esta sería igual a 1.

Nombre: InterfazWeb
Descripción El InterfazWeb extiende la metaclase de UML "*Class::Interfaces::Interface*" (ver Figura. 21).
Atributos No tiene atributos adicionales.
Restricciones **context** PerfilMC::PerfilNucleoComponentes::InterfazWeb *-- Todas las operaciones que contiene una InterfazWeb son instancias de OperacionWeb.* **inv:** self.baseInterface.ownedOperation->forAll (o | o.isStereotiped("OperacionWeb"))

Semántica InterfazWeb representa la funcionalidad asociada al componente, la cual ofrece o requiere del resto del sistema para poder realizar sus tareas.
Notación Una InterfazWeb tiene una notación lollipop (ver Figura. 21). Existen dos notaciones que corresponden a cada tipo de InterfazWeb. La interfaz requerida se muestra como un semicírculo abierto, y la interfaz ofertada se muestra como un círculo completo.

Nombre: PuertoWeb
Descripción Un PuertoWeb extiende de la metaclase UML "*CompositeStructure::Ports::Port*" (ver Figura. 21) .
Atributos No tiene atributos adicionales.
Restricciones **context** PerfilMC::PerfilNucleoComponentes::PuertoWeb *-- Un PuertoWeb provee y ofrece instancias de InterfazWeb.* **inv**: self.basePort.provided->forAll (in | in.isStereotyped ("InterfazWeb")) and self.basePort.required->forAll (in | in.isStereotyped ("InterfazWeb"))
Semántica Un PuertoWeb es un punto de interacción entre ComponenteWeb y su entorno, permitiendo desacoplar los aspectos internos del componente de la interacción con otros componentes. Esto permite hacer al ComponenteWeb más reusable para cualquier entorno, siempre que cumpla las restricciones de las interacciones impuestas por el PuertoWeb.
Notación Un PuertoWeb se representa por un cuadro pequeño en el borde de la clase (ver Figura. 21).

Nombre: ParteWeb
Descripción Una ParteWeb extiende de la metaclase de UML 2.0 "*CompositeStructure::InternalStructures::Property*".
Atributos *Multiplicidad*: usa la notación [x{...y}], donde (x) especifica la cantidad de instancias contenidas cuando el ComponenteWeb es creado e (y) indica la cantidad máxima de instancias en la vida del componente.

Constraints **context** PerfilMC::PerfilNucleoComponentes::ParteWeb -- *Una ParteWeb tiene instancias de PuertoWeb* **inv:** self.baseProperty.ownedPort->forAll (p | p.isStereotyped ("PuertoWeb"))
Semantics ParteWeb representa a un conjunto de instancias pertenecen por composición a una instancia de ComponenteWeb
Notation ParteWeb se representa por una caja en el interior de un ComponenteWeb o un PatronWeb. La multiplicidad es especificada en el interior de la caja que representa la ParteWeb (ver Figura. 24)

Nombre: OperacionWeb
Descripción OperacionWeb extiende de la metaclase de UML "*Classes::Kernel::Operation*"
Atributos No tiene atributos adicionales.
Restricciones **context** PerfilMC::PerfilNucleoComponentes::OperacionWeb -- *Un OperacionWeb contiene instancias de ParametroWeb.* **inv**: self.baseOperation.formalParameter->forAll (p | p.isStereotyped("ParametroWeb"))
Semántica Un OperacionWeb representa una propiedad de comportamiento de un ComponenteWeb.
Notación Se representa mediante su nombre en el interior de la caja que define el ComponenteWeb

Nombre: ParametroWeb
Descripción Un ParametroWeb extiende de la metaclase de UML "*Classes::Kernel::Parameter*".
Atributos No tiene atributos adicionales.
Restricciones No tiene restricciones adicionales.
Semántica Un ParametroWeb es la especificación de un argumento usado para enviar o recibir información de un OperacionWeb.

Notación Se representa mediante el nombre y tipo del parámetro como parte de la definición de un OperacionWeb.

A3.2 Perfil Elementos Modelo Configuración

En este perfil se incluyen únicamente aquellos estereotipos que constituyen los elementos específicos del modelo de configuración. Estos estereotipos en la mayoría de los casos son especializaciones de los estereotipos definidos en el perfil Núcleo Componentes Común pero con particularidades propias del MC.

Nombre: CompWebCM
Descripción Un CompWebCM es una especialización del estereotipo "*PerfilMC::PerfilNucleoComponentes::ComponenteWeb*".
Atributos No tiene atributos adicionales.
Restricciones **context** PerfilMC::PerfilElementosMC::CompWebCM *-- Todas sus propiedades de comportamiento son instancia de OperacionWeb.* **inv**: self.baseClass.ownedBehavior->forAll (s | s.oclIsTypeOf (OperacionWeb)) *--Todas sus ocurrencias son instancia de PatronWeb* and self.baseClass.ocurrences->forAll (o | o.oclIsTypeOf (PatronWeb)) *-- Las clases contenidas por un CompWebCM son de la instancia TipoComponente* and self.baseClass.nestedClassifier->forAll (c | c.oclIsTypeOf(TipoComponente))
Semántica Un CompWebCM representa una abstracción de uno o más componentes software que comparten la misma funcionalidad o tarea en el contexto de la familia de aplicaciones Web. Se utiliza cuando desee definir un componente que represente a un patrón de arquitectura o (2) en aquellos componentes atípicos, que no pueden catalogarse dentro de ningún tipo identificado por WebSA.
Notación Se representa mediante misma notación definida para ComponenteWeb. (ver Figura. 21)

Nombre: PatronWeb
Descripción Un PatrónWeb extiende de la metaclase de UML "*CompositeStructure:: StructuredClasses:: Collaboration*" (ver Figura. 25).
Atributos No hay atributos adicionales.

Restricciones **context** *PerfilMC::PerfilElementosMC::PatronWeb* *-- Un PatronWeb puede contener instancias de ParteWeb y ConectorWeb.* **inv**: self.baseCollaboration.collaborationRole->forAll (cr | cr.isStereotyped("ParteWeb") or cr.isStereotyped("ConectorWeb"))
Semántica Un PatronWeb representa a un patrón de arquitectura Web, el cual especifica un elemento compuesto formado por un conjunto de ConectorWeb y ParteWeb que corresponden a instancias de ComponenteWeb que cumplen con unos roles concretos para cumplir la tarea del patrón.
Notación Un PatronWeb se representa mediante la notación de una clase Collaboration de UML, es decir con un icono en forma de eclipse punteada, que en su interior contiene el nombre del patrón y un conjunto de elementos ParteWeb y ConectorWeb.

Nombre: ParteWebCM
Descripción Una ParteWebCM es una especialización del estereotipo de WebSA *"PerfilMC::PerfilNucleoComponentes::ParteWeb"* (ver Figura. 24).
Atributos No hay atributos adicionales.
Restricciones **context** *PerfilMC::PerfilElementosMC::ParteWebCM* -- El tipo de la ParteWebCM es una instancia de TipoComponente **inv**: self.baseProperty.datatype.oclIsTypeOf(TipoComponente)) -- Una parteWebCM puede pertenecer a un CompWebCM o a un PatronWeb and self.baseProperty.class.oclIsTypeOf(CompWebCM) or self.class.oclIsTypeOf (PatronWeb)
Semántica La *ParteWeb* representa al conjunto de instancias de un *CompWebCM* que pueden ser propiedad por composición de otro *CompWebCM* o pertenecer como actores de *PatronWeb*.
Notación Tiene la misma notación que la ParteWeb

Nombre: OWDominio
Descripción OWDominio es una especialización del estereotipo de WebSA *"PerfilMC::PerfilNucleoComponentesComun::OperacionWeb"*
Atributos No tiene atributos adicionales.

Restricciones No tiene restricciones adicionales
Semántica Un OWDomino representa una operación que contiene la funcionalidad del modelo de dominio en los componentes del modelo de configuración
Notación Se representa indicando el estereotipo <<OWDominio>> antes del nombre de la operación que contiene el ComponenteWeb

Nombre: OWNavegacion
Descripción OWNavegación es una especialización del estereotipo de WebSA *"PerfilMC::PerfilNucleoComponentes::OperacionWeb"*
Atributos No tiene atributos adicionales.
Restricciones No tiene restricciones adicionales
Semántica Un OWNavegación representa una operación que contiene la funcionalidad del modelo de navegación en los componentes del modelo de configuración
Notación Se representa indicando el estereotipo <<OWNavegación>> antes del nombre de la operación que contiene el ComponenteWeb

Nombre: OWProceso
Descripción SWProceso es una especialización del estereotipo de WebSA *"PerfilMC::PerfilNucleoComponentes::OperacionWeb"*
Atributos No tiene atributos adicionales.
Restricciones No tiene restricciones adicionales
Semántica Un OWProceso representa una operación que contiene la funcionalidad proveniente de la vista de proceso dentro de los componentes del modelo de configuración
Notación Se representa indicando el estereotipo <<OWProceso>> antes nombre de la operación que contiene el ComponenteWeb

A3.3 Perfil Componentes Comunes

Primero de los tres perfiles que contienen los estereotipos con los diferentes tipos de componentes Web identificados por WebSA para los modelos de configuración e

integración. Como su nombre indica, en el Perfil Componentes Comunes se sitúan aquellos tipos de componentes identificados como comunes o que poseen la misma tarea o función en cada uno de los subsistemas de la aplicación Web.

Nombre: DatosEntidad
Descripción DatosEntidad es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *tieneIdClase*: atributo booleano que indica que dicha entidad posee uno o más atributos que identifican a la clase de dominio. *tieneAtributos*: atributo booleano que indica que DatosEntidad posee los atributos de la clase de dominio. *tipoAtributos*: atributo cadena que indica el modelo funcional de procedencia de los atributos. Sus valores son: (1) *navegación*, (2), *proceso* y (3) *dominio* (valor por defecto). *tienePropiedades*: atributo cadena que indica si la clase tiene propiedades de acceso público a los atributos. Sus valores pueden ser (1) *lectura*, si la propiedad permite leer los atributos, (2) *escritura*, si las propiedades permiten escribir en los atributos, (3) *lect-esc*, si las propiedades permite leer y escribir en los atributos, (4) *ninguno*, si no posee propiedades (valor por defecto).
Restricciones No tiene restricciones adicionales.
Semántica Representa una entidad funcional definida por una clase del modelo de dominio
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo DatosEntidad.

Nombre: VistaLegada
Descripción VistaLegada es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *invocación:* atributo cadena que puede ser (1) asíncrono o (2) sincrono según el tipo de llamadas que admite el sistema legado. *codificación:* atributo cadena que tiene los siguientes valores: (1) des, si la vista descodifica la llamada desde el sistema legado al sistema, (2) cod, si la vista codifica la invocación del sistema hacia el sistema legado, (3) descod, si hace ambas tareas, (4) ninguno, si no hace ninguna tarea.
Restricciones No tiene restricciones adicionales.
Semántica Componente que representa una puerta de acceso a una funcionalidad proporcionada o requerida por un sistema externo

Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo VistaLegada.
Nombre: Almacén
Descripción Almacén es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *acceso*: indica cual es el tipo de acciones que se puede realizar sobre el almacén. Sus valores son: (1) lectura, (2) escritura y (3) lect-esc: lectura y escritura. *tipoOrganización:* que puede ser de tres tipos: (1) Relacional: Base de Datos Relacional, compuesta por un conjunto de tablas relacionadas. (2) OrientadoObjetos: Base de Datos Orientada a Objetos. (3)FicherosPlanos: ficheros físicos, ya sea con un formato propio o estándar (XML). *tipoDatos:* indica que el tipo de datos que contiene: (1) dominio, (2) navegación y (3) proceso.
Restricciones No tiene restricciones adicionales.
Semántica Componente que mantiene y gestiona un conjunto de información de forma persistente y permite que otros componentes dispongan de ella.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo Almacén.

Nombre: CAD
Descripción CAD es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos No tiene atributos adicionales
Restricciones No tiene restricciones adicionales.
Semántica CAD: indica Componente de Acceso a Datos. Su funcionalidad es la de acceder a los datos persistentes del sistema, realizando el mapeo de datos a entidades y viceversa, independizando así a las componentes del dominio del tipo de persistencia que el sistema utiliza.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo CAD.

Nombre: LibreríaFuncional

Descripción LibreríaFuncional es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos No tiene atributos adicionales
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que contiene una colección de funciones estructuradas y accesibles por el resto de componentes del sistema
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo LibreríaFuncional.

Nombre: CacheWeb
Descripción CacheWeb es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *alcance*: atributo de control que puede tener 4 valores, *sesión*: mantiene la información para la sesión de un determinado usuario. (1) *aplicación*: componente que mantiene la información común para toda la aplicación desde que inicia hasta que termina. (2) *usuario*: mantiene la información para un determinado usuario. Y (3) *petición*: mantiene la información solo durante el tiempo de la petición del cliente
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que mantiene la información durante un periodo de tiempo determinado y dentro de un ámbito datos
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo CacheWeb.

Nombre: Recurso
Descripción Recurso es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *tipo:* atributo que puede tener los valores: imagen, sonido, video, datos según el tipo de datos que almacena.
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente Web que contiene un elemento utilizado para la presentación de una página.

Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo Recurso.

A3.4 Perfil Componentes Interfaz Usuario

En este perfil se sitúan aquellos componentes que su tarea o funcionalidad está vinculada únicamente con la Interfaz de Usuario. En este grupo incluimos además, aquellos componentes que están relacionados con alguna de las subcapas de Interfaz de Usuario, como son Presentación y ControlUsuario.

Nombre: AgenteUsuario
Descripción AgenteUsuario es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *tipoDispositivo:* atributo de tecnología que puede tener los valores Ordenador, PDA, TV o Mobil, según el tipo de dispositivo que se conecta.
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que recibe las peticiones del cliente, reenviándolas al servidor y convirtiéndose en el último receptor de la respuesta del servidor, para a continuación mostrársela al cliente.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo AgenteUsuario.

Nombre: ObjetoWeb
Descripción ObjetoWeb es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos No tiene atributos adicionales
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que es ejecutado en el cliente proporcionando la funcionalidad tanto de presentación como de control de diálogo.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo ObjetoWeb.

Nombre: PaginaEstática

Descripción PaginaEstática es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos No tiene atributos adicionales
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que se ejecuta en el navegador y contiene el contenido necesario para presentar la información al usuario y realizar las invocaciones al servidor.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo PaginaEstática.

Nombre: PaginaEstilo
Descripción PaginaEstilo es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos No tiene atributos adicionales
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que únicamente mantiene información referente a la presentación de la interfaz de usuario.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo PaginaEstilo.

Nombre: PaginaFuncional
Descripción PaginaFuncional es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *tipoTarea*: indica el tipo de tarea que realiza dicha página. Puede ser: (1) validación, (2) presentación o (3) invocación
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que mantiene funcionalidad para la interacción del usuario con la interfaz de la aplicación. Dicha funcionalidad puede ser validación, presentación, navegación, etc.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo PaginaFuncional.

Nombre: ControladorWeb
Descripción ControladorWeb es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos No tiene atributos adicionales.
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que se encarga de recibir las peticiones realizadas por el usuario, reenviarlas a la lógica de negocio y establecer la reacción de la interfaz. La funcionalidad que aporta es la de enviar las peticiones de servicios a la lógica de negocio, y redireccionar las peticiones de navegación a páginas servidoras.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo ControladorWeb.

Nombre: Vista
Descripción Vista es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::DatosEntidad*".
Atributos *tipoPersonalización*: atributo cadena que indica que tipo de personalización realiza la Vista. Sus valores son: (1) *contenido:* que restringe el contenido de navegación mostrado por la vista *y* (2) *formato*: formatea la información mostrada y (3) *ninguno:* si muestra la información como llega de dominio (valor por defecto).
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente obtiene la información del modelo de navegación y muestra la información al cliente introduciendo aspectos del formateo de salida y de personalización. Independizando los aspectos de interfaz de usuario del dominio de la aplicación.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo Vista.

A3.5 Perfil Componentes Servidor

En este perfil se especifican aquellos tipos de componentes que están localizados en alguno de los subsistemas de servidor.

Nombre: EntidadWeb
Descripción EntidadWeb es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::DatosEntidad*".
Atributos *esCompartido*: atributo booleano que indica si una Entidad puede ser compartida por múltiples, concurrentes transacciones o usuarios. Por defecto es verdadero. *esPersistente*: atributo booleano que indica si dicha Entidad tiene un mecanismo de almacenamiento asociado, incorporado por el entorno. Por defecto es falso. *esSegura*: atributo booleano que indica si dicha Entidad tiene un mecanismo de seguridad otorgado por el entorno. Por defecto es falso. *esDistribuible*: atributo booleano que indica si dicha Entidad es accesible desde la red, esto implica que disponga de una interfaz remota. Si no es distribuible tiene que ser accedida desde otra entidad que si lo sea. Por defecto es verdadero.
Restricciones No tiene restricciones adicionales.
Semántica Componente que representa un concepto del dominio de la aplicación en la capa de lógica de negocio, esto supone que en algunos casos incorpora los mismos datos que un componente DatosEntidad, pero además provee la funcionalidad de comportamiento derivada del modelo de dominio.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo EntidadWeb.

Nombre: ComponenteProceso
Descripción ComponenteProceso es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *tieneEstado*: atributo booleano que indica si el componente almacena un estado específico de una sesión a través de las interacciones. El mecanismo de control de dicha sesión queda fuera del alcance de WebSA. Por defecto es falso. *esTransactional:* atributo booleano que indica si las operaciones realizadas se realizan de forma transaccional o no. Por defecto el valor es verdadero. *esDistribuido:* atributo booleano que indica si el componente es alcanzable a través de la red. Por defecto el valor es verdadero. *esCompartido:* atributo booleano que indica si el componente puede ser accedido por múltiples sesiones. Por defecto es verdadero. *esSincrono:* atributo booleano que indica si el componente recibe peticiones síncronas o asíncronas. Por defecto es sincrono.
Restricciones No tiene restricciones adicionales.

Volver al índice/Tornar a l'índex

Semántica Componente funcional que representa a una unidad de procesamiento activa. El componente puede proporcionar acceso a operaciones o implementar las mismas operaciones mediante los datos provenientes de las Entidades.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo ComponenteProceso.

Nombre: GestorConexiones
Descripción GestorConexiones es una especialización del estereotipo "*PerfilMC::PerfilElementosMC::CompWebCM*".
Atributos *cantidad:* indica la cantidad de conexiones que establece la fuente de datos con un sistema de almacenamiento o un sistema legado. *conexionReutilizable:* si es falso, indica que cada conexión se crea cada vez que la solicita el usuario. En caso contrario, es obtenida al inicio y reutilizada posteriormente. *esTransaccional:* Dicha fuente de datos tiene la posibilidad de admitir transacciones con el almacén de datos. Por defecto es verdadero. esSegura: Necesita validación para el establecimiento de dicha conexión. Por defecto es verdadero. *esRemota*: permite que la fuente de datos acceda a un almacén de datos a través de la red de forma remota. Por defecto es verdadero.
Restricciones No tiene restricciones adicionales.
Semántica Representa a un componente que proporciona una o más conexiones con un almacén de datos o un sistema legado.
Notación Se representa mediante misma notación definida para CompWebCM indicando el estereotipo GestorConexiones.

APÉNDICE IV

Los Estereotipos del Lenguaje de Transformaciones UPT

En este apéndice se presenta la especificación completa de cada uno de los estereotipos que permiten la representación del lenguaje de transformaciones UPT (UML Para Transformaciones). UPT ha extendido el diagrama de clases para expresar las transformaciones, definiendo un alineamiento de cada uno de los conceptos definidos en UPT con las metaclases definidas para representar el diagrama de clases de UML y además, con las metaclases OCL que hace uso el modelo.

Para poder describir los esterotipos, es necesario navegar a través de la relación *extension* que proporciona el mecanismo UML 2.0. Para ello, se define la siguiente operación OCL *isStereotyped*:

isStereotyped (stereotypeName:String) : Boolean;
self.extension-> exists (x | x.ownedEnd.type = stereotypeName)

A la hora de presentar la información los 14 estereotipos de UPT, cada uno de ellos se muestra mediante una plantilla que presenta su nombre, descripción, atributos o valores definidos, restricciones, la semántica que tiene asociada el estereotipo y su notación.

La Figura. 92 muestra el vínculo entre los diferentes estereotipos y las metaclases de UML extendidas. La descripción comienza con la metaclase Class, desde la cual los estereotipos *Transformación*, *PatronClase* y *Regla* heredan. Es importante destacar que el estereotipo Regla es abstracto y no puede ser usado directamente. Es su hijo, el estereotipo Relación, el que se instancia en su lugar.

En la parte derecha, el estereotipo *PatronPropiedad* hereda de *Property*, así *PatronPropiedad* mantiene la misma relación con *PatronClase* que *Property* con *Class*. Los estereotipos *OrdenTrans* y *OrdenRegla* que establecen el orden entre las transformaciones y reglas extienden de la metaclase *Dependency* de UML.

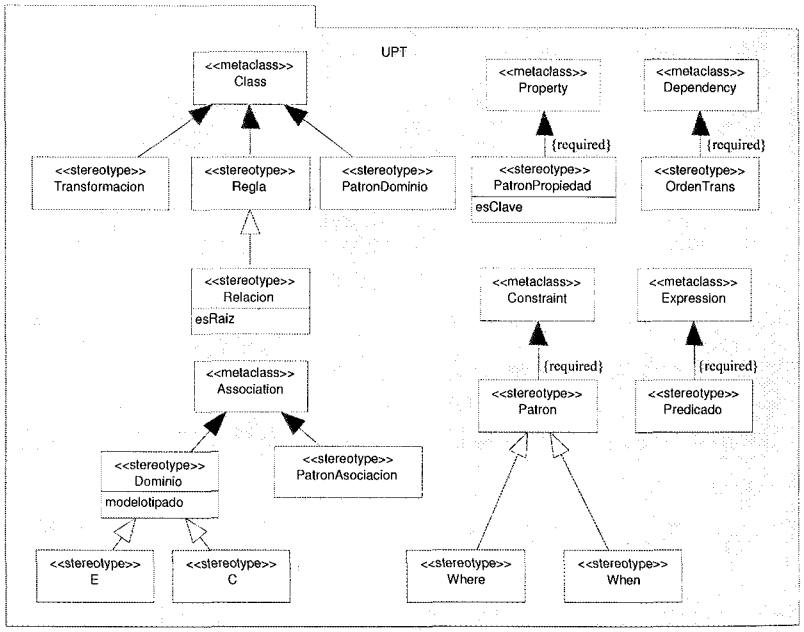

Figura. 92. Definición de los Esterotipos de UPT

En la parte inferior izquierda, son *Dominio* y *PatronAsociación* quienes extienden de la metaclase *Association*. Ambos estereotipos son usados para establecer relaciones entre elementos *Relación* y *PatronClase* respectivamente. *Dominio* establece un tipo definido llamado metamodelo donde indica un espacio de nombres o ruta a la parte del metamodelo. Además, de *Dominio* extienden dos estereotipos *E* y *C* que corresponden a los tipos de dominio enforceable y checkonly, respectivamente.

Como aconseja la especificación OCL [93], la metaclase UML *Expression* es el lugar donde deben estar situadas las expresiones OCL. En este sentido, Constraint es un punto de definición de las expresiones OCL que están vinculadas con una clase UML. Y concretamente en UPT, las expresiones OCL deben estar vinculadas con el estereotipo *Relación*. En este sentido, es el estereotipo *Patrón* quien se define como una extensión de la metaclase Constraint y desde *Patrón* extienden a su vez dos estereotipos *When* y *Where* que corresponden a las dos restricciones que podemos asociarle a la *Relación* de UPT. Finalmente, el *Predicado* es definido como una extensión de la metaclase *Expression* y representa una expresión booleana que pertenece a un *Patrón*.

Además de este emparejamiento de los estereotipos con la metaclases de UML y OCL, a continuación se representan las relaciones del metamodelo de UPT mediante la definición de restricciones sobre los metamodelos extendidos. La descripción comienza con los estereotipos que extienden de las metaclases del paquete Transformación, es decir, la propia Transformación UPT, Regla, la DependenciaTrans y el Dominio.

Nombre: Transformacion
Descripción Transformación extiende de la metaclase de UML "*Classes::Kernel::Class*"
Atributos No tiene atributos adicionales.
Restricciones **context** *UPT::Transformacion* *-- Una Transformación no tiene propiedades estáticas (atributos) ni dinámicas (operaciones)* **inv**:self.baseClass.attributes->isEmpty() and self.ownedOperation->isEmpty() *-- Las clases contenidas por una Transformación son instancia de Regla* and self.baseClass.nestedClassifier->forAll (c | c.oclIsTypeOf(Regla)) *-- Una Transformación extiende de otra Transformación* and self.baseClass.superClass->forAll (s | s.oclIsTypeOf(Transformación))
Semántica Una transformación especifica un conjunto de relaciones o reglas de transformación que deben llevarse a cabo o satisfacerse entre los elementos de un conjunto de modelos candidatos.
Notación Transformación mantiene la notación de la metaclase *class* de UML. (ver Figura. 60)

Nombre: Regla
Descripción Regla extiende de la metaclase de UML "*Classes::Kernel::Class*"
Atributos No tiene atributos adicionales.
Restricciones **context** *UPT::Regla* *-- Todas las relaciones de asociación que tenga una regla son instancia de Dominio.* **inv**: self.baseClass.attributes->forAll (a | a.association.oclIsTypeOf(Dominio)) *-- Una Regla extiende de otra Regla* and self.baseClass.superClass->forAll (s | s.oclIsTypeOf(Regla)) *--Regla es abstracta* and self.baseClass.isAbstract=true
Semántica Una Regla especifica de forma declarativa como los elementos de un dominio son computados desde los elementos de modelo de otros dominios.
Notación Regla no se representa al ser abstracta.

Nombre: DependenciaTrans

Descripción DependenciaTrans extiende de la metaclase de UML *"Classes::Dependencies::Dependency"*
Atributos No tiene atributos adicionales.
Restricciones **context** *UPT::DependenciaTrans* *-- Todos los roles supplier o client de Dependencia para un estereotipo DependenciaTrans, están relacionados con instancias de Transformacion.* **inv**: self.baseDependency.supplier->forAll (d|d.oclIsTypeOf ("Transformacion")) and self.baseDependency.client->forAll (d|d.oclIsTypeOf("Transformacion"))
Semántica DependenciaTrans es una relación que establece el orden de ejecución entre dos Transformaciones.
Notación DependenciaTrans mantiene la notación de la metaclase *Dependency* de UML. (ver Figura. 60)

Nombre: Dominio
Descripción Dominio extiende de la metaclase de UML *"Classes::Kernel::Association"*
Atributos *modeloTipado*: un modelo tipado indica la parte del metamodelo que participa en la Relación.
Restricciones **context** *UPT::Dominio* *-- Las clases a las que referencian los extremos de una relación son un PatronClase y una Relación.* **inv**: self.baseAssociation. endType->exists (p | p.oclIsTypeOf("PatronClase")) and self.baseAssociation.endType->exists (r | r.oclIsTypeOf("Relacion")) *-- Un dominio tiene únicamente 2 extremos* and self.baseAssociation.memberEnd->size( ) = 2 *-- Es una clase abstracta, solo se instancian sus hijos* and self.baseAssociation.isAbstract = true
Semántica Un dominio especifica el conjunto de elementos de un metamodelo que son de interés para una relación.
Notación El dominio no se representa al ser abstracto.

Los estereotipos que vienen a continuación son los que permiten definir las relaciones de UPT. Destaca la definición del estereotipo Relación y los dominios E (enforceable y C (checkable). Además, para reproducir los metamodelos en las relaciones los estereotipos PatronClase, PatronPropiedad y PatronAsociación. Y por último, para definir las condiciones OCL, los estereotipos Where, When, Patron y Predicado.

Nombre: Relacion
Descripción Relación es una especialización del estereotipo "*UPT::Regla*".
Atributos *esRaiz*: atributo booleano que indica si su valor es verdadero que la relación es invocada al iniciarse la ejecución de la transformación. Cuando su valor es falso indica que la relación es invocada por la parte where de otra relación.
Restricciones **context** UPT::Relacion *-- Las Constraints que contiene Relacion son instancia de Patron.* **inv**: self.baseClass.ownedRule->forAll (c | c.isOclIsTypeOf (Patron))
Semántica La Relación es la unidad básica dentro de la especificación declarativa de UPT. Una relación indica cuáles son las restricciones que deben cumplirse por elementos de los modelos candidatos.
Notación Relación mantiene la notación de la metaclase *class* de UML. (ver Figura. 55)

Nombre: C
Descripción C es una especialización del estereotipo "*UPT::Dominio*"
Atributos No tiene atributos adicionales.
Restricciones **context** *UPT::C* -- Este esterotipo puede ser instanciado. **inv**: self.baseAssociation.isAbstract = false
Semántica C señala que se trata de un dominio checkonly ("comprobar solamente"). Es decir, indica que se comprueba que los elementos definidos en el dominio se encuentran de la misma forma en el metamodelo referenciado. Si al comprobarlo no coinciden la relación falla.
Notación C mantiene la notación de la metaclase *Association* de UML. (ver Figura. 55)

Nombre: E
Descripción E es una especialización del estereotipo "*UPT::Dominio*"
Atributos No tiene atributos adicionales.
Restricciones **context** *UPT::E* -- El esterotipo E puede ser instanciado. **inv**: self.baseAssociation.isAbstract = false

Semántica E señala que se trata de un dominio enforceable ("hace cumplir la norma"). Indica que si los elementos del dominio no se encuentran igual en el metamodelo referenciado deben modificarse para conseguirlo.
Notación E mantiene la notación de la metaclase *Association* de UML. (ver Figura. 55)

Nombre: PatronClase
Descripción PatrónClase extiende de la metaclase de UML "*Classes::Kernel::Class*"
Atributos No tiene atributos adicionales.
Restricciones **context** UPT::PatronClase *-- Un PatronClase no tiene propiedades de comportamiento (operaciones)* **inv**: self.baseClass.ownedOperation->isEmpty() *-- Los atributos contenidos por un PatronClase son instancia de PatronPropiedad* self.attributes->forAll (a | a.oclIsTypeOf(PatronPropiedad)) *-- Un PatronClase no extiende de ninguna clase.* and self.baseClass.superClass->isEmpty()
Semántica Una PatrónClase es una referencia a una instancia de una metaclase dentro de un dominio.
Notación Patrón mantiene la notación de la metaclase *class* de UML. (ver Figura. 55)

Nombre: PatronAsociacion
Descripción PatronAsociación extiende de la metaclase de UML "*Classes::Kernel::Association*"
Atributos No tiene atributos adicionales.
Restricciones **context** UPT::PatronAsociacion *-- Todas las clases a las que referencian los extremos de una relación son instancia de PatronPropiedad.* **inv**: self.baseAssociation.endType->forAll (p | p.oclIsTypeOf ("PatronPropiedad"))
Semántica PatronAsociación establece una relación entre las instancias de dos PatronClase. Permite reproducir la relación de asociación que se establece en el metamodelo referenciado por el dominio entre las instancias de dos de sus metaclases.

Notación PatronAsociacion mantiene la notación de la metaclase *Association* de UML. (ver Figura. 55)

Nombre: PatronPropiedad
Descripción PatronPropiedad extiende de la metaclase de UML *"Classes::Kernel::Property"*
Atributos *esClave*: atributo booleano que permite indicar que los atributos identifican de forma unívoca a cada instancia de la metaclase referenciada.
Restricciones **context** UPT::PatronPropiedad *-- El tipo de la Propiedad es o un tipo primitivo o un PatronClase.* **inv**: self.baseProperty.datatype.oclIsTypeOf(PatronClase) or self.datatype.oclIsTypeOf(PrimitiveType) -- El valor por defecto de un PatronPropiedad es del tipo ExpresionOCL and self.baseProperty.defaultValue.oclIsTypeOf(ExpresionOCL)
Semántica PatronPropiedad especifica las restricciones en los valores que las instancias PatronClase pueden tomar.
Notación PatronPropiedad mantiene la notación de la metaclase *Property* de UML. (ver Figura. 55)

Nombre: Patron
Descripción Patrón extiende de la metaclase de UML *"Classes::Kernel::Constraint"*
Atributos No tiene atributos adicionales.
Restricciones **context** UPT::Patron *-- Los elementos que contiene Patron son del tipo Predicado.* **inv**: self.baseConstraint.constrainedElement->forAll (e | e.oclIsTypeOf (Predicado)) *-- Es una clase abstracta, solo se instancian sus hijos* and self.baseConstraint.isAbstract = true
Semántica Un Patrón es un conjunto de declaraciones de variables y predicados las cuáles cuando son evaluadas en el contexto de un modelo resultan en un conjunto de enlaces para las variables.
Notación No tiene notación puesto que no se instancia directamente.

Nombre: When

Volver al índice/Tornar a l'índex

Descripción When es una especialización del estereotipo "*UPT::Patron*"
Atributos No tiene atributos adicionales.
Restricciones No tiene restricciones adicionales.
Semántica When consiste en una condición que debe ser cumplida por las variables de la relación, para que esta sea llevada a cabo. When debe ser ejecutado antes de la relación
Notación Se especifica con la palabra When y una asignación de la expresión OCL basada en un conjunto de predicados.(ver Figura. 55)

Nombre: Where
Descripción Where es una especialización del estereotipo "*UPT::Patron*"
Atributos No tiene atributos adicionales.
Restricciones No tiene restricciones adicionales.
Semántica Where específica una condición que debe ser satisfecha por todos los elementos que han participado en la relación y puede restringir cualquiera de las variables de la relación. Where es ejecutada después de la relación.
Notación Se especifica con la palabra Where y una asignación de la expresión OCL basada en un conjunto de predicados. (ver Figura. 55)

Nombre: Predicado
Descripción Predicado extiende de la metaclase de UML "*Classes::Kernel::Expression*"
Atributos No tiene atributos adicionales.
Restricciones **context** *UPT::Predicado* *-- Un Predicado contiene a su vez una ExpresiónOCL.* **inv**: self.baseExpression.operand->forAll (e | e.oclIsTypeOf (OCLExpression))
Semántica Un Predicado es una expresión OCL booleana que pertenece a un Patrón. Es especificada mediante una expresión que puede contener referencias a variables del Patrón al que pertenece el Predicado.
Notación Se especifica de forma textual una expresión OCL. (ver Figura. 55)

APÉNDICE V

Relaciones de la Transformación T1 en UPT

Este apéndice presenta la especificación de las 20 relaciones principales de la transformación T1 de WebSA usando la notación UPT. Para ello, se describen con detalle las reglas que son más relevantes o aportan aspectos distintos en cada una de las 5 transformaciones que componen T1.

A5.1 T1_1: Creación e Indexación de Módulos

T1_1 propone la creación de un ModuloWeb por cada uno de los subsistemas definidos en el modelo de subsistemas. Una vez realizada la creación de los módulos, se procede a ubicar los componentes del modelo de configuración en cada uno de los módulos creados. T1_1 se basa en la correspondencia que existe en la tipología de componentes respecto a la tipología de subsistemas.

La Figura. 61 representa un mapa de relaciones simplificado de la transformación T1_1. La transformación T1_1 comienza la ejecución invocando a dos relaciones raíz. Primero se invoca a la relación *SubsistemasAModulos*, dicha la relación realiza la tarea de recorrer todas las instancias de SubsistemaWeb, y por cada una crear instancia de Módulo, (ver relación en la Figura. 93). Invocará a su vez por cada SubsistemaWeb a la relación *SubsistemaAModulo*, la cual está encargada de transformar las propiedades de un SubsistemaWeb a un Modulo. Sin embargo, no se utiliza directamente sino que es sobrescrita por otras relaciones que corresponden a los diferentes tipos de subsistemas definidos por el metamodelo de subsistemas (p.e. InterfazUsuario, Servidor, LogicaNegocio, etc.).

A continuación, se describen estas mismas relaciones pero aplicando la notación ejecutable definida por el lenguaje UPT. La primera relación mostrada en Figura. 93 es *SubsistemasAModulos*. Esta relación es una relación raíz que como indica su atributo *esRaiz=true*, es decir, es invocada desde la transformación. La relación posee dos dominics, el dominio *ModeloSubsistemas* de tipo checkonly y el dominio

ModeloIntegracion de tipo enforceable. En el patrón de la izquierda se recorren todas instancias de SubsistemaWeb que contenga el modelo de subsistemas y en el patrón de la derecha se indica la creación de un MóduloWeb por cada SubsistemaWeb. Además, por cada una de las instancias de SubsistemaWeb y ModuloWeb se invocará en la parte Where a la relación *SubsistemaAModulo*.

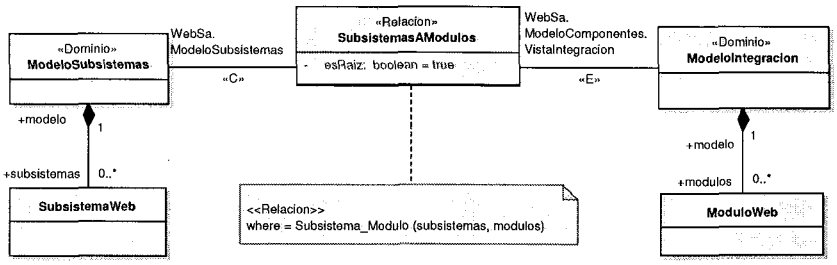

Figura. 93. Relación SubsistemasAModulos

La relación *SubsistemaAModulo* (ver Figura. 94) recibirá entonces como dominio los dos parámetros enviados, el SubsistemaWeb y el ModuloWeb. A partir de los dominios realizará la conversión de las propiedades de las dos instancias. Concretamente, crea el nombre del ModuloWeb a partir del nombre del SubsistemaWeb, utilizando la variable *n* de tipo String. Esta variable *n* permite que cualquier valor que tenga de tipo String sea copiado del nombre de SubsistemaWeb al nombre del ModuloWeb.

Sin embargo, la relación *SubsistemaAModulo* es sobrescrita por un conjunto de relaciones cada una de las cuáles corresponde a los diferentes tipos de Subsistemas definidos. Se han definido 10 relaciones correspondiendo a las 10 capas o tipos de subsistemas. Las relaciones especializadas pueden crear un módulo más específico a partir del subsistema tipado. El SubsistemaWeb presenta una jerarquía de herencia en la cual se especializan los diferentes tipos de subsistemas, a partir de la cual el módulo es tipado utilizando el atributo *capa*. Por ejemplo, *InterfazUsuarioAModulo* es una relación de un subsistema InterfazUsuario crea un Modulo en el cual se pone el campo *capa="InterfazUsuario"*. El atributo *capa* tiene como función respetar la distribución en capas propuesta en el modelo de subsistemas y situar la ubicación de los componentes según la capa.

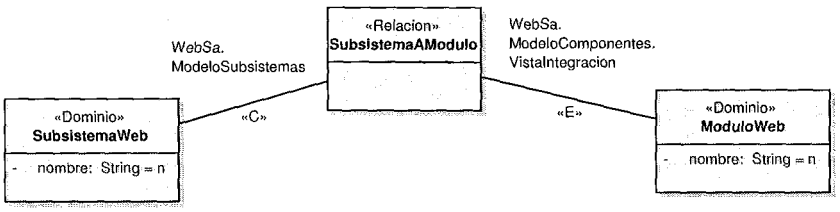

Figura. 94. Relación SubsistemaAModulo

La Figura. 95 muestra la relación *InterfazUsuarioAModulo*. Esta relación sobrescribe a *SubsistemaAModulo* y para ello utiliza la relación de herencia con el estereotipo *overrides* (sobrescribir). La relación especializada ha modificado el dominio *SubsistemaWeb* por una metaclase especializada *InterfazUsuario.* Así, la relación puede aportar más información a *ModuloWeb*, formando su nombre a partir del prefijo "IU" más el nombre del Subsistema e introduciendo el valor del atributo *capa.*

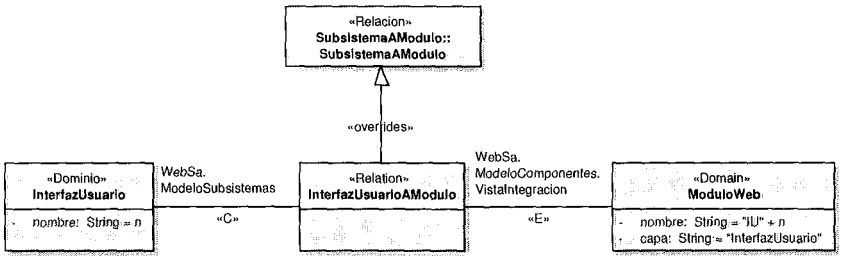

Figura. 95. Relación InterfazUsuarioAModulo

Una vez generados las instancias de *ModuloWeb*, la transformación T1_1 procede a localizar a los componentes del modelo de configuración, indicando cual es su ubicación. Para ello, se basa en la jerarquía de componentes definida en el metamodelo de configuración, en la cual existen tipos de componentes asociados a capas por ejemplo *ComponenteServidor, ComponenteInterfazUsuario,* etc.

Volviendo al mapa de relaciones de la transformación T1_1 (ver Figura. 61), en la parte derecha, está relacionada la transformación T1_1 con la relación raíz *LocalizarComponentesEnModulos,* igual que ocurría con el caso anterior, esta relación se encarga de iniciar la ejecución y de activar el recorrido de los elementos que van a intervenir en cada uno de los dominios. Esta relación contiene un conjunto de relaciones *LocalizarComponenteEnModulo* que se encarga de trabajar como dominio sobre cada una de las instancias recorridas, pero son en este caso las relaciones especializadas que la sobrescriben, por ejemplo *LocalizarComponenteServidorEnModulo* las que hacen la tarea de la transformación finalmente.

La Figura. 96 muestra la relación *LocalizarComponentesEnModulos*. Esta relación contiene dos dominios checkonly y un dominio enforceable. La parte checkonly contiene el Modelo de Configuración en el cual procede a recorrer todas las instancias de CompWebCM que contenga este modelo pero que cumplan con el When. En el

When se comprueba que CompWebCM sea igual a alguno de los tipos de componentes asociados a un determinado subsistema (p.e ComponenteInterfazUsuario, ComponenteServidor, etc.), el otro dominio checkonly contiene el ModeloIntegracion en el cual se recorren todas las instancias ModuloWeb creadas anteriormente. Una vez comprobados los dominios checkonly se recorre el enforceable en el cual se referencia de nuevo al dominio ModeloConfiguracion y se recorren los mismos elementos comprobados, pero en esta ocasión son modificados.

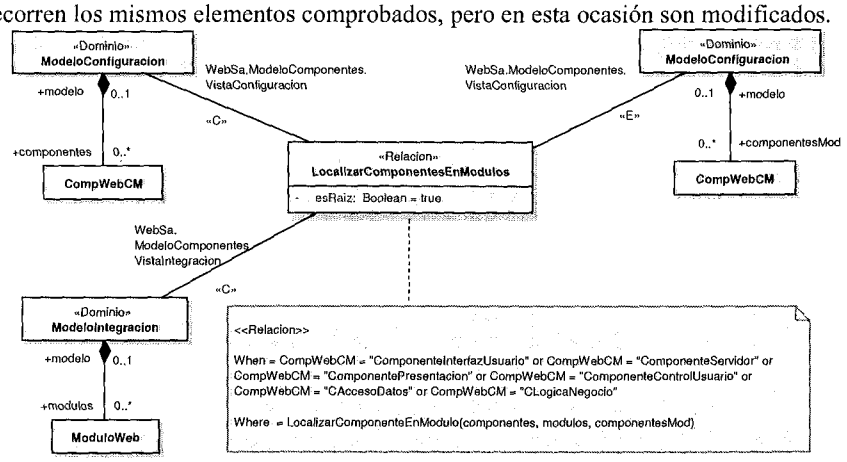

Figura. 96. Relación LocalizarComponentesEnModulos

Para modificar las propiedades de las instancias de CompWebCM, en la parte Where se procede a la invocación de la relación *LocalizarComponenteEnModulo*, una vez por cada una de las instancias de *CompWebCM*.

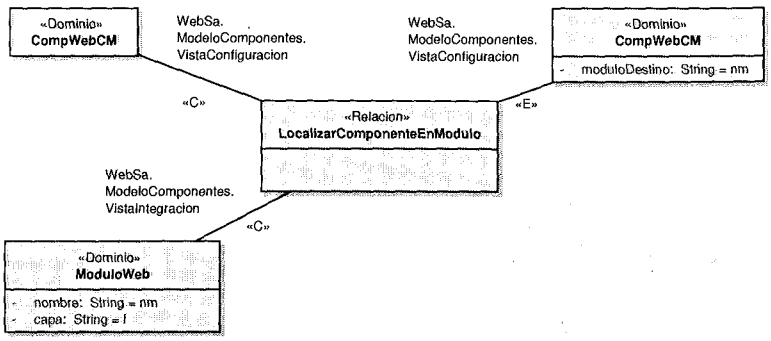

Figura. 97. Relación LocalizarComponenteEnModulo

Una vez la ejecución se encuentra en *LocalizarComponenteEnModulo* (ver Figura. 97), cada una de las instancias enviadas como parámetro son convertidas en dominio de dicha relación. Dicha relación comprueba la existencia de una instancia de *CompWebCM* y de una instancia de *ModuloWeb*. *ModuloWeb* lee su nombre y su capa almacenándolas en las variables *nm* y *l* respectivamente. La relación procede

entonces a modificar la misma instancia *CompWebCM* poniéndole a la variable *moduloDestino* el valor de *nm* que corresponde al nombre del *ModuloWeb*. Como puede apreciarse, esta relación es demasiado general y no consigue completar satisfactoriamente la vinculación entre el tipo del componente y el módulo que le corresponde. Para ello, se define un conjunto de relaciones que sobrescriben esta relación y que realizan la función de la especialización de cada una de las instancias de *CompWebCM*, por ejemplo, *LocalizarComponenteServidorEnModulo*, *LocalizarComponenteIUEnModulo*, etc.

Para ejemplificar, en la Figura. 98 se muestra la relación *LocalizarComponenteServidorEnModulo* que sobrescribe a la relación anterior *LocalizarComponenteEnModulo*. En el primer dominio checkonly referencia a un CompWebCM que está relacionado con un *ComponenteServidor*, es decir, el CompWebCM es del tipo ComponenteServidor, con lo que cumple el requisito de que debe estar localizado en un modulo servidor. Por ello, en el siguiente dominio el *ModuloWeb* tiene el nombre genérico en la variable *nm*, pero su capa debe ser tener el valor *capa*="*Servidor*". Finalmente, en el dominio enforceable al *ComponenteServidor* se le asigna el nombre del *ModuloWeb* que está almacenado en la variable *nm*.

Los componentes de integración no han sido creados hasta este momento. Solamente se ha marcado a los componentes de configuración para posteriormente ubicarlos. En las transformaciones siguientes se procederá a la creación de los componentes con la información del modulo destino. Existe un tipo de componentes de configuración que se llaman ComponentesComunes que pueden pertenecer potencialmente a cualquiera de los módulos. Los componentes tienen el campo *moduloDestino* con el valor "*none*" (en castellano ninguno). Los componentes comunes se sitúan en el módulo en función de la ubicación de los componentes que tienen relacionados.

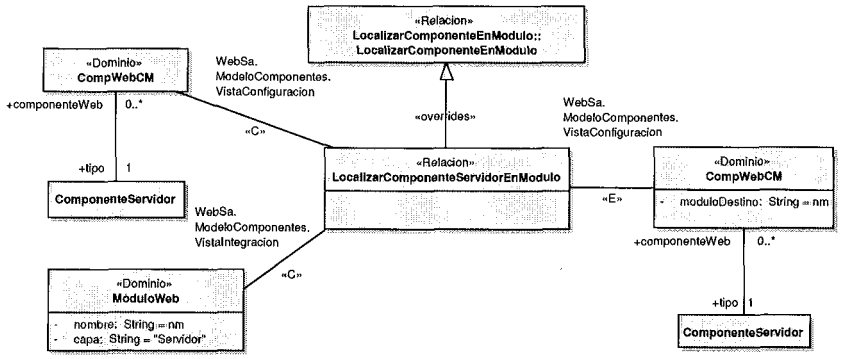

Figura. 98. Relación LocalizarComponenteServidorEnModulo

A5.2 T1_2: Creación de las Relaciones entre Módulos

La T1_2 es la segunda transformación dentro de la serie de transformaciones que compone T1. T1_2 tiene como objetivo establecer las relaciones o conexiones entre los diferentes módulos. Para ello, se sirve de los resultados obtenidos en la

transformación anterior, es decir, las instancias de *ModuloWeb* creadas y la indexación de los componentes del modelo de configuración a los diferentes *ModuloWeb*. T1_2 tomará los casos donde los componentes de configuración estén relacionados entre sí y se indexen a ModulosWeb distintos y establecerá una relación entre los diferentes módulos.

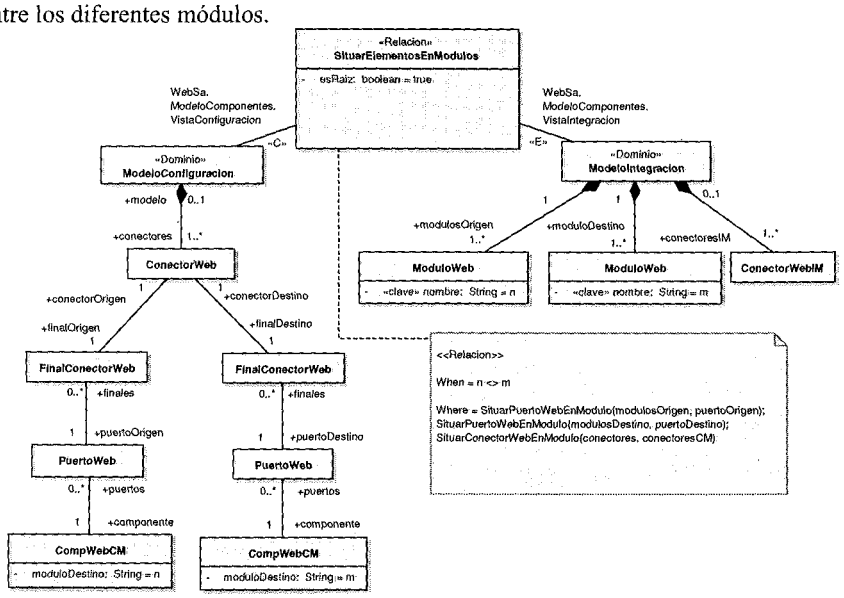

Figura. 99. Relación SituarElementosEnModulos

La Figura. 62 muestra el mapa de composición de T1_2, en el cual la transformación se vincula con la relación raíz *SituarElementosEnModulos* que comienza la ejecución invocando a las relaciones que sitúan los diferentes elementos de la conexión. Por un lado, *SituarConectorWebEnModulo*, y por otro lado, *SituarPuertoWebEnModulo*. Esta última relación invocará a la relación *DePuertoCMAPuertoIM*.

A continuación, se describen estas mismas relaciones aplicando la notación definida por el lenguaje UPT. Se comienza con la relación raíz *SituarElementosEnModulos* (ver Figura. 99). En el lado izquierdo está situado como dominio checkonly el modelo de configuración. El primer PatronClase de este dominio es el propio modelo de configuración, el cual es el punto de entrada al conjunto de elementos que el modelo contiene. Así, el modelo de configuración contiene un conjunto de instancias ConectorWeb, del cual parten dos extremos o FinalConectorWeb que están relacionados con dos puertos de dos componentes.
Los dos componentes tienen el atributo *moduloDestino* referenciado por las variables *n* y *m*. El patrón recorre los conectores de los componentes del modelo de configuración, y busca aquellas ocurrencias en las que los componentes tengan un *modeloDestino* diferente. Una vez comprobado, la relación continúa en el dominio enforceable situado en el modelo de integración. Este dominio comprueba la existencia de dos módulos que tienen los mismos nombres que las variables *n* y *m*.

Por último, se crean las instancias de *ConectorWebIM* que conectan los elementos *ModuloWeb*.

Una vez completada la relación, se realizan las invocaciones a otras relaciones en la parte Where. Se invoca a la relación *SituarPuertoWebEnModulo* en dos ocasiones, primero para el puerto y el modulo en origen y después para el puerto y el modulo en destino. Por último, se llama a la relación *SituarConectorWebEnModulo*, a la cual se le pasa como parámetro los conectores del modelo de configuración que cumplen la relación para convertirlos en nuevos conectores en el modelo de integración.

La relación *SituarPuertoWebEnModulo* (ver la Figura. 100) tiene como objetivo añadir un puerto a cada uno de los módulos implicados en la relación. En la parte derecha se sitúa el dominio checkonly donde el modelo de configuración únicamente contiene un PatronClase llamado PuertoWeb que es recibido de la relación anterior. En la parte izquierda se aprecia el dominio enforceable sobre el modelo de integración, donde el primer PatronClase es el ModuloWeb que está relacionado con un conjunto de instancias PuertoWebIM. La relación creará una instancia de PuertoWebIM a partir de la instancia de PuertoWeb perteneciente al modelo de configuración. Una vez creada la instancia, se invoca a la relación *DePuertoWebAPuertoWebIM* encargada de copiar las propiedades del PuertoWeb a PuertoWebIM.

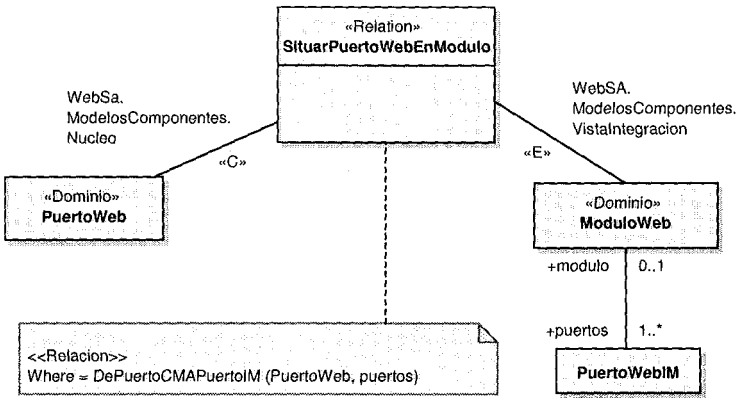

Figura. 100. Relación SituarPuertoWebEnModulo

La última relación descrita de la transformación T1_2 es *SituarConectorWebEnModulo*. Esta relación mostrada en la Figura. 101 se encarga de establecer la conexión de un ConectorWebIM entre dos módulos diferentes. Para ello, primero en la parte checkonly, se comprueba la existencia de una instancia de *ConectorWebCM* que establezca la relación con dos instancias *PuertoWeb* que pertenecen a dos componentes diferentes *CompWebCM* de nombres *n* y *m*. Una vez se constata la existencia del conector, se crea en el dominio enforceable localizado en el modelo de integración, un *ConectorWebIM* que relacione los módulos. Para ello, en la raíz del dominio se referencia a un PatronClase *ConectorWebIM* del mismo nombre *nc*, que el *ConectorWebCM* del que se parte. Seguidamente, se crea un *ConectorWebIM* con dos extremos FinalConector1M1

con los puertos de cada módulo. Las instancias PuertoWebIM a priori no son creadas, puesto que la transformación ya las ha creado en la relacion *SituarPuertoWebEnModulo*, por lo tanto se hace uso del estereotipo <<clave>> para los nombres de estos puertos y comprobar así la existencia de dichas instancias PuertoWebIM. En el caso de que la instancia de PuertoWebIM tenga el mismo nombre, ya sea *porigin* o *ptarget*, entonces no es necesaria su creación. Por lo contrario, si no existe un PuertoWebIM con el mismo nombre de origen o destino entonces se creará el correspondiente puerto.

Una vez ejecutada T1_2, el modelo de integración es un conjunto de módulos vacíos que disponen de los correspondientes puertos y conectores entre ellos. A partir de aquí, el siguiente paso consiste en completar los módulos con los componentes de integración. Esa es la tarea de las siguientes transformaciones.

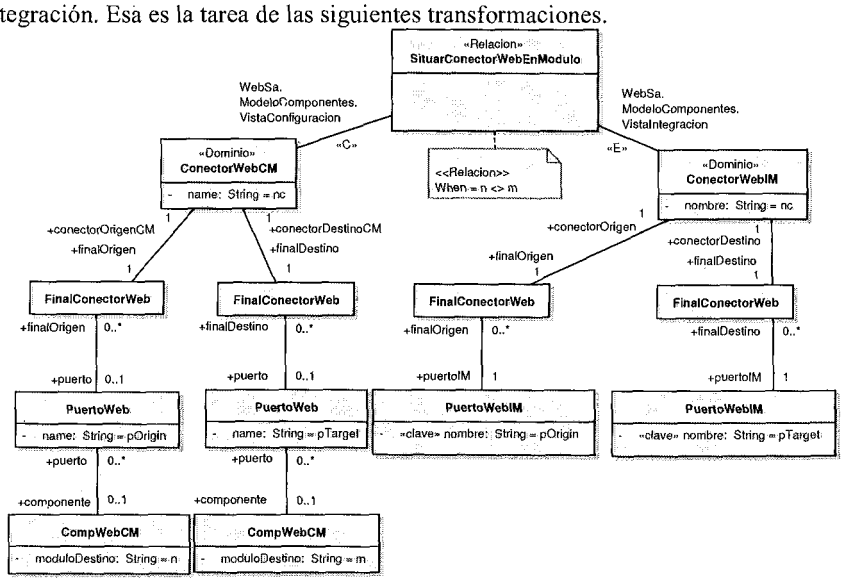

Figura. 101. Relación SituarConectorWebEnModulo

A5.3 T1_3: Creación de Componentes Independientes

T1_3 se inicia con la creación de los componentes de integración que son independientes de la funcionalidad y su colocación en los módulos. Es decir, aquellos componentes que únicamente tienen información procedente de la vista de arquitectura. Además, junto al propio componente se crean propiedades como atributos, operaciones y puertos. Y por último, las posibles relaciones entre los componentes de este tipo mediante interfaces y conectores. El resto de relaciones son creadas en posteriores transformaciones.

La Figura. 63 muestra el mapa de composición de la transformación T1_3. A continuación, se muestran las relaciones de mayor relevancia o que pueden aportar aspectos nuevos que no hayan sido tratados en transformaciones previas.

La primera relación descrita es la relación raíz *SituarCompWebCMIndyEnModulos* (ver Figura. 102). El primer PatronClase del dominio checkonly es el modelo de configuración. El MC referencia al conjunto de instancias de *CompWebCM* que cumplen la restricción que se ha fijado en el When, es decir, que las instancias de *CompWebCM* sean de alguno de los tipos de los llamados componentes independientes (*Almacén, CacheWeb, LibreriaFuncional, PaginaFuncional, PaginaEstilo, VistaLegada, Recurso* y *GestorConexiones*).

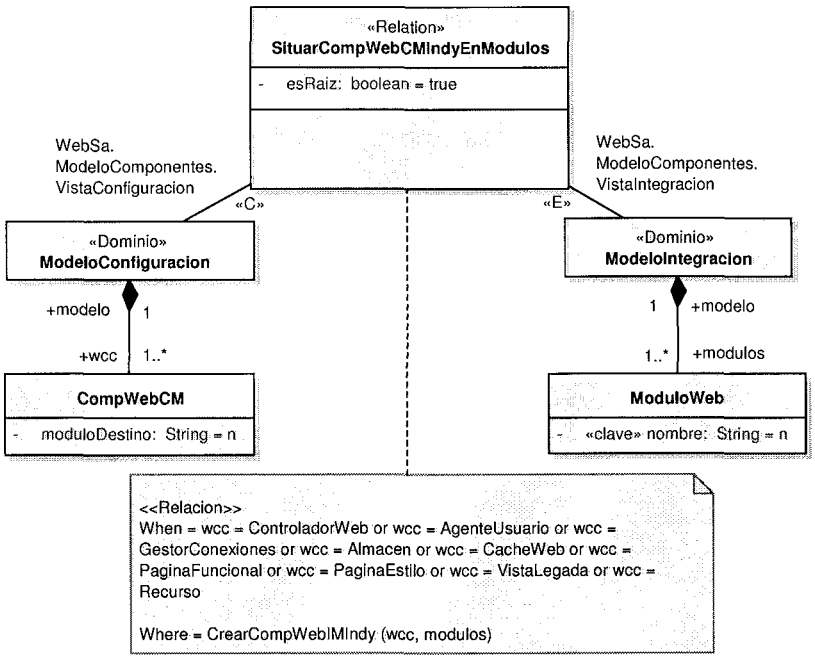

Figura. 102. Relación SituarCompWebCMIndyEnModulos

La relación consulta el atributo *moduloDestino* de cada uno de los componentes y lo almacena en la variable *n*. Seguidamente, en el dominio enforceable su PatronClase es una instancia de ModeloIntegración, este modelo contiene un conjunto de instancias ModuloWeb que tienen como clave principal el *nombre*. El *nombre* se iguala a *n* que es el nombre del moduloDestino del componente, y así la relación solo restringe los módulos a un subconjunto que son referenciados por los componentes independientes. Por último, en la parte Where se invoca a la relación *CrearCompWebIMIndy* pasándole cada uno de los componentes y módulos consultados.

La siguiente relación por orden de importancia en T1_3 es *SituarPropiedadesCompWebIMIndy* (Ver Figura. 103). Esta relación se encarga de crear las instancias de cada una de las propiedades del componente de integración a partir de las propiedades del componente de configuración. En la parte izquierda, el dominio checkonly tiene como PatronClase a una instancia de CompWebCM, la cual contiene un nombre que se almacena en la variable *nc*, y además, referencia a un

conjunto de instancias de *OperacionWebCM* y *PuertoWeb*. Como se puede apreciar, cada una de estas propiedades tiene una cardinalidad cero o muchos, permitiendo que estos componentes puedan tener o no cualquiera de dichas propiedades de forma alternativa sin que afecte a la ejecución de la relación. En la parte derecha, el dominio enforceable tiene como PatronClase a una instancia CompWebIM, que ya se había creado en la relación anterior, a dicha instancia se le introduce el nombre desde la variable *nc* y además se procede a crear cada una de las propiedades del componente de integración a partir de las propiedades del componente de configuración. Una vez creadas las instancias de las propiedades, la relación procede a la invocación de las diferentes relaciones que introducirán los valores, se invocará a *SituarPropiedadIMEnComp*, *SituarOperacionIMEnComp*, y finalmente *SituarPuertoIMEnCmp*, a cada una de ellas se le envía la instancia de la propiedad del componente de configuración y la instancia de la propiedad del componente de integración (p.e. operacionesCM, operacionesIM).

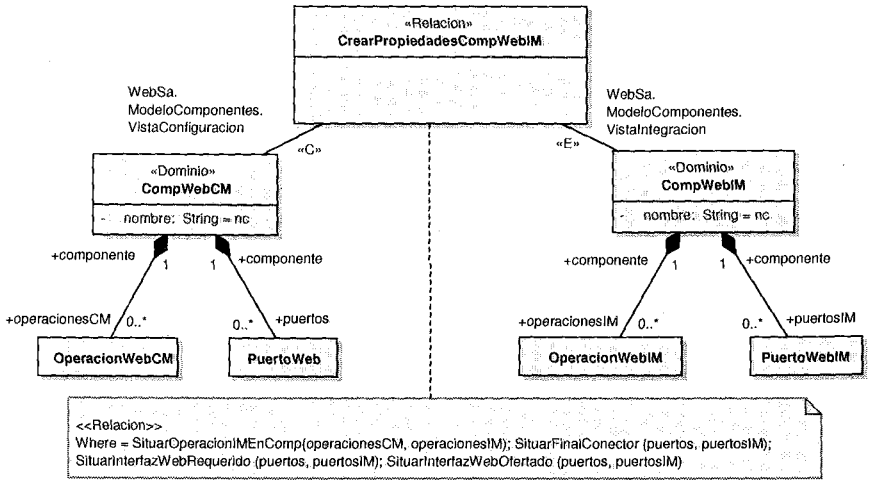

Figura. 103. Relación SituarPropiedadesCompWebIMIndy

La última relación de mención en T1_3 es *CrearConectorWeb* (ver Figura. 104) que establece una conexión entre dos componentes de integración independientes. Se restringe en el dominio checkonly que la instancia de ConectorWeb relacionada con el PuertoWeb, esté a su vez relacionada con un CompWebCM (de rol wcc) que sea de alguno de los tipos de los componentes que fija la parte When (*ControladorWeb, Almacén*, etc.). Si esto se cumple, en el dominio enforceable se creará un ConectorWebIM a partir del ConectorWeb indicando que el atributo nombre es clave (con el estereotipo <<key>>), para que no se cree un conector ya creado. Es importante remarcar que en la creación de los elementos FinalConectorWeb se copian los atributos cardSuperior y cardInterior, para que se mantenga la misma cardinalidad en el MI. Los nombres de los elementos PuertoWeb, son copiados a los nuevos elementos PuertoWebIM que se crean en el MI.

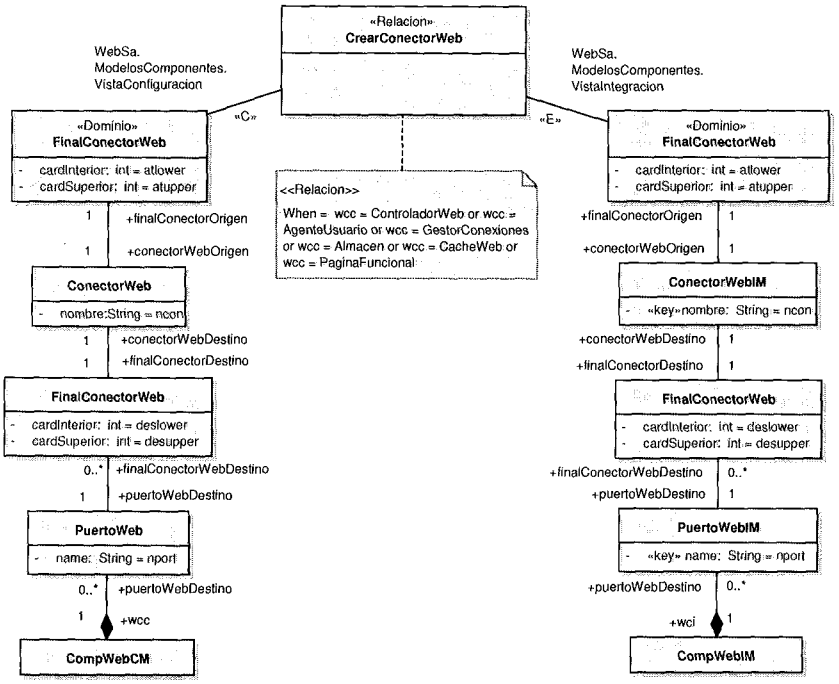

Figura. 104. Relación CrearConectorWeb

A5.4 T1_4: Creación de los Componentes Dependientes de Dominio

La transformación T1_4 procede a la creación de los componentes de integración que están vinculados con la funcionalidad proporcionada por el dominio del problema. Se establece por lo tanto que los modelos origen son (1) el modelo de configuración junto con (2) el modelo de dominio y el destino es el (3) modelo de integración. T1_4 según MDA guide [87] es una transformación de tipo fusión (merge). Los componentes Web implicados son del tipo *DatosEntidad, EntidadWeb, ComponenteProceso, Vista* y *CAD*.

La transformación T1_4, como se puede apreciar en el mapa de la Figura. 64 está constituida por 21 relaciones, por ello se aborda únicamente la descripción de las relaciones más importantes o que introducen relaciones con aspectos no tratados hasta ahora.

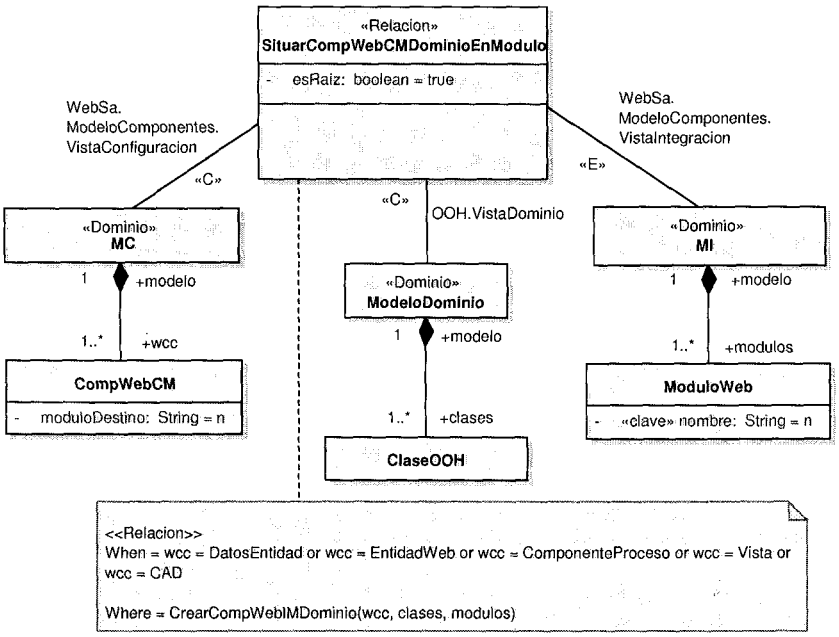

Figura. 105. Relación SituarCompWebCMDominioEnModulo

La primera relación por su importancia es la relación raíz, *SituarCompWebCMDominioEnModulo* (ver Figura. 105), esta relación interroga dos dominios checkonly que parten desde la metaclase que corresponde a las instancias de los modelos consultados. Así, el PatronClase del primer dominio checkonly es el MC y el segundo es una instancia de *ModeloDominio* de OO-H. Se comienza con el *MC* que referencia a un conjunto de instancias de *CompWebCM* que cumplen la restricción que se fija el When, es decir, que CompWebCM sea alguno de los tipos especificados previamente (*DatosEntidad, EntidadWeb, ComponenteProceso, Vista* y *CAD*). Estos componentes dependen de la funcionalidad proporcionada por el modelo de dominio para completar su información. El segundo dominio hace referencia a una instancia de *ModeloDominio* el cual contiene un conjunto de instancias *ClaseOOH*, que van a ser recorridas para extraer su información en posteriores relaciones. Respecto al dominio enforceable referencia a una instancia de *ModeloIntegracion* que contiene un conjunto de instancias de *ModuloWeb* con el mismo nombre que el campo moduloDestino de los componentes.

Finalmente, se invoca a la relación *CrearCompWebIMDominio* a la que se les pasa el conjunto de componentes, clases y módulos referenciados. Esta relación tiene como dominios checkonly a un *CompWebCM* y a una *ClaseOOH*, mientras que como dominio enforceable tiene a un *ModuloWeb*, a partir del cual se crea una instancia *CompWebIM*. Una vez creada la instancia de *CompWebIM*, se invoca a la siguiente relación que se encargará de introducir sus propiedades.

La siguiente relación es *SituarPropiedadesCompWebIMDominio* (Ver Figura. 106), esta relación tiene la tarea de gestionar la introducción de todas las propiedades

de un componente de integración. Para ello, comienza con la introducción del atributo *nombre* que es propiedad que identifica al CompWebIM creado. Este nombre se crea con la concatenación del nombre del CompWebCM y del nombre de la instancia de la ClaseOOH. Además, una vez realizada esta tarea, la parte Where se encarga de invocar a las relaciones que introducen el resto de propiedades del *CompWebIM*, para la creación de operaciones *CrearOperacionesDeDominio*, creación de atributos y sus métodos de acceso con *CrearPropiedadesIMDeDominio*, crear sus puertos con sus interfaces y conectores llamando a *CrearPuertosIMEnComp*.

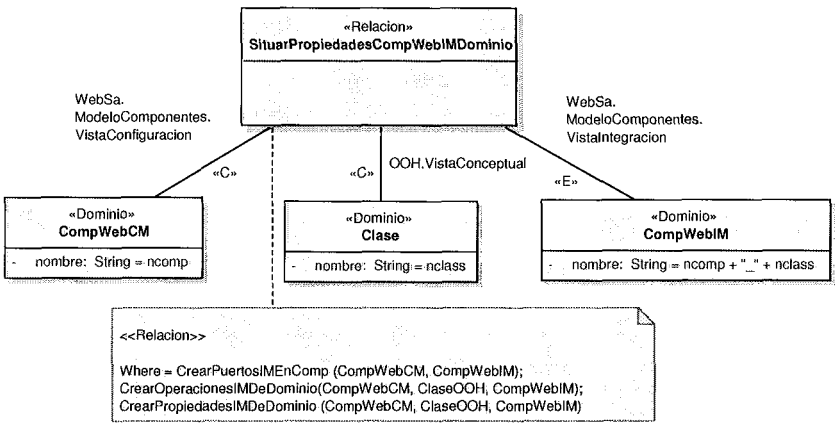

Figura. 106. Relación SituarPropiedadesCompWebIMDominio

Considerando que la misión principal de la transformación T1_4 es obtener la información del modelo de dominio para introducirla en los componentes de integración, la siguiente relación describe la obtención de los atributos del modelo de dominio para la creación de propiedades dentro de los componentes. La relación es *CrearPropiedadesDominioIM* (Ver Figura. 107), esta relación contiene dos dominios checkonly el modelo de configuración y el modelo de dominio. En el modelo de configuración, el PatronClase apunta a una instancia de CompWebCM, la cual debe poseer dos atributos con valores concretos. Por un lado, *tieneAtributos* debe poseer el valor *true* (verdadero) indicando así que el componente tiene atributos, y por otro lado, *tipoAtributos* debe contener el valor *Dominio*, indicando así que los atributos son del tipo dominio. En el otro dominio checkonly situado en el *ModeloDominio*, el PatronClase referencia a una instancia de la *ClaseOOH*. Esta clase debe contener una o más instancias de Atributo. Si los dos dominios checkonly se cumplen, se procede a ejecutar el dominio enforceable localizado en la vista de integración, cuyo PatronClase referencia a una instancia de *CompWebIM*. A la instancia de *CompWebIM* se le asocia un conjunto de instancias *PropiedadWebIM*, las cuáles tienen una relacion 1 a 1 con las instancias de *Atributo* del Modelo de Dominio. Además, se crea un conjunto de *OperacionWebIM*, que corresponden a las instancias de las operaciones de acceso a las *PropiedadesWebIM*. Por cada instancia de *Atributo* de dominio, se crea un *OperacionWebIM* de tipo *Set* y otro de tipo *Get*, que permiten dar valor a la *PropiedadWebIM* y obtener el valor respectivamente. Se crea también

un conjunto de instancias *PuertoWebIM* que serán obtenidas para la creación de los atributos que se correspondan a roles de asociación. Y por último, se crea un conjunto de instancias *ParteWebIM* que corresponden a roles que pertenecen a relaciones de composición. Una vez comprobada la relación, se procede a la ejecución de la parte Where donde se tienen las invocaciones a las relaciones que se encargan de introducir las propiedades de las instancias creadas en esta relación.

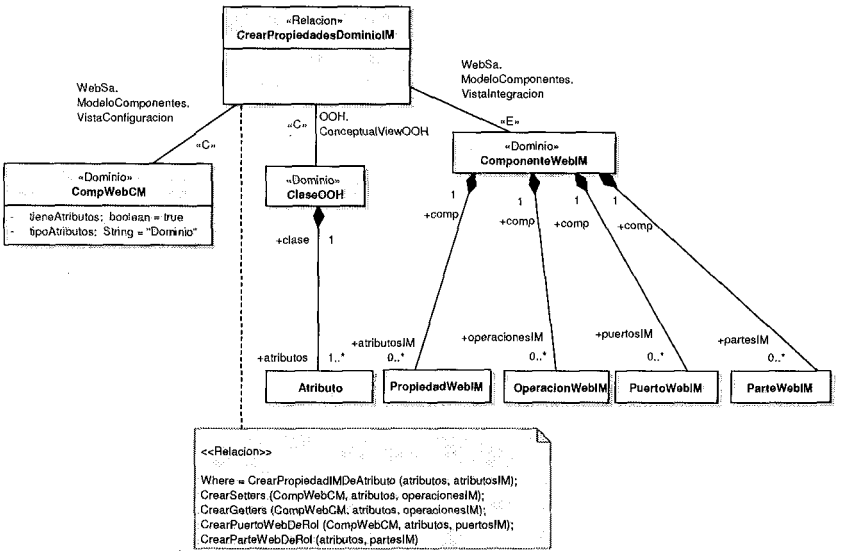

Figura. 107. Relación CrearPropiedadesDominioIM

Primero se invoca a la relación *CrearPropiedadIMDeAtributo* que a partir del nombre y el tipo de la instancia de la metaclase *Atributo* del dominio, procederá a crear el nombre y el tipo de *PropiedadWebIM*. A continuación se invoca a *CrearSetters* que procederá a crear un *OperacionWebCM* en el componente de integración, el cual va a ser una operación que recibe un parámetro con el mismo tipo que el atributo de dominio, y no va a tener valor de retorno. Seguidamente se invoca a la relación *CrearGetters* que se encarga de crear un *OperacionWebCM* en el componente de integración, que tiene un parámetro de salida del mismo tipo que el atributo de dominio. En tercer lugar se invoca a la relación *CrearPuertoWebDeRol* que recibirá el componente de configuración, los atributos y las instancias de PuertoWebIM que serán completadas en dicha relación. En último lugar se invoca a la relación *CrearParteWebDeRol* que recibirá el atributo que se corresponde a un rol de una composición y a la instancia de la *ParteWeb* que será completada en dicha relación.

La última relación de la transformación T1_4 que se describe en este capítulo es *CrearOperacionIMDeOperación* (ver Figura. 108), el motivo es muestra la importante tarea de la inserción de la información proveniente de la *Operacion* del modelo de dominio en los componentes del modelo de integración. Esta relación se encarga de convertir un *OperacionWebCM* de tipoComportamiento *Dominio* definido en un componente del modelo de configuración, en un conjunto de *OperacionWebIM*

de un componente del modelo de integración, cada uno de los cuáles son creados a partir de los valores proporcionados por el conjunto de instancias *Operacion* que contiene la *ClaseOOH*.

El primer dominio checkonly localizado en la vista de configuración referencia a una instancia de *OperacionWebCM*. El dominio almacena el nombre de la operación y comprueba que el atributo *tipoComportamiento* sea igual a *Dominio*. En el segundo dominio checkonly localizado en la vista de dominio de OOH tiene como PatronClase una *Operación* de la cual se almacena su nombre y su tipo. Una vez comprobados los dominios checkonly, se procede a crear en el dominio enforceable localizado en la vista de integración. Una instancia de *OperacionWebIM* contiene el nombre de la *Operación* y posee también un conjunto de parámetros, cada uno de los cuáles se crean a partir de los parámetros definidos por la *Operación*. Una vez creados los parámetros, la relación invoca por cada uno de ellos a la relación *CreaParametroIM* que asigna el nombre y el tipo del al *ParametroWeb* del *OperacionWebIM*.

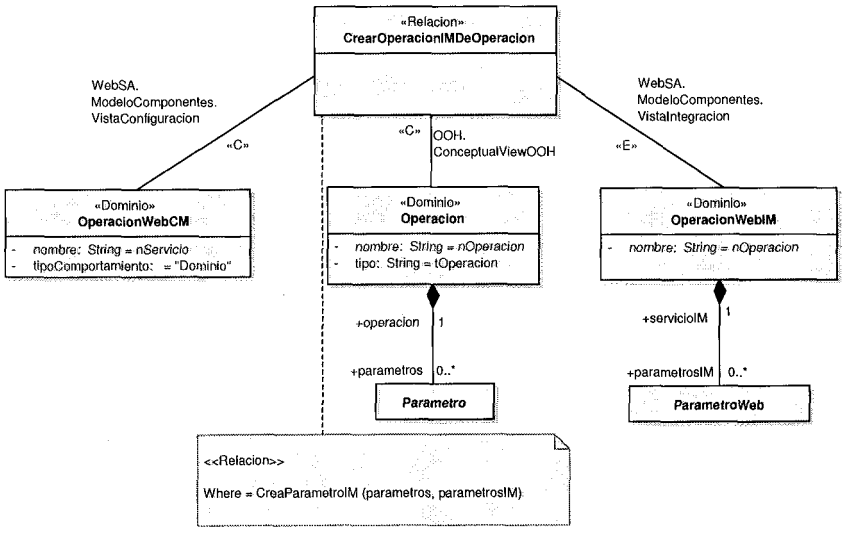

Figura. 108. Relación CrearOperacionIMDeOperación

A5.5 T1_5: Creación de los Componentes Dependientes de Navegación

La transformación T1_5 procede a la creación y ubicación en módulos de los componentes de integración que están vinculados con la funcionalidad proporcionada por la navegación. Los modelos origen son (1) el modelo de configuración junto con (2) el modelo de navegación y el modelo destino es el modelo de integración. Las transformaciones mostradas en esta sección se centran en la integración con el modelo de navegación llamado DAN definido por OO-H [19]. El modelo de navegación de OO-H proporciona la información necesaria para crear aquellos componentes de integración y las propiedades que obtienen información desde la funcionalidad proporcionada por la navegación, se trata por lo tanto de una transformación de tipo

fusión (merge). Los componentes de configuración son PaginaEstatica, PaginaServidor y ObjetoWeb. A la hora de proceder a su creación en el modelo destino, no solamente se completa los componentes de integración sino que se creará un componente de integración por cada instancia de ClaseNavegacional del modelo de navegación. La transformación T1_5, como muestra Figura. 65 contiene por 22 relaciones, por ello en este capítulo se realiza la descripción de las relaciones más importantes o que introducen relaciones con características no tratadas.

La primera relación por su importancia en la transformación T1_5, es la relación raíz *SituarCompWebCMNavEnModulo* (ver Figura. 109), esta relación interroga dos dominios checkonly que parten desde la metaclase que corresponde a las instancias de los modelos consultados. Así, el PatronClase del primer dominio checkonly es el modelo de configuración y el segundo PatronClase es una instancia del modelo de navegación de OO-H. Comenzando con el modelo de configuración, esta instancia referencia al conjunto de instancias de *CompWebCM* que cumplen la restricción que fija la clausula When, es decir, que *CompWebCM* sea alguno de los tipos especificados previamente (*PaginaServidor, ObjetoWeb o PaginaEstatica*), estos componentes dependen de la funcionalidad proporcionada por la navegación para completar su información. El segundo dominio que hace referencia a una instancia de *ModeloNavegación* que contiene un conjunto de instancias de *NodoNavegacional*, que van a ser recorridas para extraer su información en posteriores relaciones. Respecto al dominio enforceable el *PatronClase* referencia a una instancia de *ModeloIntegracion*, que contiene un conjunto de instancias de *ModuloWeb*, que tienen el mismo nombre que el campo *moduloDestino* de los componentes.

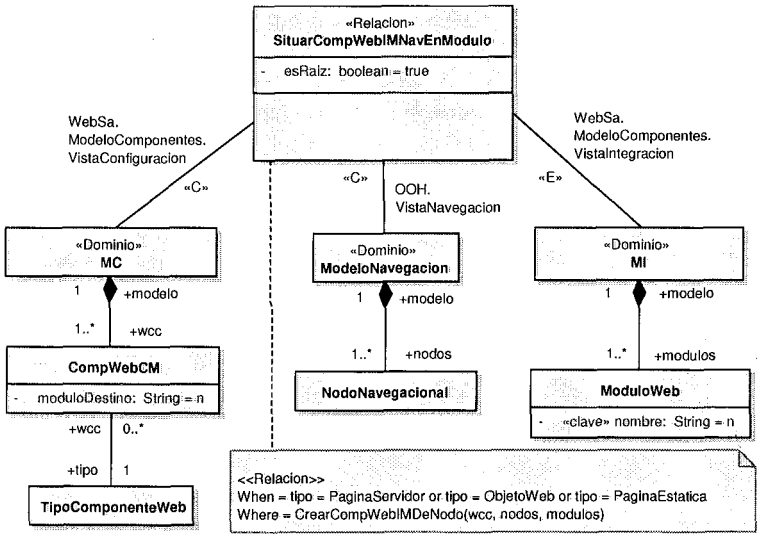

Figura. 109. Relación SituarCompWebCMNavEnModulo

Finalmente se invoca a dos relaciones *CrearCompWebIMDeNodo* a la cual se les pasa el conjunto de componentes, nodos de navegación y módulos referenciados. Estas

relaciones tienen como dominios checkonly a un *CompWebCM* y a un *NodoNavegacional*, mientras que como dominio enforceable tiene a un *ModuloWeb*, a partir del cual se crea una instancia *CompWebIM*. Una vez creada la instancia de *CompWebIM*, invocará a la siguiente relación que se encargará de introducir las propiedades de la *ClaseNavegación*.

La siguiente relación es *SituarPropiedadesCompWebIMNav* (ver Figura. 110) tiene la tarea de gestionar la introducción de las propiedades que provienen de un nodo navegacional en un componente de integración. Para ello, comienza con la introducción del atributo *nombre* que es la propiedad que identifica al *CompWebIM* creado. El nombre del *CompWebIM* se crea con la concatenación del nombre del *CompWebCM* y del nombre de la instancia de *NodoNavegacion*. Además, una vez realizada esta tarea, la parte Where se encarga de invocar a las relaciones que introducen el resto de propiedades. La creación de operaciones con *CrearServiciosNav*, creación de atributos y sus métodos de acceso con *CrearPropiedadesNav* y crear sus puertos con sus conectores e interfaces con *CrearPuertosIMEnComp*.

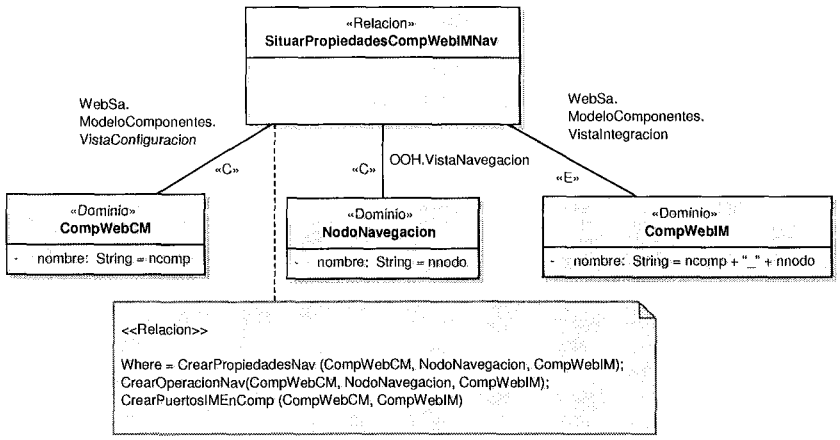

Figura. 110. Relación SituarPropiedadesCompWebIMNav

Considerando que la peculiaridad principal que proporciona un modelo de navegación es el establecimiento de enlaces navegacionales entre los nodos, las siguientes relaciones que se describen abordan la creación de propiedades a partir de las instancias de EnlacesNavegacion.

La relación *CrearOperacionWebDeEnlaceNav* (ver Figura. 111) está encargada de crear un *OperacionWeb* en un componente de integración, por cada uno de las instancias *EnlaceNavegacion* definidas en el modelo de navegación que naveguen a un nodo diferente, es decir, su propiedad *esMismoNodo=false*. La relación define como dominios checkonly el MC y el Modelo NAD de OOH. En MC tiene como PatronClase a una instancia de la metaclase *OperacionWebCM* en la cual se comprueba que su propiedad *tipoComportamiento=navegación*. En el ModeloNavegacion de OO-H contiene como PatronClase una instancia de EnlaceNavegacional que contiene un conjunto de atributos como *nombre,*

filtroOrigen, filtroDestino y *esMismoNodo* = *false* (indica que el enlace de navegación salta a un nodo diferente). Respecto al dominio enforceable el PatronClase referencia a una instancia de *OperacionWebIM* que contiene el atributo *nombre* al que será asignado el *nombre* del *EnlaceNavegacional*. Además, contiene un conjunto de *ParametrosWebIM* que serán creados para contener diferentes propiedades que provienen del *EnlaceNavegacional*. Para crear cada uno de los parámetros en la parte Where se invoca a las relaciones *CrearParamNodoOrigen, CrearParamFiltroOrigen y CrearParamFiltroDestino.* Cada uno de estas relaciones leerán del *EnlaceNavegacional* el valor para cada uno de sus atributos y creará un *ParametroWeb* de entrada, con el nombre y el tipo de cada propiedad.

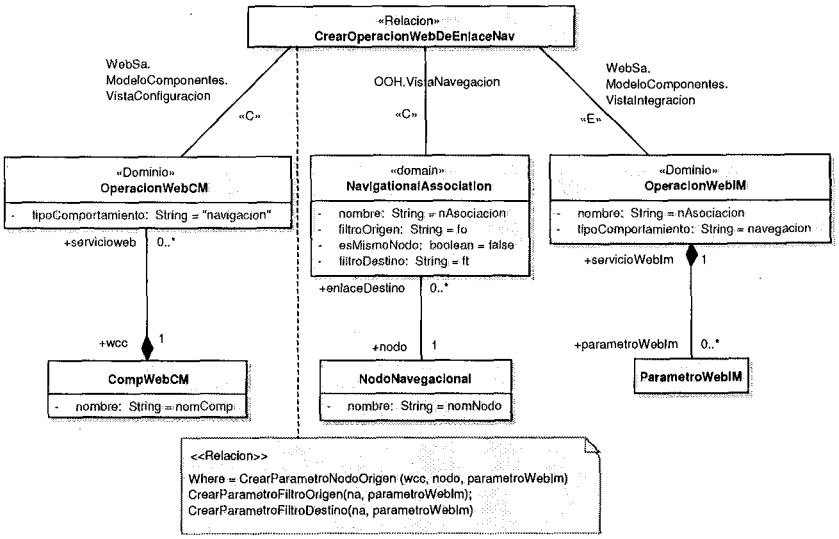

Figura. 111. Relación CrearOperacionWebDeEnlaceNav

Por último, se describe la relación *CrearParteWebDeEnlaceTrav* (ver Figura. 112) encargada de convertir un EnlaceNavegacional en una instancia de ParteWeb en el modelo de navegación. Para ello, se hace referencia a dos dominios checkonly, MC y el modelo de navegación de OOH. Por un lado, MC contiene como PatronClase a una instancia de *OperacionWebCM* que tiene como *tipoComportamiento= navegación.* Además, *OperacionWebCM* se relaciona con un *CompWebCM* del que se obtiene su *nombre.* Por otro lado, en el modelo de navegación el PatronClase es la metaclase *EnlaceNavegacional* el cual referencia a dos atributos, el *nombre* y *esMismoNodo* del cual se comprueba que sea igual a *true* (verdadero). El *EnlaceNavegacional* se relaciona a su vez con el *NodoNavegacional* destino de la navegación. Mientras en el dominio enforceable el PatronClase es la instancia de la metaclase ParteWeb. La relación creará un *ParteWeb* con el nombre del *EnlaceNavegacional*, además, *ParteWebIM* tendrá una referencia al tipo de componente Web al cual representa dicha ParteWeb. Para ello, la relación establece una relación entre la *ParteWeb* y un

CompWebIM cuyo nombre sea el resultado de la concatenación *nombre = nomComp + "_"+ nomNodo*.

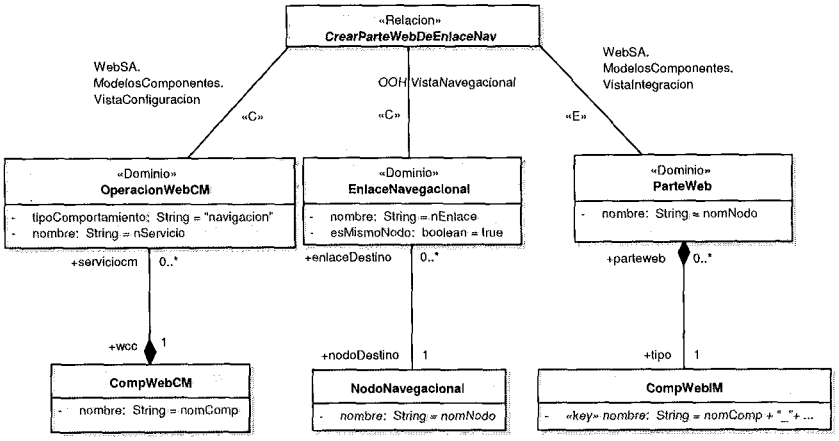

Figura. 112. Relación CrearParteWebDeEnlaceNav

Volver al índice/Tornar a l'índex

Referencias

1. Agrawal A. Metamodel Based Model Transformation Language to Facilitate Domain Specific Model Driven Architecture. 18th annual ACM SIGPLAN conference on OOPSLA. October 2003
2. ArgoUML. ArgoUML Project Home. http://argouml.tigris.org. 2002
3. ArgoUWE. A CASE Tool for Modeling Web Applications. http://www.pst.informatik.uni-muenchen.de/projekte/uwe/argouwe.shtml
4. Atkinson C., Kühne T., Henderson-Sellers B. Systematic stereotype usage, Software and System Modeling 2 (3), 153-163, 2003
5. ATL GMT subproject. http://www.eclipse.org/gmt/atl. 2006
6. Bachmann F., Bass L., Chastek G., Donohoe P., Peruzzi F. The Architecture Based Design Method. CMU/SEI-2000-TR-001, 2000
7. Baresi L., Garzotto F., Maritati M. W2000 as a MOF Metamodel. In Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics - Web Engineering track. Orlando (USA), July 2002
8. Baresi, L., Garzotto, F., & Paolini, P. 2001. Extending UML for Modeling Web Applications. Proceedings of the 34th International Conference on System Sciences, 2001
9. Baresi, L., Mainetti, L. Beyond Modeling Notations: Consistency and Adaptability of W2000 Models. In Proc. of SAC'05, ACM Symposium on Applied Computing, Santa Fe, USA, 2005
10. Bass L., Klein M., Bachmann F. Quality Attribute Design Primitives, CMU/SEI-2000-TN-017, Carnegie Mellon, Pittsburgh, December 2000
11. Beck K., Andres C. Extreme Programming Explained. Pearson Books. Nov 2004
12. Bézivin J. In Search of a Basic Principle for Model Driven Engineering. Novática nº1, 21-24, June 2004
13. Bochicchio M.A., Paiano R., Paolini P. Jweb: An HDM Environment for Fast De-

velopment of Web Applications. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems. IEEE Computer Society, 1999

14. Boehm BW. Characteristics of software quality. North-Holland Pub. Co., Amsterdam, New York 1978
15. Booch G. The Architecture of Web Applications, Developer Works: IBM developer solutions, June 2001
16. Booch G., J. Rumbaugh and I. Jacobson. The Unified Modelling Language User Guide. Addison-Wesley, 1999
17. Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M. Pattern-Oriented Software Architecture – A System of Patterns, John Wiley & Sons Ltd. Chichester, England, 1996
18. Cáceres P., Marcos E., Vela B. A MDA-Based Approach for Web Information System, Workshop in Software Model Engineering, WisME 2004
19. Cachero C. OO-H. Una extensión de los métodos OO para el modelado ygeneración automática de interfaces hipermediales. Available online at http://www.dlsi.ua.es/~ccachero/pTesis.htm. 2003. (In Spanish)
20. Ceri S., Fraternali P., Matera M. Conceptual Modeling of Data-Intensive Web Applications, IEEE Internet Computing 6, No. 4, 20–30, July/August 2002
21. Clements P. C. and Northrop L. N. Software Architecture: An Executive Overview. In Brown, 55-68, 1996
22. CMU SEI CMM. Software, Systems Engineering, and Product Development Capability Maturity Models, August 2000, Available at www.sei.cmu.edu/cmm/cmms/transition.html
23. CMU SEI. How Do You Define Software Architecture? 2000. Web site maintained by Carnegie Mellon University, Software Engineering Institute. Available at www.sei.cmu.edu/architecture/definitions.html
24. Conallen J. Building Web applications with UML Second Edition. Addison Wesley Longman. September 2002
25. Cook S. and Daniels D. Designing Object Systems: Object-Oriented Modelling with Syntropy. Prentice Hall, 1994
26. Dobing D., Parsons J. Current Practice in the Use of UML. Perspectives in Conceptual Modeling: ER 2005, BP-UML, Klagenfurt, Austria, October 2005
27. Eclipse consortium. Eclipse – Home page. www.eclipse.org
28. Ermel C., Rudolf M., and Taentzer G. The AGG approach: Language and tool environment. In G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Appli-cations, Languages, and Tools, pages 551 – 601. World Scientific, 1999

29. Escalona M.J., Reina A.M., Torres J., Mejías M.. NDT: A methodology to deal with the navigational aspect at the requirements phase. Proceedings of the Workshop in Aspect-Oriented Requirements Engineering and Architecture Design. Vancouver, Canada. October, 2004

30. Escalona, M. J. and Koch, N. Metamodeling Requirements of Web Systems. In Proc. 2nd Int. Conf. on Web Information System and Technologies, Portugal, April 2006

31. Fielding R., Taylor R. Principled Design of the Modern Web Architecture, ACM Transactions on Internet Technology, Vol. 2, No. 2 , 115-150, May 2002

32. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2003

33. France R., Bieman J.M. Multi-view Software Evolution: A UML-based Framework for Evolving Object-Oriented Software. Proceedings of the International Conference on Software Maintenance (ICSM 2001), November 2001

34. Fraternali, P., Paolini, P. A Conceptual Model and a Tool Environment for Developing more Scalable, Dynamic, and Customizable Web Applications. 1998

35. Gaedke, M., Lyardet, F., & Werner-Gellersen, H. 1999. Hypermedia Patterns and Components for Building better Web Information Systems. In: ACM Conference on Hypertext and Hypermedia

36. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

37. Garlan D. ACME: Architectural Description of Component Based Systems. Foundations of Component-Based Systems, Cambridge University Press, 47-68, 2000

38. Garlan D., Shaw M. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994

39. Garrigós I., Gómez J., Barna P., Houben G.J.: A Reusable Personalization Model in Web Application Design. International Workshop on Web Information Systems Modeling (WISM 2005) (Held in conjunction with ICWE 2005). Sydney, Australia. 2005

40. Garzotto, F., Paolini, P., Schwabe, D. HDM A Model-Based Approach to Hypertext Application Design. ACM Transactions on Information Systems (TOIS), 11(1), 1–26. 1993

41. Greenfield J., Short K. Software factories Assembling Applications with Patterns, Models, Frameworks and Tools. OOPSLA '03, Anaheim, Ca., USA, 2003

42. Gómez J., Cachero C., Pastor O. Conceptual Modeling of Device-Independent Web Applications. IEEE Multimedia, 8(2), 26–39, 2001

43. Hassan A., Holt R. Architecture Recovery of Web Applications, International Conference on Software Engineering (ICSE'02), May 2002

44. Hayes-Roth B. A Domain-Specific Software Architecture for a Class of Intelligent Patient Monitoring Agents. Computer Science Department Stanford University. March, 1994
45. Hofmann C., Eckart H., Keller H., Renzel K., Schmidt M. The field of software architecture. November 1996
46. Hofmeister C., R. Nord L., Soni D. Siemens Corporate Research, Princeton, New Jersey, USA. 1999
47. IBM. IBM Rational Unified Process (RUP). Http://www.rational.com/products/rup/index.jsp, 2003
48. Isakowitz, T., Stohr, E. A., Balasubramanian, V. RMM: A Methodology for Structured Hypermedia Design. CACM: Communications of the ACM, 08, 34–44. 1995
49. ITU. Specification and Description Language (SDL) Z.100, Available at::http://www.itu.int/ITUT/studygroups/com10/languages/Z.100_1199.pdf, November, 1999
50. Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process. Addison-Wesley. 1999
51. Jacyntho M.D., Schwabe D., Rossi G. A Software Architecture for Structuring Complex Web Applications. Journal of Web Engineering, 1(1):37-60, 2002
52. Java Metadata Interface (JMI). http://java.sun.com/products/jmi/. 2006
53. Jobs S. The Next Insanely Great Thing. The Wired Interview. 1996
54. Kappel G., Pröll B., Reich S., Retshitzegger W. Web Engineeering. John Wiley & Sons. 2006
55. Kent S. Model Driven Engineering. IFM 2002, LNCS 2335, pp.286 –298, 2002
56. Kleppe A. Models and Transformations in an Increasingly Complex Software World, or the Awakening of Sleeping Beau. Keynote speech. European Workshop on Milestones, Models and Mappings for Model-Driven Architecture. 2006.
57. Kleppe A., Warmer J., Bast W. MDA Explained. The Model Driven Architecture, Practice and Promise, Addison-Wesley, 2003
58. Koch N. Transformation Techniques in the Model-Driven Development Process of UWE. 2nd International Workshop on Model-Driven Web Engineering (MDWE 2006). 6th International Conference on Web Engineering (ICWE 2006). July 2006
59. Koch N., A. Kraus, Cachero C., Meliá S. Integration of Business Processes in Web Aplication Models. Journal of Web Engineering (JWE). Vol.3, 2004
60. Koch N., Cachero C., Kraus A., Meliá S. Modeling Web Business Processes with OO-H and UWE. Proceedings of 3rd International Workshop on Web Oriented Software Technology,Oviedo, Spain, July 2003

61. Koch, N. Software Engineering for Adaptative Hypermedia Applications. Ph. Thesis, FAST Reihe Softwaretechnik Vol(12), Uni-Druck Publishing Company, Munich. Germany. 2001

62. Koch, N., Kraus, A. Towards a Common Metamodel for the Development of Web Applications. In Third International Conference on Web Engineering (ICWE 2003). LNCS 2722, ©Springer Verlag, 497-506, July 2003

63. Krobyn C. UML 3.0 and the Future of Modeling, Software and System Modeling, Vol. 3, No. 1, 4-8, 2004

64. Kruchten P. The 4+1 View Model of Architecture, IEEE Software. 1995

65. Lange, D. An Object-Oriented Design Approach for Developing Hypermedia Information Systems. Journal of Organizational Computing and Electronic Commerce, 3(6), 269–293. 1996

66. Lewis G.A., Wrage L. Problems in Technologies for Interoperability: Model-Driven Architecture. Technical Note CMU/SEI-2005-TN-022. May 2005

67. List B., Korherr B. A UML 2 Profile for Business Process Modelling. 1st Workshop in Best Practices in UML, ER 2005, October 2005

68. Manifesto for Agile Software Development. http://agilemanifesto.org

69. Medvidovic N. Architecture-Based Specification-Time Software Evolution. PhD thesis, University of California, Irvine, 1999

70. Meinecke J., Gaedke M., Nussbaumer M.. A Web Engineering Approach to Model the Architecture of Inter- Organizational Applications. COEA'05, 125-137, 2005

71. Meliá S., Cachero C. An MDA Approach for the Development of Web Applications. Web Engineering. Lecture Notes in Computer Science. © Springer-Verlag. Vol. 3140/2004. ISSN: 0302-9743. JCR-2004 Impact Factor: 0,513. 4th International Conference on Web Engineering (ICWE'04). Munich, Germany, July 2004

72. Meliá S., Cachero C., Gómez J. Using MDA in Web Software Applications. 2nd International Workshop in Generative Techniques in the context of MDA. OOPSLA 2003. Anaheim, California, EEUU, October 2003

73. Meliá S., Gomez J. Applying Transformations to Model Driven Development of Web applications. Perspectives in Conceptual Modeling. Lecture Notes in Computer Science © Springer-Verlag. Volume 3770/2005. ISSN: 0302-9743. JCR-2005 Impact Factor: 0,402. 1st International Workshop on Best Practices of UML. (ER, 2005), Austria, October 2005

74. Meliá S., Gomez J. Applying WebSA to a case study: A travel agency system. 1st International Workshop on Model-Driven Web Engineering (ICWE, 2006).Sydney, Australia, July 2005

75. Meliá S., Gomez J. The WebSA Approach: Applying Model Driven Engineering to Web Applications. Journal of Web Engineering, © Rinton Press. Vol. 5, No.2, 121-149, 2006

76. Meliá S., Gomez J. UPT: A Graphical Transformation Language based on a UML Profile. Proceedings of European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA 2006). 2nd European Conference on Model Driven Architecture (EC-MDA 2006). July 2006

77. Meliá S., Gomez J., Koch N. Improving Web Design Methods with Architecture Modeling. E-Commerce and Web Technologies. Lecture Notes in Computer Science © Springer-Verlag. Volume 3590/2005. ISSN: 0302-9743. JCR-2005 Impact Factor: 0,402. 6th International Conference on Electronic Commerce and Web Technologies (ECWeb, 2005). Copenhagen, Denmark, August 2005

78. Meliá S., Kraus A., Koch N. MDA Transformations Applied to Web Application Development. Web Engineering. Lecture Notes in Computer Science. © Springer-Verlag. Volume 3579/2005. ISSN: 0302-9743. JCR-2005 Impact Factor: 0,402. 5th International Conference on Web Engineering (ICWE'05). Sydney, Australia, July 2005

79. Metadata Repository Project HOME (MDR) http://mdr.netbeans.org/ architecture.html. 2006

80. Miller G.G. The Characteristics of Agile Software Processes. The 39th International Conference of Object-Oriented Software and Systems (TOOLS 39). Santa Barbara. 2001.

81. Moreno N., Fraternali P., Vallecillo A. A UML 2.0 Profile for WebML Modeling. 2nd International Workshop on Model-Driven Web Engineering (MDWE 2006). 6th International Conference on Web Engineering (ICWE 2006). July 2006

82. Murugesan S., Deshpande Y., Hansen S., Ginige A. "Web Engineering: A New Discipline for Development of Web-Based Systems." Lecture Notes in Computer Science 2016 Springer 2001, pag 3 – 13.

83. Nilson R., Kogut P., Jackelen, G. Component Provider's and Tool Developer's Handbook Central Archive for Reusable Defense Software (CARDS), STARS Informal Technical Report STARS-VC-B017/001/00, Unisys Corporation, March 1994

84. Noack J., Schienmann B. Introducing Object Technology in a Large Banking Orgainzation, IEEE Software, 71-81, May 1999

85. OMG. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. OMG doc. ad/03-08-02

86. OMG. Common Warehouse Metamodel (CWM) v. , OMG doc. ad/2001-02-03

87. OMG. MDA Guide, OMG doc. ab/2003-05-01

88. OMG. Meta Object Facility (MOF) v1.4, OMG doc. formal/02-04-03

89. OMG. Model Driven Architecture, OMG doc. ormsc/2001-07-01

90. OMG. MOF 2.0 query, views, transformations request for proposals. http://www.omg.org/techprocess/meetings/schedule/MOF2.0QueryViewTransf.RFP.html

91. OMG. MOF to Text Transformation Language Final Adopted Specification. OMG doc. ptc/06-11-01

92. OMG. MOF Query/Views/Transformations Draft Adopted Specification: OMG doc. ptc/05-11-01

93. OMG. Object Constraint Model (OCL). OMG doc. ad/2003-01-07

94. OMG. Second Revised Submission for MOF Model to Text Transformation Language RFP. OMG Doc. ad/04-04-07

95. OMG. UML 2.0 Superstructure Specification: formal/05-07-04

96. OMG. UML Profile for Enterprise Distributed Object Computing Specification. OMG doc. ad/2001-06-09

97. OMG. XML Metadata Interchange (XMI). Specification. Version 2.0. 2003. OMG doc. formal/03-05-02

98. Pressman R.S. "Ingeniería del Software. Un enfoque práctico". Tercera eds. McGraw-Hill. 1992

99. Rational Rose XDE Developer. http://www-306.ibm.com/software/awdtools/developer/rosexde/.2006

100. Renzel K., Keller W. Client/Server Architectures for Business Information Systems. A Pattern Language, PLoP Conference, 1997

101. Retschitzegger W., Schwinger W. Towards Modeling of DataWeb Applications - A Requirement's Perspective. Pages 149–155 of: Proceedings of the American Conference on Information Systems AMCIS 2000, vol. 1. 2000

102. Ronin International. Enterprise Unified Process (EUP). http://www.enterpriseunifiedprocess.info/, 2003

103. Rosenberg, D., Scott, K. Use Case Driven Object Modeling with UML. A Practical Approach. Addison Wesley. 1998

104. Rossi, G. An Object Oriented Method for Designing Hipermedia Applications. PHD Thesis, Departamento de Informática, PUC-Rio, Brazil, 1996

105. Schwabe D., Almeida R., Moura I. Leveraging Template-based Website Implementations Using Design Methods, In Proc. of 8th International World Wide Web Conference, 1999

106. Schwabe D., Almeida R., Moura I. OOHDM-Web: an environment for implementation of hypermedia applications in the WWW. SIGWEB Newsl. 8, 2, 18-34, June 1999

107. Selic B. An Overview of UML 2.0 (Tutorial), UML 2004

108. Selic B., Gullekson G., Ward P. T. Real-Time Object-Oriented Modeling, John Wiley & Sons, New York, 1994

109. Shaw M. and Clements P. C. A Field Guide to Boxology. Preliminary Classification of Architectural Styles for Software Systems. Technical report, Carnegie Mellon University, Software Engineering Institute, April 1996

110. Sommerville I. Software Engineering. Addison-Wesley, 5th ed. 1996

111. Tai H., Mitsui K., Nerome T., Abe M., Ono K. Model-driven development of large-scale Web applications, IBM J. Res. & Dev. Vol. 48 No. 5/6. november 2004

112. Takahashi, K., & Ling, E. Analysis and Design of Web-based Information Systems. Pages 377–389 of: Proc. of Sixth International World Wide Web Conf. Springer-Verlag. Lecture Notes in Computer Science. 1997

113. Texen. Jakarta Proyect. http://jakarta.apache.org/velocity/docs/texen.html

114. The IEEE Architecture Working Group. Draft Recommended Practice for Architectural Description, IEEE P1471/D5.1. Technical report, IEEE, October 1999a. Available at www.pitecanthropus.com/~awg.

115. TM J2EE Blueprint. Java Petstore 1.1.2, http://developer.java.sun.com/developer/releases/petstore/petstore1_1_2.html, November 2004

116. Together2006. http://www.borland.com/us/-products/together/index.html. 2006

117. Together2006. http://www.borland.com/us/-products/together/index.html. 2006

118. Trowbridge D., Mancini D. Enterprise Solution Patterns Using Microsoft .NET. Patterns and Practices. Version 2.0. Microsoft Corp. 2003

119. Troyer, O.M.F. De, Leune, C.J. WSDM: a user centered design method for Web sites. In: Seventh International World Wide Web Conference. Springer-Verlag. Lecture Notes in Computer Science. 1998

120. UML Version 1.3. Unified Modeling Language. The Object Management Group. http://www.omg.org. 1999

121. UMLX Development Tools GMT subproject http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/UMLX/index.html. 2006

122. Valderas P., Fons J., Pelechado V. Developing E-Commerce Application from Task-Based Descriptions. EC-Web 2005. Copenhagen. August 2005

123. Velocity. Jakarta Proyect. http://jakarta.apache.org/velocity/

124. VisualWADE Tool. Universidad de Alicante. http://www.visualwade.com

125. WebE. 2002. The WebEngineering.org community site. www.webengineering.org.

126. WebRatio. The CASE Tool for the web. Politecnico di Milano. http://www.webratio.com/sv1.do