

Desarrollo de Familias de Aplicaciones Web con Transformación de Modelos

por Hernán Casalánguida

Presentado ante la Facultad de Matemática, Astronomía y Física como parte de los requerimientos de la obtención del grado de Doctor en Ciencias de la Computación de la



UNIVERSIDAD NACIONAL DE CÓRDOBA

Abril, 2020

©FaMAF - UNC 2020

Juan Eduardo Durán



Esta obra está bajo una [Licencia Creative Commons Atribución – No Comercial – Sin Obra Derivada 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Abstract

The development using the Software Product Line (SPL) approach provides several benefits in the development of software systems, such as: reduction of development costs, enhancement of quality and reduction of time to market. For these reasons, its adoption has been spreading in different types of software systems; in particular, in recent years numerous proposals have emerged that use the SPL approach for developing web applications or important parts of them, as a user interface. The features models play a very an important role in SPL development, because they allow to specify the domain of an SPL (commons or variants requirements between products). Furthermore, due to its concise and easy to read notation, they are very useful for validating the requirements of a family of systems with the clients. Domain models, in addition to feature models, gives the opportunity to specify common or variants requirements for products, in notations more conducive to guide the development in a progressive and incremental way.

Model driven development (MDD) consists of creating abstract models of software systems that are systematically transformed to realize implementations or new models. The adoption of MDD speeds development through automations and that analysts formally specify their knowledge about the type of applications to be developed, the technology and its mapping, clearing details of other areas.

The evolution of web applications, in terms of technology and access devices, has caused many applications to be generated similar ones, that are developed individually. Moreover, some highly configurable large-scale desktop systems (CRMs, ERPs, among other types) have been migrating to the web. For these reasons, many web applications have been transformed into a family of applications that need to be modeled as such, in order to take advantage of the benefits that this type of development provides.

The primary objectives of this thesis are to solve the main problems found in the SPL development approaches of web applications: manual construction of features model (in addition to domain models); lack of automatism to generate domain model configurations; do not model or take into account variability in user interfaces and do not prescribe a method to model user interface. In addition, other objectives are to solve some important problems in the SPL area: automatic generation of features model from domain models, gaps in notations to model variability, MDD consideration for interactive configuration of domain models; and problems of the web applications area: abstract consideration of access to functionalities and content, consideration of the production process of user interfaces automatically adaptable to different access devices.

The main contribution of this work is an approach for SPL development of web applications taking into account the resolution of the problems found in SPL development approaches of web applications. Other contributions of this work are the resolution of the problems found in the area of SPL and in the area of web applications, through: notation extension for UML use case diagram with variability, definition of a UML activity diagram notation to capture variability, definition of automatic feature model generation process based on domain models, definition of an MDD process for interactive configuration of domain models, definition of a notation to model access to functionalities and content, definition of metamodel for abstract user interface that takes into account the latest technological advances and definition of production process of user interfaces automatically adaptable to different access devices.

The process defined here has been validated through a complete case study, corresponding to a highly configurable online library system (Koha library system style). For the definition of some notations or situations included in the work, other case studies were partially taken into account.

Resumen

El enfoque de desarrollo de Líneas de Productos de Software (SPL, por sus siglas en inglés) trae aparejado varios beneficios en el desarrollo de sistemas de software, tales como: reducción de costos de desarrollo, mejora en la calidad y reducción en tiempos de comercialización. Por estos motivos, su adopción se ha ido propagando en distintos tipos de sistemas de software; en particular, en los últimos años han surgido numerosas propuestas que utilizan el enfoque SPL para desarrollo de aplicaciones web o partes importantes de éstas, como la interfaz de usuario. Los modelos de features juegan un papel importante en SPL, debido a que permiten especificar el dominio de una SPL (requisitos comunes o variantes entre productos) y, por su notación concisa y de fácil lectura, son de gran utilidad para validación de requisitos de una SPL con el cliente. En adición a los modelos de features, los modelos de dominio permiten especificar requisitos comunes y variantes de productos, utilizando notaciones más propicias para guiar el desarrollo de una manera progresiva e incremental.

El Desarrollo Dirigido por Modelos (MDD, por sus siglas en inglés) consiste en la creación de modelos abstractos de sistemas de software que son sistemáticamente transformados para concretar implementaciones o nuevos modelos. La adopción de MDD permite agilizar el desarrollo a través de automatizaciones y que los analistas de software especifiquen formalmente su conocimiento sobre el tipo de aplicaciones a desarrollar, la tecnología y su mapeo; despejándose de detalles de otras áreas.

La evolución de las aplicaciones web, en cuanto a tecnología y dispositivos de acceso, hizo que se generen muchas aplicaciones similares que son desarrolladas individualmente. Además, algunos sistemas de escritorio de gran escala altamente configurables (por ejemplo sistemas CRM, sistemas ERP) han ido migrando a la web. Por estas razones, muchas aplicaciones web se han transformado en SPL y necesitan ser modeladas como tales, para así aprovechar los beneficios que este tipo de desarrollo provee.

Los objetivos primarios de este trabajo son atacar los principales problemas hallados en los enfoques de desarrollo SPL de aplicaciones web: construcción manual de modelo de features (en adición a modelos de dominio); no provisión de automatismos para generar configuraciones de modelos de dominio; no modelado ni consideración de variabilidad en interfaces de usuario y no prescripción de método para modelar interfaz de usuario. Además se desean tener en cuenta problemas generales del área de SPL: generación automática de modelo de features a partir de modelos de dominio, carencias en notaciones para modelar variabilidades, consideración de MDD para configuración interactiva de modelos de dominio; y aspectos propios del área de aplicaciones web: consideración abstracta de acceso a funcionalidades y contenido, y consideración de proceso de producción de interfaces de usuario adaptables automáticamente a diferentes dispositivos de acceso.

Como principal contribución de este trabajo se presenta un enfoque para desarrollo SPL de aplicaciones web teniendo en cuenta la resolución de los problemas hallados en los enfoques de desarrollo SPL de aplicaciones web. Otras contribuciones de la tesis son la resolución de los problemas hallados en el área de SPL y en el área de aplicaciones web, a través de: extensión de notación para diagrama de casos de uso de UML con variabilidades, definición de una notación de diagramas de actividad de UML para capturar variabilidades, definición de proceso de generación automática de modelo de features en base a modelos de dominios, definición de un proceso MDD para configuración interactiva de modelos de dominio, definición de una notación para modelar acceso a funcionalidades y contenido, definición de metamodelo para interfaz de usuario abstracta que tiene en cuenta los últimos adelantos tecnológicos y definición de proceso de producción de interfaces de usuario adaptables automáticamente a diferentes dispositivos de acceso.

El proceso definido aquí ha sido validado a través de un caso de estudio integral, correspondiente a un sistema de biblioteca online altamente configurable (del estilo del sistema de bibliotecas Koha). Para la definición de algunas notaciones o situaciones incluidas en el trabajo se tuvieron en cuenta otros casos de estudio de manera parcial.

Índice general

1. Introducción	1
1.1. Conceptos básicos	1
1.1.1. Líneas de productos de software	1
1.1.2. Aplicaciones Web	2
1.1.3. Lenguaje Unificado de Modelado	2
1.1.4. Desarrollo Dirigido por Modelos	2
1.2. Motivación	3
1.2.1. ¿Cuáles son los beneficios del desarrollo SPL?	3
1.2.2. ¿Por qué modelar Aplicaciones Web y Sistemas de Software como SPL?	3
1.2.3. ¿Por qué adoptar MDD?	3
1.3. Estado del arte, SPL en el dominio de aplicaciones web	4
1.3.1. Trabajos relacionados	4
1.3.2. Carencias halladas en la bibliografía	4
1.4. Metas y objetivo	5
1.5. Enfoque presentado	6
1.6. Visión general	8
I Notaciones de modelado utilizadas	9
2. Notaciones para modelado de aplicaciones	10
2.1. Requisitos	10
2.1.1. Taxonomía para describir acciones de diagramas de actividad de Aplicaciones Web	10
2.1.2. Framework NFR(Non-Functionals Requirements)	12
2.2. Diseño	14
2.2.1. Notación para acceso a funcionalidades y contenido de interfaz de usuario	14
2.2.2. Notación para diseño abstracto de interfaz de usuario	21

2.2.3. Taxonomía para describir actividades autónomas del sistema en diagramas de actividad	30
3. Notaciones para modelado de dominio	34
3.1. Diagramas de Casos de Uso con variabilidades	34
3.1.1. Background	34
3.1.2. Trabajo relacionado	35
3.1.3. Notación propuesta para Diagramas de Casos de Uso con variabilidades	36
3.1.4. Evaluación	39
3.2. Diagramas de Actividad con variabilidades	40
3.2.1. Background	40
3.2.2. Trabajo relacionado	43
3.2.3. Notación propuesta para Diagramas de Actividad con variabilidades	44
3.2.4. Evaluación	57
3.3. Framework NFR con variabilidades	58
3.3.1. Background	58
3.3.2. Trabajo relacionado	59
3.3.3. Notación propuesta para Framework NFR con variabilidades	59
3.3.4. Configuración del SIG de dominio	62
3.3.5. Procedimiento de configuración de SIG de dominio	62
3.3.6. Evaluación	64
3.4. Modelos de features	64
3.4.1. Background	64
3.4.2. Notación propuesta para modelo de features	66
3.5. Modelos de interfaz de usuario abstracta con variabilidades	69
3.5.1. Background	69
3.5.2. Trabajo relacionado	69
3.5.3. Una propuesta de notación para interfaz de usuario abstracta con variabilidades	70
3.5.4. Algunas situaciones genéricas de variabilidad en interfaz de usuario identificadas	70

II Desarrollo de modelos de dominio de familias de aplicaciones web y generación de modelo de features **75**

4. Proceso de desarrollo de modelos de dominio y generación de modelo de features	76
4.1. Background	76

4.2. Trabajo relacionado	78
4.3. Descripción del proceso	80
4.3.1. Desarrollo de modelos de dominio para familias de aplicaciones	80
4.3.2. Generación de modelo de features a partir de modelos de dominio	81
4.3.3. Actualización de modelos de dominio	82
4.4. Desarrollos Open Source que pueden ser considerados como familias de aplicaciones	84
5. Desarrollo de modelos de requisitos de dominio	85
5.1. Desarrollo de diagramas de casos de uso de dominio	85
5.1.1. Guías para construir diagramas de casos de uso de dominio	85
5.1.2. Aplicación al caso de estudio de Sistema de Administración de Biblioteca Online	86
5.2. Desarrollo de diagramas de actividad de dominio	102
5.2.1. Guías para construir diagramas de actividad de dominio	102
5.2.2. Aplicación al caso de estudio de Sistema de Administración de Biblioteca Online	103
6. Generación automática de modelo de features a partir de modelos de requisitos de dominio	121
6.1. Mapeando diagramas de casos de uso de dominio a modelo de features inicial	121
6.1.1. Background	121
6.1.2. Trabajo relacionado	122
6.1.3. Reglas de transformación	122
6.2. Mapeando diagramas de actividad de dominio a modelo de features de requisitos	124
6.2.1. Background	124
6.2.2. Trabajo relacionado	125
6.2.3. Reglas de transformación	125
6.3. Evaluación de reglas de transformación	130
7. Desarrollo de modelos de dominio de diseño y generación automática de features a partir de ellos	132
7.1. Desarrollo de modelos de interfaz de usuario abstracta de dominio	132
7.1.1. Guías para desarrollo de modelos de interfaz de usuario de dominio	132
7.1.2. Aplicación al caso de estudio de Sistema de Administración de Bibliotecas Online	134
7.2. Mapeando modelos de interfaz de usuario abstracta de dominio a features	138
7.2.1. Background	138
7.2.2. Trabajo relacionado	139

7.2.3. Reglas de transformación	139
III Configuración de modelos de dominio de aplicaciones web	141
8. Proceso de configuración de modelos de dominio	142
8.1. Background	142
8.2. Trabajo relacionado	143
8.3. Descripción del proceso	144
9. Configuración de diagramas de features	146
9.1. Definición de configuración	146
9.2. Guías para configurar diagramas de features	148
10. Configuración de modelos de dominio	151
10.1. Configuración de diagramas de Casos de Uso de dominio	151
10.2. Configuración de diagramas de actividad de dominio	153
10.3. Configuración de modelos de interfaz de usuario abstracta de dominio	157
IV Desarrollo de modelos de aplicación e implementación de interfaz de usuario responsiva	161
11. Actualización de modelos configurados en base a requisitos de la aplicación	162
11.1. Actualización de diagramas de casos de uso	163
11.2. Actualización de diagramas de actividades	163
11.3. Actualización de interfaz de usuario abstracta	163
12. Desarrollo de modelo de acceso a funcionalidades y contenido	165
12.1. Proceso de desarrollo de modelo de acceso a funcionalidades y contenido de una aplicación	165
12.1.1. Aplicación al caso de estudio	166
12.2. Definición de relaciones de traza	173
13. Implementación de interfaz de usuario de aplicaciones web responsivas	180
13.1. Background	180
13.2. Trabajo relacionado	181
13.3. Frameworks responsivos	182
13.3.1. Conceptos básicos	182

13.3.2. Selección de frameworks responsivos	183
13.4. Mapeo de elementos de interfaz de usuario abstracta a elementos de interfaz de usuario de frameworks responsivos	184
13.5. Proceso de desarrollo de interfaz de usuario final usando un framework responsivo	185
13.5.1. Implementando interfaz de usuario final utilizando una herramienta para frameworks responsivos	186
V Conclusión	192
14. Conclusión	193
14.1. Resumen de contribuciones	193
14.2. Discusión	195
14.3. Perspectivas y trabajo futuro	196
14.4. Tendencias y avances en el período 2017–2020 en el ámbito de aplicabilidad del trabajo .	197
Appendices	199
A. Código fuente transformaciones en ATL	200
A.1. Transformación de diagramas de casos de uso de dominio a modelos de features	200
A.2. Trasnformación de diagramas de actividad de dominio a modelos de features	207
A.3. Transformación de restricciones de dependencia en modelos de dominio a restricciones de dependencia en modelos de features	211
A.4. Reglas para mapeo de partes de interfaz de usuario con variabilidades a modelo de features	211
B. Implementación de configuraciones	213
B.1. Implementando la producción de una configuración	213
B.2. Configurando modelos de features	218
B.3. Configurando diagramas de de casos de uso de dominio	220
Bibliografía	230

Índice de figuras

1.1. Resumen de actividades para el enfoque de desarrollo de familia de aplicaciones web.. . . .	7
2.1. Ejemplo de uso del perfil para describir acciones en actividad Editar imagen con auto guardado de aplicación web.	11
2.2. Resumen de la notación utilizada para los SIGs	13
2.3. Ejemplo de SIG para el requisito no funcional Usabilidad para las operaciones de Préstar de libro y Buscar ítems de una aplicación de Biblioteca Online.	14
2.4. Meta-modelo para acceso a funcionalidades y contenido. Parte para expresar vista estática.	17
2.5. Sintaxis concreta para elementos de vista estática de modelo de acceso a funcionalidades y contenido.	17
2.6. Ejemplo de modelo de vista estática de acceso a funcionalidades y contenido para grouping <i>User Agent</i> de aplicación de correo electrónico online.	18
2.7. Meta-modelo para acceso a funcionalidades y contenido. Parte para expresar vista dinámica.	19
2.8. Sintaxis concreta para elementos de vista dinámica de modelo de acceso a funcionalidades y contenido.	19
2.9. Ejemplo de modelo de vista dinámica de acceso a funcionalidades y contenido para algunos elementos del grouping <i>User Agent</i> de una aplicación para correo electrónico online.	20
2.10. Meta-modelo para diseño abstracto de interfaces de usuario. Parte 1.	22
2.11. Meta-modelo para diseño abstracto de interfaces de usuario. Parte 2.	24
2.12. Meta-modelo para diseño abstracto de interfaces de usuario. Parte 3.	25
2.13. Meta-modelo para diseño abstracto de interfaces de usuario. Parte 4.	26
2.14. Sintaxis concreta para elementos del meta-modelo RIAAD.	28
2.15. Ejemplo de modelo de interfaz de usuario abstracta para grouping <i>User Agent</i> de aplicación de correo electrónico online.	29
2.16. Clasificación de eventos atómicos.	30
2.17. Ejemplo para refinamiento de «job» genérico <i>validate data and add item job</i>	32

2.18. Ejemplo para refinamiento de «job» genérico <i>Resize image</i>	32
2.19. Ejemplo para refinamiento de «job» genérico <i>Process payment</i>	33
3.1. Comparación de notaciones de diagramas de casos de uso con variabilidades.	35
3.2. Meta-modelo para variabilidades en Diagramas de Casos de Uso.	36
3.3. Meta-modelo para variabilidades en Modelos de Casos de Uso.	37
3.4. Parte del paquete de casos de uso OPAC del caso de estudio de Administración de Bibliotecas Online.	38
3.5. Parte del paquete de casos de uso My Account, incluido dentro del paquete de casos de uso OPAC.	39
3.6. <i>CreateIndexAccess ExtendVariability</i> . Parte de paquetes de casos de uso Orders y Baskets.	39
3.7. Comportamiento en Look for reservation.	41
3.8. Comportamiento Obtain Due Date que es usado por el caso de uso renewing.	42
3.9. Comparación de notaciones de diagramas de actividad con variabilidades.	43
3.10. Meta-modelo para anotaciones de variabilidad en diagramas de actividad.	44
3.11. Diagrama de actividad para el elemento «execute behavior» <i>Obtain due date</i>	46
3.12. Diagrama de actividad para el elemento «execute behavior» <i>suggest actions for records</i>	47
3.13. Diagrama de actividad para el elemento «execute behavior» <i>add item types, authorized value and 856u</i>	48
3.14. Diagrama de actividad para el caso de uso <i>renewing</i>	49
3.15. Parte de la descripción del caso de uso <i>check out</i>	51
3.16. Definición de conjuntos parámetro.	52
3.17. Ejemplo de variabilidad de conjunto parámetro para tarea de agregar un registro de nuevo usuario a una aplicación.	52
3.18. Parte de la descripción del caso de uso <i>search catalog in the OPAC</i>	53
3.19. Elemento de variabilidad de condición <i>independent branches</i>	54
3.20. Regla de negocio para agregar un miembro a la biblioteca.	55
3.21. Restricciones de dependencia para diagramas de actividad y ejecuciones de comportamiento.	56
3.22. Restricción de dependencia entre una arista de actividad variante y un variante asociación de diagrama de casos de uso.	57
3.23. Restricción de dependencia entre una arista de actividad variante y un variante de condición.	57
3.24. Extensión al meta-modelo del framework NFR para modelar variabilidades.	61
3.25. Parte de SIG con variabilidades para el requisito no funcional de Accesibilidad.	62

3.26. SIG con variabilidades para Usabilidad en subsistema OPAC del Sistema de Administración de Bibliotecas Online.	63
3.27. SIG especializado para Usabilidad en subsistema OPAC del Sistema de Administración de Bibliotecas Online.	64
3.28. Ejemplo de diagrama de features para una familia de automóviles utilizando relaciones tradicionales.	65
3.29. Meta-modelo de features.	67
3.30. Diagrama de features para para el caso de estudio Sistema de Administración de Bibliotecas Online, incluyendo las features de más alto nivel del subsistema OPAC.	68
3.31. Diagrama de features para los niveles más altos del paquete My account perteneciente al subsistema OPAC del caso de estudio Sistema de Administración de Bibliotecas Online.	68
3.32. Meta-modelo para anotaciones de variabilidad en modelado abstracto de interfaz de usuario abstracta.	70
3.33. Variabilidad opcional en modelado abstracto de interfaz de usuario abstracta: botón para limpiar pantalla.	71
3.34. Variabilidad opcional en modelado abstracto de interfaz de usuario abstracta: foto de perfil opcional.	71
3.35. Variabilidad en patrones de interfaz de usuario: tablas con distintos patrones.	72
3.36. Ejemplo de variabilidad de alternativas de tipos de records en ítems.	72
3.37. Ejemplo de variabilidad de selección de distintas composiciones de interfaz de usuario abstracta.	73
3.38. Ejemplo de variabilidad en visualización de una información de medida en distintos formatos: formatos de temperatura.	73
4.1. Comparación entre enfoques seleccionados para procesos de generación de modelos de features a partir de modelos de dominio.. . . .	79
4.2. Proceso de desarrollo de modelos de dominio y feaures de familias de aplicaciones web.	80
4.3. Proceso de producción de modelos de dominio de familia de aplicaciones web desde cero.	81
4.4. Actividad para generar modelo de features a partir de modelos de dominio de requisitos.	81
4.5. Actividad para generar modelo de features a partir de modelos de interfaz de usuario abstracta con variabilidades.	82
5.1. Actores identificados para el caso de estudio de familia de sistemas de administración de bibliotecas online.	86

5.2. Subsistemas identificados para el caso de estudio de familia de sistemas de administración de bibliotecas online.	86
5.3. Paquetes identificados para el subsistema Administration del caso de estudio de familia de sistemas de administración de bibliotecas online.	87
5.4. Paquetes identificados para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.	89
5.5. Paquetes identificados dentro del paquete de casos de uso Acquisitions para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.	89
5.6. Paquetes identificados dentro del paquete de casos de uso Cataloguing para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.	90
5.7. Paquete identificado dentro del paquete de casos de uso Serials para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.	90
5.8. Paquetes identificados para el subsistema OPAC del caso de estudio de familia de sistemas de administración de bibliotecas online.	91
5.9. Paquetes identificados dentro del paquete de casos de uso My account del subsistema OPAC del caso de estudio de familia de sistemas de administración de bibliotecas online.	92
5.10. Variabilidades identificadas en paquetes del subsistema Administration del caso de estudio de familia de sistemas de administración de bibliotecas online.	92
5.11. Variabilidades identificadas en paquetes del subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.	93
5.12. Variabilidades identificadas en paquetes del subsistema OPAC del caso de estudio de familia de sistemas de administración de bibliotecas online.	94
5.13. Paquete de casos de uso OPAC.	95
5.14. Paquete de casos de uso My Account.	96
5.15. Casos de uso, variabilidades y relaciones entre casos de uso identificados en los paquetes, del subsistema OPAC, Public Search y Private Search.	98
5.16. Casos de uso, variabilidades y relaciones entre casos de uso identificados en los paquetes (que están dentro del subsistema OPAC y del paquete My account) Tags, Search history, Reading history y Purchase suggestions.	99
5.17. Casos de uso y relaciones entre casos de uso identificados en el paquete (que está dentro del subsistema OPAC y del paquete My account) Lists.	100
5.18. Casos de uso, variabilidades y relaciones entre casos de uso identificados en los paquetes (que están dentro del subsistema OPAC y del paquete My account) Holds in OPAC, Comments in OPAC, Cart in OPAC y OPAC Renewal.	101

5.19. Casos de uso identificados en el paquete (que está dentro del subsistema OPAC y del paquete My account) Browse Subject Authorities.	102
5.20. Primera parte de diagrama de actividad para el caso de uso Search catalog from the OPAC, incluido en los paquetes Public Search y Private Search del subsistema OPAC.	105
5.21. Segunda parte de diagrama de actividad para el caso de uso Search catalog from the OPAC, incluido en los paquetes Public Search y Private Search del subsistema OPAC.	106
5.22. Primera parte de diagrama de actividad para el caso de uso Show record in the the OPAC del subsistema OPAC.	107
5.23. Segunda parte de diagrama de actividad para el caso de uso Show record in the the OPAC del subsistema OPAC.	108
5.24. Comportamientos reutilizables generalizados, utilizados en el diagrama de actividad del caso de uso Search catalog from the OPAC y en otros diagramas de actividad (Search catalog from the Staff Client y Show record in the the OPAC).	109
5.25. Comportamientos reutilizables generalizados, utilizados en el diagrama de actividad del caso de uso Show record in the the OPAC y en otros diagramas de actividad del caso de estudio.	110
5.26. Diagrama de actividad final para el caso de uso Search catalog from the OPAC.	112
5.27. Diagrama de actividad final para el caso de uso Show record in the the OPAC.	113
5.28. Diagrama de actividad final para el caso de uso Renewing item (usado en los subsistemas Staff client y OPAC).	114
5.29. Diagrama de actividad final para el caso de uso Check Out (usado en el subsistema Staff client).	115
5.30. Diagrama de actividad final para el caso de uso Check In (usado en el subsistema Staff client).	117
5.31. Diagrama de actividad final para el caso de uso Place Hold in staff client.	118
5.32. Diagrama de actividad final para el caso de uso Place Hold from the OPAC.	119
5.33. Diagrama de actividad final para el caso de uso Self-Checkout, del subsistema Staff client.	120
6.1. Reglas de transformación de modelo de casos de uso a modelo de features.	123
6.2. Reglas de transformación de diagramas de actividad con variabilidades a modelo de features.	126
6.3. Modelo de features para descripción del caso de uso <i>Renewing</i>	127
6.4. Primera parte del modelo de features para descripción del caso de uso <i>Search catalog in the OPAC</i>	128

6.5.	Segunda parte del modelo de features para descripción del caso de uso <i>Search catalog in the OPAC</i>	129
6.6.	Primera parte del modelo de features para descripción del caso de uso <i>Show record in the OPAC</i>	129
6.7.	Segunda parte del modelo de features para descripción del caso de uso <i>Show record in the OPAC</i>	130
7.1.	Interfaz de usuario abstracta con variabilidades del caso de uso <i>Search catalog in the OPAC</i> .	135
7.2.	Interfaz de usuario abstracta con variabilidades para la funcionalidad <i>Advanced search</i> . . .	136
7.3.	Variabilidades identificadas en la interfaz de usuario abstracta del caso de uso <i>Show record in the OPAC</i>	137
7.4.	Transformación <i>UI2FM</i> para adjuntar features y relaciones provenientes de interfaz de usuario abstracta con variabilidades a modelo de features de la familia de aplicaciones. . .	139
7.5.	Modelo de features generado utilizando la transformación <i>emphUI2FM</i> para la interfaz de usuario abstracta del caso de uso <i>Search catalog in the OPAC</i>	140
7.6.	Modelo de features generado utilizando la transformación <i>emphUI2FM</i> para la interfaz de usuario abstracta del caso de uso <i>Show record in the OPAC</i>	140
8.1.	Comparación entre enfoques seleccionados para configuración automatizada basada en modelos de features.	144
8.2.	Proceso de desarrollo de modelos de una aplicación a partir de modelos de dominio de aplicaciones web.	145
9.1.	Diagrama de features configurado (aplicando la configuración <i>SIMBOConf</i>) para el caso de estudio: Sistema de Administración de Bibliotecas Online, incluyendo las features de más alto nivel y desarrollando parte del subsistema OPAC.	149
9.2.	Diagrama de features configurado (aplicando la configuración <i>SIMBOConf</i>) para los niveles más altos del paquete My account perteneciente al subsistema OPAC del caso de estudio Sistema de Administración de Bibliotecas Online.	149
9.3.	Diagrama de features configurado (aplicando la configuración <i>SearchCatalogConf</i>) para el diagrama de features generado a partir de la descripción e interfaz abstracta del caso de uso <i>Search catalog in the OPAC</i> del caso de estudio Sistema de Administración de Bibliotecas Online.	150

9.4. Diagrama de features configurado (aplicando la configuración <i>ShowRecordOpacConf</i>) para el diagrama de features generado a partir de la descripción e interfaz abstracta del caso de uso <i>Show Record in OPAC</i> del caso de estudio Sistema de Administración de Bibliotecas Online.	150
10.1. Diagrama de casos de uso configurado (aplicando la configuración <i>SIMBOConf</i>) para el paquete de casos de uso más alto nivel del subsistema OPAC del caso de estudio Sistema de Administración de Bibliotecas Online.	153
10.2. Diagrama de actividad configurado (aplicando la configuración <i>SearchCatalogConf</i>) para el caso de uso Search catalog in the OPAC del subsistema OPAC.	155
10.3. Comportamientos reutilizables configurados (aplicando la configuración <i>SearchCatalogConf</i>) que aparecen en el caso de uso Search catalog in the OPAC del subsistema OPAC.	156
10.4. Diagrama de actividad configurado (aplicando la configuración <i>ShowRecordOpacConf</i>) para el caso de uso Show record in the OPAC del subsistema OPAC.	157
10.5. Interfaz de usuario abstracta configurada (aplicando la configuración <i>SearchCatalogConf</i>) para el caso de uso Search catalog in the OPAC del subsistema OPAC.	159
10.6. Interfaz de usuario abstracta configurada (aplicando la configuración <i>SearchCatalogConf</i>) para la funcionalidad de Advanced Search que aparece en el caso de uso Search catalog in the OPAC del subsistema OPAC.	159
10.7. Interfaz de usuario abstracta configurada (aplicando la configuración <i>SearchCatalogConf</i>) para el caso de uso Show record in the OPAC del subsistema OPAC.	160
11.1. Proceso de actualización de modelos de aplicación de UML configurados.	162
12.1. Agrupamiento inicial visualizado por un usuario de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	167
12.2. Agrupamiento para acceso a listas de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	167
12.3. Agrupamiento para ingreso al sistema de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	167
12.4. Agrupamiento para barra de usuario de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	167
12.5. Agrupamiento principal para usuario autenticado de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	168

12.6. Agrupamiento para el contenido y funcionalidades de un item de biblioteca de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	168
12.7. Agrupamiento para el contenido y funcionalidades del resultado de búsquedas de items de biblioteca de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	168
12.8. Agrupamiento para la funcionalidad de búsqueda avanzada de items de biblioteca de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	169
12.9. Agrupamiento para el elemento de contenido Cart que representa el carrito de préstamos de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	169
12.10. Agrupamiento para el elemento de contenido Lists que representa a listas de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	169
12.11. Agrupamiento para el elemento de contenido List que representa una lista de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	169
12.12. Agrupamiento para el elemento de contenido Purchase suggestions que representa sugerencias de compra de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	170
12.13. Agrupamiento para el elemento de contenido Tags que representa etiquetas de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	170
12.14. Agrupamiento para el elemento de contenido Reading History que representa historial de búsquedas en el catálogo de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	170
12.15. Agrupamiento para el elemento de contenido Search History que representa historial de búsquedas en el catálogo de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	170
12.16. Agrupamiento para el elemento de contenido Overdue Items que representa items con retrasos de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	170
12.17. Agrupamiento para el elemento de contenido Hold items que representa retenciones de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.	171
12.18. Vista dinámica de los elementos Login Group, Account Bar, Public Main, User Main (relacionados a funcionalidades de ingreso y salida del sistema) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio caso de estudio.	171

12.19	Vista dinámica de los elementos Search, Advanced Search (relacionados a funcionalidades de búsquedas de ítems) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio.	172
12.20	Vista dinámica del elemento view all public lists (relacionados a funcionalidades del agrupamiento access to lists) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio. Lo mismo se replica (solo cambia el nombre de la funcionalidad inicial) para show all my lists.	172
12.21	Vista dinámica del elemento view public list (relacionados a funcionalidades del agrupamiento access to lists) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio. Lo mismo se replica (solo cambia el nombre de la funcionalidad inicial) para show my list.	172
12.22	Vista dinámica del elemento show purchase suggestions (relacionados a funcionalidades del agrupamiento utilities bar) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio. Lo mismo se replica (solo cambia el nombre de la funcionalidad inicial) para my purchase suggestions.	173
12.23	Vista dinámica del elemento view cart (relacionados a funcionalidades del agrupamiento access to cart) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio.	173
12.24	Refinamiento de elementos de tipo Input de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	174
12.25	Refinamiento de elementos de tipo Read only Content de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	175
12.26	Refinamiento del elemento de tipo Content with interaction Cart de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	175
12.27	Refinamiento del elemento de tipo Content with interaction Lists de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	176
12.28	Refinamiento del elemento de tipo Content with interaction List de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	176
12.29	Refinamiento del elemento de tipo Content with interaction Purchase Suggestions de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	176

12.30Refinamiento de los elementos de tipo Content with interaction Tags, Reading history y Search History de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	177
12.31Refinamiento de los elementos de tipo Content with interaction checkout items, overdue items y holds items de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.	178
13.1. Tabla de mapeos de elementos de <i>WAAUID</i> a elementos de Frameworks responsivos. Parte 1.	184
13.2. Tabla de mapeos de elementos de <i>WAAUID</i> a elementos de Frameworks responsivos. Parte 2.	185
13.3. Creando una página en blanco en <i>Pingendo</i>	187
13.4. Estructuras de layout predefinidas en <i>Pingendo</i>	188
13.5. Incluyendo una barra de navegación en <i>Pingendo</i>	189
13.6. Edición de código <i>HTML</i> en <i>Pingendo</i>	190
B.1. Detalle de actividades para el proceso de producción de una configuración válida utilizando la herramienta realizada.	213
B.2. Ejemplo simple de modelo de features para algunas características de un automóvil.	214
B.3. Pantalla generada para selección de variantes en ejemplo simple de modelo de features de un automóvil.	217
B.4. Detalle de actividades para el proceso de configurar un modelo de features.	218
B.5. Pantalla que se visualiza en caso de no cumplirse con la restricción de selección de variantes.219	
B.6. Resultado de la configuración utilizando la herramienta realizada en ejemplo simple de modelo de features de un automóvil.	220
B.7. Detalle de actividades para el proceso de configurar un diagrama de casos de uso con variabilidades.	220
B.8. Resultado de la configuración aplicada a diagrama de casos de uso con variabilidades en ejemplo simple de modelo de features de un automóvil.	222

Capítulo 1

Introducción

1.1. Conceptos básicos

1.1.1. Líneas de productos de software

Una Línea de Productos de Software (SPL, por sus siglas en inglés, Software Product Lines) «*es un conjunto de sistemas intensivos en software que comparten un conjunto de características común y administrado, las cuales satisfacen necesidades específicas de un segmento o nicho de mercado en particular y se desarrollan a partir de un conjunto común de elementos núcleo de una manera prescrita*» (ver [Clements, 2002]). También se utilizan los términos Familia de Productos de Software o Familia de Sistemas de Software para referirse a una SPL (ver [Linden, 2002]). En este trabajo, al tratar con aplicaciones web, utilizaremos en general los términos “Familia de Aplicaciones Web” (FAW) ó “Familia de Aplicaciones” (FA).

La *Ingeniería de SPL* es un paradigma para desarrollar aplicaciones de software (sistemas intensivos de software y productos de software) usando plataformas y personalización masiva. La Ingeniería de SPL se divide en dos procesos: Ingeniería de Dominio e Ingeniería de Aplicación.

Ingeniería de Dominio

Este proceso es responsable de establecer un núcleo de elementos reutilizables y de definir las características comunes y variables (variabilidades) de una FA. Los elementos reutilizables de una familia de aplicaciones son artefactos de software de cualquier etapa del ciclo de desarrollo de software (requisitos, diseño, implementación, pruebas) que son utilizados para la construcción de distintas aplicaciones de una FA.

La ingeniería de dominio representa una actividad iterativa, ya que se realimenta de nuevos requisitos de dominio provenientes de nuevos conocimientos de dominio o de nuevas aplicaciones incorporadas a la familia. Dos tipos de modelos juegan un rol importante durante esta etapa, y en general en el campo de la ingeniería SPL, estos son:

- **Modelo de features.** Una feature se define como «*una característica de una familia de aplicaciones, que un sistema puede poseer o no*». Los modelos de features son un conjunto de features estructuradas, en general, en forma de árbol. Surgieron en el año 1990 con FODA (Feature-Oriented Domain Analysis, ver [Kang, 1990]) y su propósito principal es representar en un diagrama de features un análisis de variabilidades y características en común de la familia de aplicaciones. Este modelo es muy importante en SPL, porque además de permitir en un solo modelo representar las partes variables y comunes de una familia de aplicaciones, facilita el proceso de creación de aplicaciones (a través de selección de features variables a incluir o no en una familia dada), es una notación concisa y de fácil lectura y es de gran utilidad para validación de requisitos SPL con clientes.
- **Modelos de dominio.** Parte de los elementos del núcleo reutilizable que se establece en SPL son los modelos de dominio, los cuales son modelos en alguna notación de modelado que incluyen variabilidades y características en común para alguna etapa del ciclo de vida de modelado de FA. En adición a los modelos de features, los modelos de dominio permiten guiar el desarrollo de

una aplicación a través de enlaces trazabilidad, los cuales facilitan la reutilización sistemática y consistente, y permiten guiar el desarrollo de una SPL de una manera iterativa e incremental.

Ingeniería de aplicación

Este proceso es responsable de construir aplicaciones (o partes de ellas) en base a modelos de aplicación (modelos de dominio configurados de acuerdo a la resolución de todas las variabilidades incluidas en ellos) y a requisitos concretos de una aplicación. Durante esta etapa, puede ser necesario actualizar los modelos de aplicación por medio de configuración (proceso de resolución de variabilidades) en base a los requisitos de la aplicación, para luego en base a enlaces de trazabilidad entre modelos realizar la construcción de una aplicación completa o de alguna parte en particular (en este trabajo nos hemos concentrado en interfaces de usuario). En general, en esta etapa se explota la variabilidad de la FA y se garantiza la unión correcta de dicha variabilidad según las necesidades específicas de una aplicación.

1.1.2. Aplicaciones Web

Una Aplicación Web es «*un sistema de aplicación que es invocado a través de un navegador web sobre Internet*». En los orígenes de Internet, la Web solo consistía de sitios web con la finalidad de compartir información. Los sitios web consistían de un conjunto de páginas web estáticas accedidas de forma síncrona entre un cliente y un servidor web. Con el correr de los años, los sitios web se fueron sofisticando, incorporando nuevas características que los fueron convirtiendo en aplicaciones con las que un usuario interactúa dinámicamente. En los últimos años, el notable avance en cuanto a dispositivos de acceso a Internet (tabletas, televisores, teléfonos móviles) y a tecnología de desarrollo web (Web 2.0, frameworks de desarrollo web, HTML5, CSS3, ECMAScript 2019) han tornado a las aplicaciones web cada vez más útiles y necesarias, a la vez que ha ido creciendo su adopción para tipos de sistemas más complejos, antes solo accesibles de manera local.

1.1.3. Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado, conocido en inglés como Unified Modeling Language (UML, ver [OMG, 2009]) es un lenguaje de modelado de sistemas de software creado por el Object Management Group (OMG). Es uno de los lenguajes de modelado más conocidos y utilizados en la actualidad, al punto de haberse convertido en un estándar. Permite a los desarrolladores especificar, visualizar, construir y documentar artefactos de un sistema de software. UML es un lenguaje de propósito general pero posee un mecanismo de extensión en base al uso de perfiles que ha permitido que sea utilizado para modelar diversos dominios de sistemas de software, en particular aplicaciones web. Si bien UML no provee ninguna notación para modelos de dominio, algunos autores han extendido distintos modelos de UML para contemplar definición de modelos de dominio (por ejemplo [Riebisch, 2000], [Robak, 2002], [Bragança, 2007]). Esto se debe a que es esperable que sea más fácil producir modelos de dominio utilizando un lenguaje de modelo ampliamente difundido como UML (con extensiones de ser necesario) que utilizando un lenguaje definido desde cero, el cual necesita ser aprendido y estudiado primero; la situación es peor aún si los encargados del desarrollo de la FA no cuentan con especialistas en SPL.

1.1.4. Desarrollo Dirigido por Modelos

El Desarrollo de Software Dirigido por Modelos, conocido en inglés como Model Driven Development (MDD) «*describe enfoques de desarrollo en los cuales modelos abstractos de sistemas de software son creados y transformados sistemáticamente para concretar implementaciones*» (según [France, 2007]). Otra definición, que en [Nare, 2008] denominan *definición de Forrester*, dice que «*el desarrollo dirigido por modelos es un enfoque iterativo de desarrollo de software donde los modelos son el origen de ejecución de programas con o sin generación de código*». En este trabajo optamos por esta última definición, ya que pensamos que se puede realizar un desarrollo dirigido por modelos sin necesidad de llegar hasta la generación de código fuente. Además, como se menciona en [Texier, 2012]; el objetivo de MDD es «*separar el diseño del sistema de la arquitectura de las tecnologías, para que puedan ser modificados independientemente. Para lograr esto, se asigna a los modelos un rol central y activo bajo el cual se derivan modelos que van desde los más abstractos a los concretos, este proceso se realiza a través de transformaciones*

sucesivas e iteraciones».

1.2. Motivación

1.2.1. ¿Cuáles son los beneficios del desarrollo SPL?

El desarrollo de sistemas de software utilizando SPL trae aparejado como principales beneficios (ver [Pohl, 2005]): *reducción de costos de desarrollo* (cuando se reutilizan artefactos en diferentes tipos de sistemas, esto implica una reducción de costo para cada sistema), *mejora en la calidad* (los artefactos a reutilizar son revisados y testeados en muchos productos) y *reducción en tiempo de comercialización* (el tiempo de comercialización es inicialmente más alto, ya que los artefactos comunes tienen que ser construidos primero. Sin embargo, después de haber superado este obstáculo, el tiempo de comercialización se acorta considerablemente ya que se pueden reutilizar muchos artefactos para cada nuevo producto). Además, en [Pohl, 2005] se mencionan como beneficios secundarios: *reducción de esfuerzo de mantenimiento* (por ejemplo para corrección de errores, los cambios pueden propagarse a todos los productos en los que un artefacto es usado), *evolución organizada* (la introducción de un nuevo artefacto para reutilizar —o el cambio de uno existente— da la oportunidad de evolucionar todo tipo de productos derivados), *lidar con complejidad* (el hecho de que las partes comunes se reutilicen en toda la línea de productos reduce la complejidad significativamente), *mejoras en estimación de costos* (los productos que necesitan extensiones pueden venderse a precios más altos que los productos creados mediante la reutilización de artefactos reutilizables solamente) y *beneficios para los clientes* (mejor calidad a precios más bajos).

1.2.2. ¿Por qué modelar Aplicaciones Web y Sistemas de Software como SPL?

En general en la Web existen numerosas aplicaciones web que son muy similares entre sí en términos de funcionalidad y propósitos, que sin embargo son desarrolladas por separado (por ejemplo, aplicaciones de subasta, de avisos, de comercio electrónico, de bibliotecas online, de agencias de ciencia y técnica, etcétera). Si para estas aplicaciones web similares (es decir que pertenecen a una misma familia) hubiera disponible una infraestructura reutilizable de elementos en común para su desarrollo, entonces a partir de dicha infraestructura se podría producir una aplicación web particular; ésto ahorraría numerosos costos en el desarrollo, reduciría los tiempos para que una aplicación web llegue al mercado, permitiría producir aplicaciones web de mayor calidad (siempre y cuando la calidad de la infraestructura reutilizable sea buena) y, en definitiva, aprovechar los beneficios que provee el desarrollo utilizando SPL para el dominio de aplicaciones web.

Otro punto a tener en cuenta es que muchos sistemas de software de escritorio han ido migrando a la web, en particular sistemas de tipo de planificación de recursos empresariales (ERP, por sus siglas en inglés, Enterprise Resource Planning), los cuales son considerados SPL en [Bragança, 2008] y sistemas de gestión de clientes (CRM, por sus siglas en inglés, Customer Relationship Management), que también pueden ser considerados como familia de aplicaciones.

Por último, hay que destacar el surgimiento de plataformas de desarrollo colaborativo de aplicaciones (por ejemplo la plataforma gitHub, donde desarrolladores van evolucionando un software en base a resolución de solicitudes de cuestiones pendientes y nuevos requisitos) han producido un elevado incremento en cuanto al tamaño de las aplicaciones web, por lo cual, muchas aplicaciones web se han transformado en familia de aplicaciones que necesitan ser modeladas como tales para aprovechar los beneficios que este tipo de desarrollo provee.

1.2.3. ¿Por qué adoptar MDD?

Según [Vallecillo, 2018], existen varias razones para adoptar MDD, como: proporcionar una manera para que los expertos en dominio especifiquen formalmente su conocimiento y la gente de tecnología defina su implementación (utilizando transformación de modelos); la posibilidad de crear diferentes implementaciones para un mismo modelo; capturar el conocimiento sobre el dominio, la tecnología y su mapeo despejándose de detalles de otras áreas; evitar preocuparse de detalles de implementación al especificar funcionalidad; actuar como un abanico de salida, es decir, MDD puede ser el origen de transformaciones

a diferentes destinos.

1.3. Estado del arte, SPL en el dominio de aplicaciones web

1.3.1. Trabajos relacionados

A partir de la primera década del año 2000, algunos autores comenzaron a explotar la idea de aprovechar los beneficios que provee el desarrollo SPL en el dominio de aplicaciones web. En [Balzerani, 2005] se presenta *Koriandol*, una arquitectura de líneas de productos para diseñar, desplegar y mantener familias de aplicaciones web. El método presentado es basado en componentes, lo cual significa que hace uso de SPL solo a nivel de componentes y no para otros artefactos de una aplicación web.

En el trabajo de [Dolog, 2006] se intenta combinar soporte para personalización y soporte para features comunes y variables en aplicaciones web, de esta manera se presenta un método de desarrollo que incluye ingeniería de dominio e ingeniería de aplicación. La ingeniería de dominio se divide en análisis de dominio y diseño de dominio. En general los modelos definidos en el dominio (incluido modelo de features) representan variabilidad a través de uso de estereotipos de clases y relaciones de tipo OR, AND y XOR.

En [Martinez, 2009] se presenta el uso industrial de un método de desarrollo que combina MDD y SPL para el dominio de sistemas web. Utiliza PLUM (Product Line Unified Modeller), que es una herramienta Open Source que se integra en Eclipse para el diseño, implementación y manejo de SPL permitiendo hacer un desarrollo dirigido por modelos. PLUM captura los requisitos del dominio en un modelo llamado *modelo de decisión* (similar a modelo de features, pero que no tiene en cuenta partes comunes, solo variables).

El trabajo de [Nerome, 2014] se basa en la definición de un meta-modelo de producto de dominio (en base a extensiones a UML y estereotipos) similar a modelo de features y el grueso del aporte está en la definición de un proceso de ingeniería de aplicaciones web. En base a una configuración del modelo de producto de dominio, presenta una arquitectura basada en patrones para desarrollar una aplicación web. Este método no presenta un proceso completo para ingeniería de dominio y está basado en una arquitectura particular para desarrollo de aplicaciones web.

En el trabajo de [Sidi, 2017] se presenta un método de desarrollo orientado a features de aplicaciones web que consta de tres fases: requisitos (incluye producción de casos de uso con variabilidades y modelo de features), análisis (modelos de entidad, de navegación y de presentación) y diseño (modelo de arquitectura). La variabilidad es modelada en el contexto del dominio web usando múltiples vistas a través de modelos de dominio de UML. Los modelos de features que utiliza están basados en clases estereotipadas (por ejemplo *common feature*, *optional feature*).

En [Hwang, 2017] se presenta un enfoque SPL para la construcción sistemática y reutilización de bienes de software de aplicaciones web en pequeñas empresas. Los autores utilizan un enfoque reactivo para expandir y refinar nuevas funcionalidades requeridas por los stakeholders, además hacen uso del patrón de fábrica del framework SPL de SEI (Software Engineering Institute); el cual proporciona una hoja de ruta de alto nivel general para planificar, analizar e implementar una SPL. Este patrón tiene 3 vistas principales: introduction, focus y working área. Define un modelo de variabilidad ortogonal en base a puntos de variación y variantes (en base a la notación de variabilidad introducida en [Pohl, 2005]).

1.3.2. Carencias halladas en la bibliografía

En los trabajos mencionados en la subsección anterior y de manera general en la bibliografía sobre el tema de competencia, hemos identificado las siguientes carencias:

- *O bien no se tiene en cuenta modelo de features (en adición a modelos de dominio) o bien no se automatiza la generación de modelos de features (deben ser construidos manualmente).*

En [Griss, 1998] se menciona un argumento que puede ser generalizado. Allí se dice que durante la Ingeniería de Dominio de requisitos de FA, tanto los modelos de features como las notaciones propias de requisitos (en particular menciona casos de uso de UML) son necesarios; ya que los modelos de features son orientados al especialista en reutilización (este se comunica con el cliente en términos de features) y los modelos de requisitos son orientados al usuario (son usados por los

diseñadores para derivación de productos). Además, el desarrollo de modelo de features y modelos de requisitos por separado puede generar numerosos problemas de inconsistencia entre tales modelos (ver [Casalánguida, 2012]), por lo tanto una buena solución es automatizar la generación de modelos de features a partir de modelos de requisitos de dominio.

- *No se provee notación para modelar requisitos no funcionales con variabilidades.*
En [Norian, 2012] se dice que «al igual que en el desarrollo de software tradicional, el concepto de calidad es crucial para el éxito de las prácticas de la línea de productos de software y, tanto las características funcionales y no funcionales deben participar en el proceso de desarrollo para lograr una línea de productos de software de alta calidad». Por este motivo, consideramos importante el hecho de contar con una notación de requisitos no funcionales que tenga en cuenta variabilidades.
- *No se proveen automatismos para generar configuraciones de modelos de dominio.*
La generación de configuraciones válidas (que cumplan un conjunto de restricciones) es una tarea dura y propensa a errores (ver [Jacob, 2015]), por este motivo en los últimos años han aparecido trabajos que buscan automatizar lo máximo posible este proceso.
- *No se modela ni se tiene en cuenta variabilidad en interfaces de usuario.*
Según [Sottet, 2015], el manejo de variabilidad en interfaces de usuario tiene un impacto significativo en los costos de diseño, desarrollo y mantenimiento de interfaces de usuario. Ya que las interfaces de usuario son una parte importante de las aplicaciones web modernas, consideramos necesario incluir consideración de variabilidad en interfaces de usuario en un proceso de desarrollo de familia de aplicaciones web.
- *No se prescribe un método para modelar interfaz de usuario adaptables automáticamente a diferentes escenarios de usuarios.*
En [Plechawska-Wojcik, 2012] se afirma que «hoy en día es necesario diseñar sitios web para diferentes escenarios de usuarios (dispositivos de acceso), no solo para pantallas diferentes. Además, los patrones de interacción de los usuarios también cambian de un dispositivo a otro, y los sitios web deben adaptarse a estos cambios. La solución actual a este problema es utilizar Diseño web responsivo (RWD), una nueva técnica de diseño web que se ha vuelto muy popular durante los últimos dos años». Por lo cual, consideramos necesario prescribir un método de desarrollo de interfaces de usuario teniendo en cuenta este avance tecnológico.

1.4. Metas y objetivo

Este trabajo se centra en las siguientes metas:

1. Extender el framework NFR (ver [Chung, 2000]) para modelar variabilidades.
2. Extender diagramas de casos de uso y diagramas de actividad de UML para modelar familia de aplicaciones web de manera lo suficientemente expresiva.
3. Aprovechar los beneficios del desarrollo SPL para proceso de desarrollo de familias de aplicaciones web.
4. Aprovechar los beneficios de MDD para producir modelo de features de manera automática a partir de modelos de dominio de la familia de aplicaciones web.
5. Automatizar la producción de configuraciones y la configuración de modelo de features y modelos de dominio en UML.
6. Modelar variabilidad en situaciones propias de interfaces de usuario.
7. En base a modelos de aplicación obtenidos por configuración, proveer un método para producir interfaces de usuario de aplicaciones web adaptables automáticamente a diferentes escenarios de usuarios.

Teniendo en cuenta las metas antes mencionadas, el objetivo principal de este trabajo es:

Producir un proceso de desarrollo de familia de aplicaciones web que incluya una etapa de modelado de dominio, una etapa de configuración, una etapa de modelado de aplicación y que satisfaga las metas enumeradas anteriormente, a partir de la número dos.

1.5. Enfoque presentado

En este trabajo se presenta un enfoque que hace uso de SPL para modelar familia de aplicaciones, durante el cual se genera de manera automática el modelo de features de la familia de aplicaciones a partir de los modelos definidos para dominio.

Para contemplar la *meta 1* se realizó una extensión al framework para requisitos no funcionales NFR (Non Functional Requirements, por sus siglas en inglés) definido en [Chung, 2000]. Dicha extensión incluye agregados al meta-modelo del framework en cuanto a extensiones para modelar variabilidades a través de descomposiciones AND/OR y restricciones de integridad en elementos del meta-modelo para modelar variabilidades. Si bien esta notación no forma parte del proceso definido para desarrollo SPL de familia de aplicaciones web, el desarrollo de este modelo es de importancia ya que puede ser utilizado como soporte de selección de features y para tomar decisiones en aspectos funcionales incluidos en el proceso.

Para alcanzar la *meta 2* se realizaron una serie de extensiones y agregados a UML para modelar variabilidades:

- Se definió una taxonomía para descripción de acciones en diagramas de actividad de UML para aplicaciones web a través de un perfil de *UML*.
- Se definió un perfil UML para descripción detallada de tareas a través de diagramas de actividad.
- Se extendió y mejoró la notación para describir variabilidad en diagramas de casos de uso de [Bragança, 2007].
- Se definió una notación para describir variabilidad en diagramas de actividad.

Además, se definieron meta-modelos para modelar interfaz de usuario abstracta para aplicaciones web (que contempla variabilidad) y acceso a funcionalidades y contenido.

Para satisfacer la *meta 3* se definió un proceso de desarrollo (ver figura 1.1) que involucra las principales actividades de Ingeniería SPL, Ingeniería de Dominio, Configuración e Ingeniería de Aplicación.

Para lograr la *meta 4* se produjeron tres transformaciones de modelos para obtener modelo de features de la familia de aplicaciones de manera incremental y en distintos niveles de abstracción: requisitos de más alto nivel (en base a transformar diagrama de casos de uso de dominio a modelo de features), requisitos más refinados (en base a transformar aspectos variables de descripciones de casos de uso a través de diagramas de actividad de dominio a features que se incorporan al modelo de features) e interfaz de usuario (en base a transformar aspectos variables de modelo abstracto de interfaz de usuario a features que se incorporan al modelo de features).

La *meta 5* se lleva a cabo por medio de la producción de una herramienta web que permite realizar la derivación de una configuración concreta respetando un modelo de features y, en base a esta configuración, configurar modelo de features y modelos de dominio de manera iterativa.

En cuanto a la *meta 6*, se definió un meta-modelo para modelar interfaces de usuario de manera abstracta en base a abstracciones de patrones de elementos de interfaces de usuario de aplicaciones RIA (Rich Internet Applications), de widgets de frameworks responsivos y de elementos de HTML en general. Esto permite que esta notación sea apta para modelar distintos tipos de aplicaciones web, incluyendo aplicaciones RIA. También se definió una sintaxis concreta para los elementos del meta-modelo. Además, se extendió esta notación para modelar variabilidades.

Para satisfacer la *meta 7* se hizo un estudio de los frameworks responsivos más importantes de la actualidad para implementación de interfaces de usuario de aplicaciones web, y en base a una selección de los frameworks más populares se creó un proceso de desarrollo que permite mapear elementos de interfaz de usuario abstracta a widgets de frameworks responsivos y, además, se incluyen guías para realizar implementación.

En la figura 1.1 se muestran las principales actividades del proceso de desarrollo de familia de aplicaciones web. El diagrama se encuentra dividido en tres partes, donde se incluyen las actividades realizadas durante la Ingeniería de Dominio, el proceso de Configuración y la Ingeniería de Aplicación.

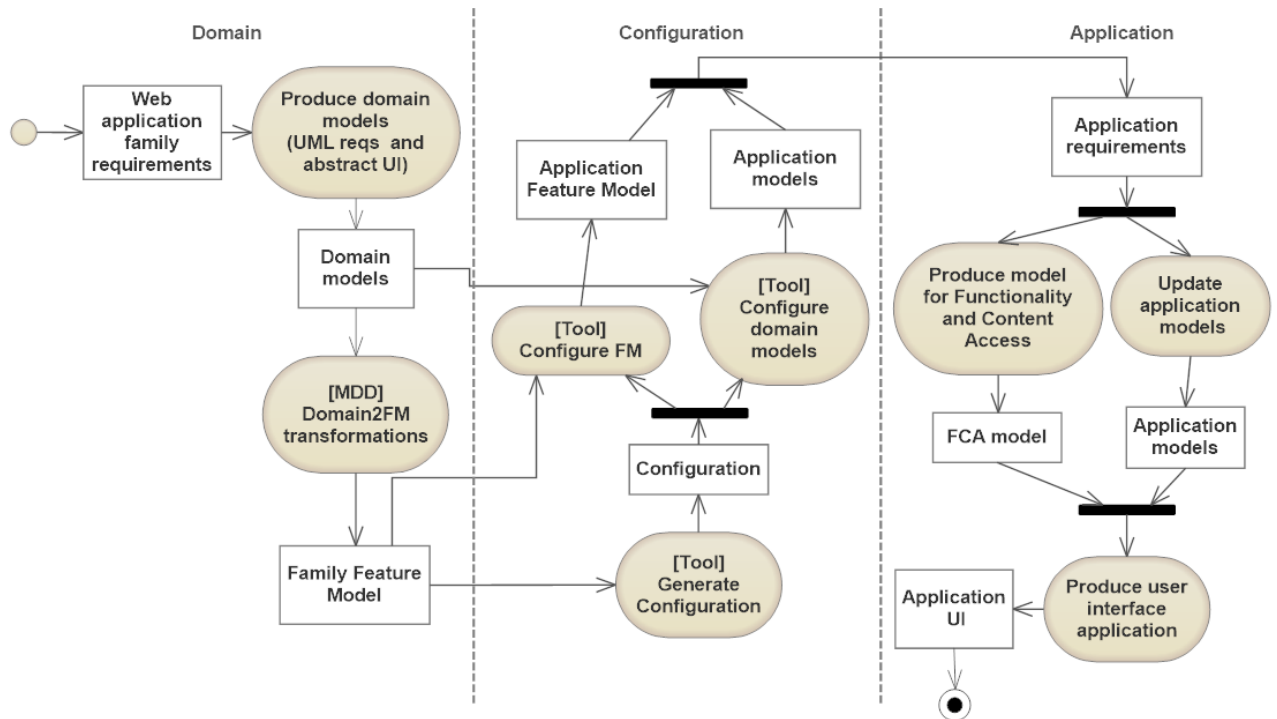


Figura 1.1: Resumen de actividades para el enfoque de desarrollo de familia de aplicaciones web..

El primer bloque (swimlane) del diagrama de actividad, que representa el enfoque de desarrollo de familia de aplicaciones web presentado aquí, se titula *Domain* y hace referencia a las distintas etapas y actividades llevadas a cabo durante la Ingeniería de Dominio. Se recibe como entrada el documento de requisitos de la FAW, donde se deben explicitar las características en común y variables; en base a estos requisitos se producen manualmente (utilizando las notaciones con variabilidades definidas) el modelo de casos de uso de dominio, los diagramas de actividad de dominio que describen los casos de uso y la interfaz de usuario abstracta de dominio para cada caso de uso. Los modelos de dominio definidos sirven como entrada a la transformación de modelos *Domain2FM*, que como resultado arroja el modelo de features de la FAW, el cual incluye distintos niveles de abstracción (subsistemas, paquetes, casos de uso, descripciones de caso de uso, interfaz de usuario). El modelo de features generado de manera automática sirve tanto como para validar con el cliente los requisitos de la FAW como para facilitar la siguiente etapa de automatizar la configuración de los modelos de dominio.

El siguiente bloque del diagrama de actividad del enfoque se titula *Configuration* e incluye las etapas y actividades realizadas durante el proceso de Configuración de modelos de dominio. Tomando como entrada el modelo de features de la familia de aplicaciones web generado de manera automática, mediante una herramienta web construida, se realiza la selección de características deseadas para una aplicación dada, para luego generar el modelo de features de la aplicación (con todas features obligatorias) y los modelos de aplicación, a partir de los modelos de dominio y la configuración definida.

El tercer bloque, titulado *Application*, comprende las etapas que el enfoque abarca en cuanto a Ingeniería de Aplicación. En base a los modelos de aplicación obtenidos durante el proceso de configuración y los requisitos particulares de una aplicación, se realizan dos actividades las cuales pueden ser desarrolladas en paralelo. La primera actividad es la producción del modelo de acceso a funcionalidades y contenido, donde se define una interfaz de usuario abstracta para acceder a la funcionalidad y el contenido de la aplicación, desde un punto de vista estático y dinámico. La segunda actividad tiene que ver con la actualización de los modelos de aplicación obtenidos por medio del proceso de configuración, de acuerdo

a los requisitos particulares de la aplicación a desarrollar. Finalmente, se produce la interfaz de usuario responsiva de la aplicación web.

1.6. Visión general

La tesis se estructura en cinco partes y tres apéndices. La primer parte presenta las notaciones utilizadas, la segunda y tercer parte tratan de modelado de dominio y configuración, respectivamente; la cuarta parte incluye el desarrollo de aplicaciones y en la quinta parte se brinda la conclusión del trabajo.

La *parte I* incluye la presentación de las notaciones de modelado utilizadas; ya sean notaciones definidas, notaciones existentes o notaciones existentes extendidas. Esta parte se divide en dos capítulos: *notaciones de modelado de aplicaciones*, donde se incluyen notaciones utilizadas o definidas para desarrollo de aplicaciones (que no incluyen variabilidades) a nivel de requisitos y diseño; y *notaciones de modelado de dominio*, donde se incluyen notaciones utilizadas o definidas para desarrollo de familia de aplicaciones.

La *parte II* de las tesis incluye el grueso del trabajo realizado en esta tesis y trata sobre desarrollo de modelos de dominio de familia de aplicaciones web y generación de modelo de features a partir de ellos. El primer capítulo de esta parte presenta el *proceso de desarrollo de modelos de dominio y generación de modelos de features*. El capítulo dos trata sobre *desarrollo de modelos de requisitos de dominio*. El tercer capítulo de esta parte comprende la *generación automática de modelo de features a partir de modelos de requisitos de dominio*. El siguiente y último capítulo de esta parte trata acerca de *desarrollo de modelos de diseño y generación automática de features a partir de ellos*.

La *parte III* de este trabajo se ocupa del proceso de configuración de modelo de dominio de aplicaciones web, es decir, como obtener modelos de aplicación a partir de selección de features y modelos de dominio. El primer capítulo de esta parte presenta el *proceso de configuración de modelos de dominio*. Los dos capítulos restantes tratan de *configuración de modelo de features* y *configuración de modelos de dominio*, respectivamente.

La *parte IV* incluye distintas consideraciones para el desarrollo de aplicaciones, incluyendo desarrollo de modelos de aplicación e implementación de interfaz de usuario responsiva. Esta parte incluye tres capítulos: *actualización de modelos configurados en base a requisitos de la aplicación*, *desarrollo de modelo de acceso a funcionalidades y contenido* e *implementación de interfaz de usuario de aplicaciones web responsivas*.

En la *parte V* se incluye una conclusión al trabajo realizado, a través de un resumen de contribuciones realizadas, un listado de las publicaciones científicas en ámbitos académicos con referato realizadas a lo largo de la realización del trabajo, una sección de discusión y una sección final de perspectivas y trabajo a futuro.

El *apéndice A* presenta el código fuente resumido de las transformaciones de modelos en ATL (Atlas Transformation Language) realizadas para generación automática de modelo de features de una FA en base a modelos de dominio.

El *apéndice B* incluye un resumen y fragmentos de códigos fuente de la herramienta web desarrollada para producir configuraciones válidas de modelos de features de una FA y para configurar dicho modelos de features, diagramas de casos de uso de dominio y diagramas de actividad de dominio.

Parte I

Notaciones de modelado utilizadas

Capítulo 2

Notaciones para modelado de aplicaciones

Este capítulo incluye notaciones sin variabilidades que han sido utilizadas en el proceso de desarrollo de aplicaciones web o que han sido definidas y publicadas en ámbitos académicos con referato. Algunas de las notaciones presentadas han sido extendidas para modelar variabilidades.

Para la etapa de requisitos de aplicaciones web en esta tesis se utilizan las notaciones de casos de uso y diagramas de actividad de UML. Estas notaciones no están incluidas en este capítulo ya que son de dominio común en la literatura. Además, en la etapa de requisitos también se brinda una breve descripción del framework *NFR* (ver [Chung, 2000]), para tratar con requisitos no funcionales.

En la primer sección se presentan notaciones utilizadas para la etapa de requisitos del proceso de desarrollo de aplicaciones web. Se incluye la definición de una taxonomía para descripción de acciones en diagramas de actividad de UML para aplicaciones web a través de un perfil de *UML*.

En la segunda sección se presentan las notaciones utilizadas para la etapa de diseño del proceso de desarrollo de aplicaciones web. Se comienza presentando una notación definida para el modelado de acceso a funcionalidades y contenido en aplicaciones web, la cual incluye representación estática y representación dinámica de dicho asunto. Luego, se presenta una notación definida para modelado de interfaz de usuario abstracta de aplicaciones web, se brinda una representación gráfica de las clases definidas para el meta-modelo de la notación, definición de la semántica de los elementos del meta-modelo, la sintaxis concreta de la notación y la definición de una notación de eventos sobre sus elementos. Finalmente, se incluye la definición de un perfil para descripción detallada de tareas. Si bien esta última notación no ha sido incluida en el proceso de desarrollo de familia de aplicaciones web, debido a que en este trabajo nos preocupamos principalmente por la implementación de interfaces de usuario, puede ser utilizada en caso de querer definir con más detalle las tareas y operaciones incluidas en una aplicación a desarrollar.

2.1. Requisitos

2.1.1. Taxonomía para describir acciones de diagramas de actividad de Aplicaciones Web

Los diagramas de actividad de UML (ver [OMG, 2009]) son utilizados para modelar aspectos dinámicos de sistemas en general, estos permiten describir el flujo entre actividades de un sistema o partes de un sistema; ya sea de manera secuencial, paralela, en rama o concurrente. A su vez, las actividades pueden ser descritas en detalle de la misma manera a través de acciones. Una acción representa un paso simple dentro de una actividad.

En el dominio de aplicaciones web es posible identificar y definir cada tipo de cada acción realizada en la interacción del usuario con la aplicación web, pero los diagramas de actividad, por su naturaleza, no brindan esta distinción. Con este fin, en este trabajo hemos elaborado un perfil UML que incluye una taxonomía para describir acciones de actividades de diagramas de actividad de UML, a través de los siguientes estereotipos:

- «search» **action**: representa una consulta sobre una base de datos (por ejemplo relacional, no

relacional, *XML*) u obtención de información (por ejemplo a través de un servicio REST). Puede incluir pines de entrada (elementos *InputPin*) para los parámetros de la consulta y pines de salida (elementos *OutputPin*) para los resultados de la consulta.

- «**job**» **action**: es un elemento del tipo *Call Behavior Action* cuya actividad representa una acción autónoma del sistema. Puede incluir pines de entrada para los posibles parámetros de la tarea y pines de salida para el resultado de la tarea.
- «**input**» **action**: representa la provisión de información por medio de un actor humano. Puede incluir pines de salida para expresar los valores ingresados por el usuario. Para el nombre de estas acciones se pueden utilizar verbos tales como seleccionar, elegir, proveer, ingresar.
- «**input and suggest**» **action**: representa la provisión de información por medio de un actor humano y la utilización de sugerencias automáticas del sistema seleccionables por el actor humano. Puede incluir pines de salida para expresar los valores ingresados por el usuario. Para el nombre de estas acciones se pueden utilizar verbos como seleccionar, elegir, proveer, ingresar.
- «**output request**» **action**: representa al sistema requiriendo provisión de una o más entradas de un actor humano. Para el nombre de estas acciones se pueden utilizar verbos tales requerir, preguntar, solicitar, pedir.
- «**output content**» **action**: representa contenido que es mostrado por el sistema (por ejemplo resultado de una tarea, resultados de una consulta, notificaciones del sistema). Para el nombre de estas acciones se pueden utilizar verbos tales como mostrar, visualizar, presentar.
- «**output message**» **action**: representa al sistema mostrando un mensaje al usuario (puede ser un mensaje de error, de éxito, de estado, de ayuda o de advertencia).
- «**output media**» **action**: representa al sistema mostrando un archivo de medios (por ejemplo video, audio, animación, presentación) y los controles de reproducción del dispositivo que reproduce el archivo, para interactuar con el usuario.

Si una acción debe ser ejecutada como una transacción se debe adjuntar el estereotipo «**transaction**» a la acción. Esto significa que para estos tipos de acciones son válidas las propiedades *ACID* (Atomicidad, Consistencia, Aislamiento y Durabilidad).

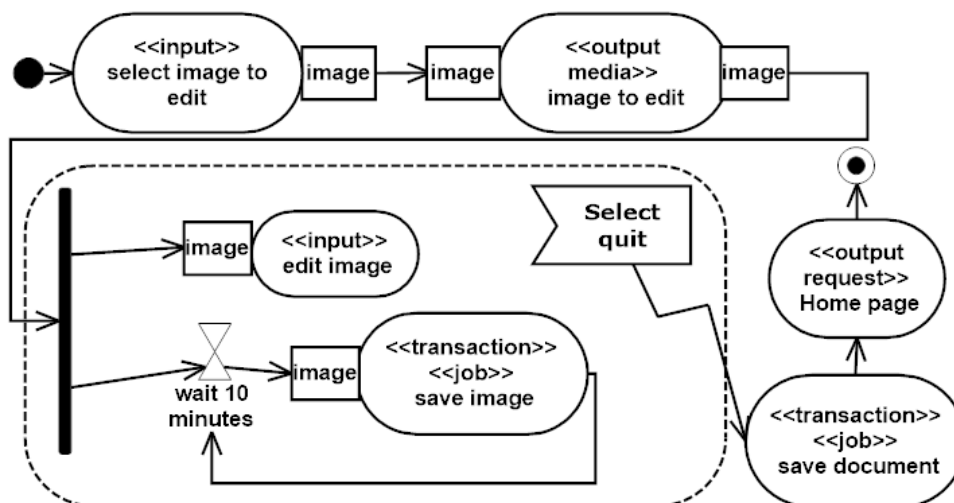


Figura 2.1: Ejemplo de uso del perfil para describir acciones en actividad Editar imagen con auto guardado de aplicación web.

En la figura 2.1 se muestra un ejemplo de aplicación del perfil para describir acciones de actividades para una actividad de edición de imágenes que se auto guardan cada diez minutos para cierta aplicación web. La actividad comienza con una acción de tipo input *select image to edit* que representa la selección

del usuario de la imagen que desea editar. Luego el sistema muestra esa imagen, situación representada con el estereotipo de tipo output media para la acción *image to edit*. El flujo continúa con dos acciones que son ejecutadas concurrentemente, una de tipo input representando la acción de editar imagen por parte del usuario (*edit image*) y otra de tipo job transaction (*save image*), que representa la acción automática del sistema de almacenar la imagen editada de manera automática cada 10 minutos. Si el usuario interrumpe la edición, seleccionando salir (*select quit*), se ejecutará una acción con tipos job y transaction (*save document*), que representa la acción del sistema de almacenar el documento que incluye la edición de la imagen. Finalmente, se incluye una acción de tipo output request, que representa la requisitoria del sistema de mostrar la página principal de la aplicación web.

2.1.2. Framework NFR(Non-Functionals Requirements)

El framework NFR (ver [Chung, 2000] y [Chung, 2000b]) utiliza requisitos no funcionales tales como seguridad, usabilidad y precisión, entre otros. Estos requisitos no funcionales dirigen el proceso de desarrollo de sistemas de software. El objetivo primordial del framework NFR es poner a los requisitos no funcionales más importantes en la mente del desarrollador.

Los pasos principales en su proceso de desarrollo son:

- Adquirir o acceder a la información acerca de:
 - El dominio particular y el sistema que se está desarrollando.
 - Requisitos funcionales del sistema particular.
 - Tipos particulares de requisitos no funcionales, y técnicas de desarrollo asociadas.
- Identificar requisitos no funcionales específicos del dominio.
- Descomponer requisitos no funcionales.
- Identificar operacionalizaciones (posibles alternativas de diseño para cumplir con los requisitos no funcionales).
- Tratar con:
 - Ambigüedades.
 - Compensaciones y prioridades.
 - Interdependencias entre requisitos no funcionales y operacionalizaciones.
- Seleccionar operacionalizaciones.
- Justificar decisiones con razones de diseño.
- Evaluar el impacto de las decisiones.

Los pasos no necesitan hacerse de manera secuencial, y puede ser necesario iterar sobre ellos varias veces durante el proceso de diseño.

Softgoal Interdependency Graph (SIG)




El resultado de la aplicación del Framework NFR puede ser visualizado en términos de construcción iterativa e incremental, elaboración, análisis y revisión de un grafo de interdependencias de metas.

La mayoría de los conceptos del Framework NFR tienen una representación gráfica en los SIG. Los principales requisitos son mostrados como *softgoals* (metas) al nivel más alto del grafo. Los softgoals son conectados a través de enlaces de interdependencia, los cuales son representados con líneas, frecuentemente con flechas. Los softgoals tienen *labels* (etiquetas) asociadas, que representan el grado en que un softgoal es logrado. Las interdependencias permiten definir refinamientos de softgoals padres en softgoals hijas más específicas. Estas también muestran la *contribution* (contribución, representa el impacto) de las softgoals hijas a las softgoals padres.

Para determinar cuando las softgoals son logradas se utiliza un procedimiento de evaluación (algoritmo de etiquetado); en el que se consideran etiquetas, contribuciones y decisiones del desarrollador.

Existen tres tipos de softgoals: softgoals NFR, softgoals operacionalizadas (operacionalizaciones) y argumentos. Las softgoals NFR representan requisitos no funcionales de alto nivel que se deberán satisfacer en la aplicación. Las contribuciones son relaciones entre el softgoal padre y los softgoals derivados del padre que son resultado de su refinamiento y el tipo de la contribución. Las contribuciones pueden ser de dos tipos. El primer tipo es aquel en el cual los softgoals se descomponen en un grupo de softgoals por medio de contribuciones AND (el padre es satisfactorio si todos los “hijos” son satisfactorios) u OR (el padre es satisfactorio si alguno de los “hijos” es satisfactorio). El otro tipo de contribución relaciona un solo softgoal con el padre asumiendo los siguientes valores : totalmente negativo ("--" o BREAK), parcialmente negativo ("- " o HURT), ("++" o MAKE) y parcialmente positivo ("+" o HELP).


Tipos de softgoals:

- **NFR** 
- **Operacionalización (técnica de satisfacción)** 
- **Argumentos (soporte/explicación de elección)** 

Softgoal := Informal Sg | Formal Sg

Formal Sg := Type[Topic]

Tipos de contribuciones:

- **AND (descomposición)** 
- **OR (alternativas)** 

Make (++)>>Help(+)>>Hurt(-)>>Break(--)

Etiquetas (evaluación de softgoals/contribuciones):

- **satisfied** 
- **denied** 
- **conflicting** 
- **undetermined** 

Figura 2.2: Resumen de la notación utilizada para los SIGs

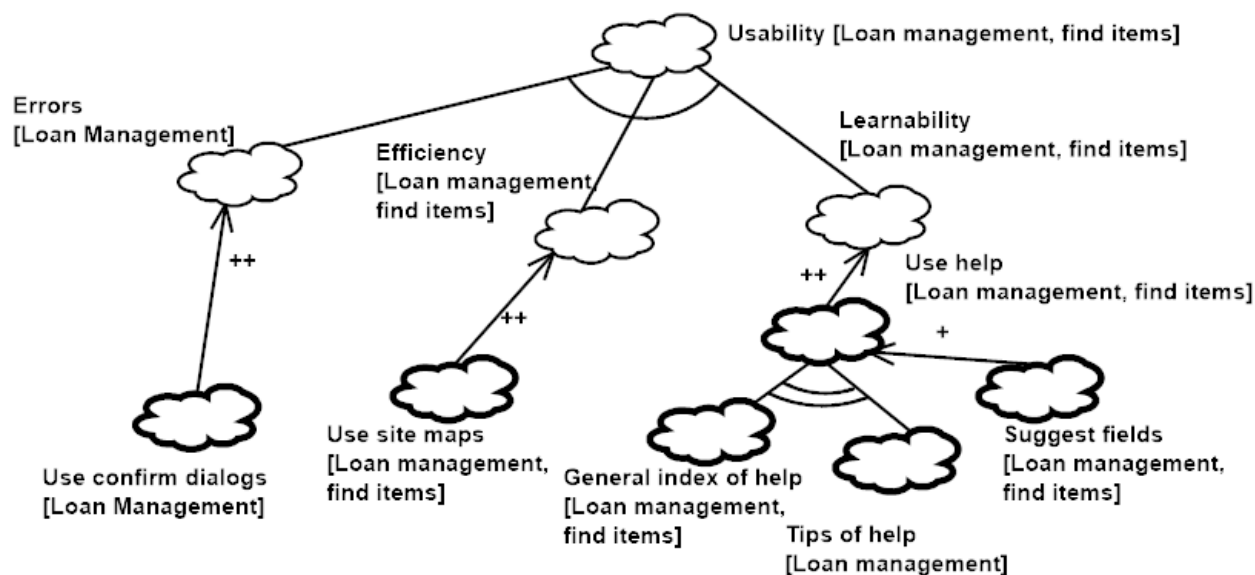


Figura 2.3: Ejemplo de SIG para el requisito no funcional Usabilidad para las operaciones de Préstar de libro y Buscar ítems de una aplicación de Biblioteca Online.

Ejemplo La figura 2.3 muestra un ejemplo de SIG del requisito no funcional *Usabilidad* para una aplicación de Biblioteca online, en particular para las operaciones de Prestar libro y Buscar ítems. El softgoal para Usabilidad (*usability*) se descompone en softgoals más específicos, las cuales son cualidades o decisiones de diseño que ayudan a alcanzar al requisito no funcional (usabilidad), estos son: *errors* (manejo de errores), *efficiency* (consideración de eficiencia) y *learnability* (consideración de facilidades para entender el sistema). La operacionalización *Use confirm dialogs* se refiere al uso de diálogos de confirmación en la operación de Prestar libro y contribuye de manera *make* al softgoal *errors*. La operacionalización *use site maps* se refiere al uso de mapas del sitio y contribuye de manera *make* al softgoal *efficiency*. La operacionalización *Use help* se refiere al desarrollo de soporte de ayuda y contribuye de manera *make* al softgoal *learnability*. Las operacionalizaciones *general index of help* (índice general de ayuda) y *tips of help* (sugerencias de ayuda) representan un refinamiento alternativo de la operacionalización *Use help*. La operacionalización *suggest fields* se refiere al uso de campos de entrada con sugerencias autocompletadas y contribuye de manera *help* al softgoal *use help*.

2.2. Diseño

2.2.1. Notación para acceso a funcionalidades y contenido de interfaz de usuario

Background

Una parte importante de la interfaz de usuario de aplicaciones web es la interfaz de usuario para acceso a funcionalidades y contenido (UIFCA, por sus siglas en inglés); esta considera la organización estructural de los elementos utilizados para proveer funcionalidades y contenido al usuario, y el cambio dinámico de dichos elementos accesibles al usuario.

A partir de la decisión de modelar interfaz de usuario para acceso funcionalidades y contenido, surgen algunos interrogantes: *¿qué se debería capturar acerca de la UIFCA para obtener un buen modelo de acceso a funcionalidades y contenido?*, *¿qué abstracciones son necesarias para esto?*

En este trabajo proponemos capturar la esencia de la UIFCA y abstraernos de: la descripción de funcionalidad (esto se captura en la especificación de la interfaz de usuario abstracta de un caso de uso), la especificación de elementos de interfaz de usuario para mostrar contenido, la especificación de elementos de interfaz de usuario para ingreso de datos y la especificación de elementos para estructuras de acceso a información (por ejemplo, menú, visita guiada, índice, ruta de navegación).

Las consecuencias, consideradas como ventajas, de estas abstracciones son: 1) menos aspectos para pensar en el desarrollo de un modelo de acceso a funcionalidad y contenido (ya sea desarrollo inicial o desarrollo en base a cambios por validación), por lo tanto menos tiempo de desarrollo; 2) como consecuencia del punto anterior, es más fácil considerar cambios en los requisitos de usuarios; 3) el modelo de acceso a funcionalidades y contenido no se altera si existen cambios en la descripción de interfaz de usuario de contenido o ingreso de información (ya que esto es abstraído); 4) se permite separación de intereses en la descripción del modelo de acceso a funcionalidades y contenido: la descripción de la interfaz para funcionalidades se separa de la descripción de la interfaz para mostrar contenido, ingreso de información y estructuras de acceso a información.

La esencia de la UIFCA tiene una importancia central para el cliente, ya que refiere a la organización y comportamiento global de la interfaz de usuario del sistema y representa la “fachada” de presentación de la interfaz de usuario; por lo tanto, idealmente, la descripción de la esencia de la UIFCA debe ser comprensible para el cliente (es decir, no debe incluir elementos correspondientes a conceptos técnicos desconocidos por los clientes, y sus elementos deben representar instancias de conceptos bien conocidos para los clientes). Tener un modelo de acceso a funcionalidades y contenido para la esencia de la UIFCA entendible por los clientes permite al cliente: validar el modelo de acceso a funcionalidad y contenido y proporcionar partes de dichos modelos (por ejemplo, partes relacionadas con conceptos y funciones innovadoras, partes específicas de un dominio que no son fáciles de comprender para los analistas).

Además, es necesario contar con una notación para construir modelos considerando la variación dinámica del conjunto de funcionalidades y contenido que son accesibles al usuario, ya que en las aplicaciones actuales la interacción es continua y la interfaz de usuario va variando por medio de las interacciones. Esta notación debe ser lo suficientemente expresiva, es decir, debe considerar un conjunto rico de acciones para representar las modificaciones dinámicas de la UIFCA.

Las notaciones de modelado encontradas en la literatura para aplicaciones web no son suficientes para contemplar la esencia de la UIFCA y no satisfacen los requisitos de abstracción que hemos detallado. En general, los métodos para aplicaciones web no brindan la posibilidad de que el cliente sea el encargado de definir parte del modelo de acceso a funcionalidades y contenido y valide la parte del modelo hecha por los desarrolladores.

El objetivo de esta sección es la definición de una notación para modelo de acceso a funcionalidades y contenido en el dominio de aplicaciones web que permita capturar la esencia de la UIFCA, y que tal notación: sea independiente de modalidad y tecnología de implementación; se abstraiga de detalles de descripción de elementos de interfaz de usuario para: mostrar contenido, ingreso de información y estructuras de acceso a información; sea de fácil entendimiento y comprensión para el cliente e incluya un conjunto rico de acciones para representar modificaciones de la UIFCA.

En esta sección presentamos las clases del meta-modelo desarrollado para definir el acceso a funcionalidades y contenido de una aplicación. El mismo ha sido llamado *WAFCA* (*Web Application Functionality and Content Access*) y se abstrae de modalidad, widgets y dispositivos específicos. Un *user role site view* es la parte de la interfaz gráfica de una aplicación utilizada para un rol específico de usuario. Cada aplicación web es estructurada en un conjunto de *user role site views*. Para representar cómo una aplicación es organizada para acceso a funcionalidad y contenido, se utiliza un modelo respetando el meta-modelo *WAFCA*.

Trabajo relacionado

Las notaciones de modelado halladas en la literatura que se abstraen de elementos de interfaz de usuario para mostrar contenido, ingreso de información y estructuras de acceso a información; y que pueden ser utilizadas para describir interfaz de usuario o requisitos para funcionalidad o acceso a contenido son: el metadomodelo OOWS 2.0 RIA de [Valverde Giromé, 2010] y las diagramas de casos de uso de [Rosado da Cruz, 2010]. Sin embargo, estas notaciones no se abstraen de descripción de funcionalidades. OOWS 2.0 permite describir la interfaz de usuario para funcionalidad y los diagramas de casos de uso de [Rosado da Cruz, 2010] pueden tener casos de uso agregados. Las otras notaciones de modelado de interfaz de usuario halladas consideran menos requisitos de abstracción.

Las notaciones más poderosas que hemos encontrado en el dominio de aplicaciones web para describir cambio dinámico del conjunto de funcionalidades accesibles y elementos de contenido disponibles para el usuario son: el meta-modelo de interacción de OOWS 2.0 de [Valverde Giromé, 2010], el modelo de diálogo de MARIA (ver [Paternò, 2009]); las dos notaciones contemplan acciones para habilitar, desha-

bilitar, ocultar y mostrar elementos; sin embargo, no consideran acciones para abrir, remover e intervalo. Además, tienen una carencia en cuanto a sintaxis visual para las acciones, lo que le resta facilidad de entendimiento al cliente. Los otros modelos abstractos para aplicaciones web que hemos hallado son modelos de navegación (por ejemplo UWE en [Koch, 2012] y WebRatio Ajax en [Brambilla, 2010]), pero éstos no permiten expresar reglas ECA con los tipos de acciones definidos en este trabajo.

Las notaciones de modelado encontradas para aplicaciones web que capturan parte de la esencia de la UIFCA y que pueden ser usadas para que los clientes construyan parte del modelo de acceso a funcionalidades y contenido son: los diagramas de casos de uso de [Rosado da Cruz, 2010] y el modelo de interfaz de usuario abstracta de UsiXML (ver [Martínez Ruiz, 2007]), debido a que tienen una sintaxis concreta que probablemente es legible por el cliente para modelar parte de la estructura de la UIFCA. Además, el modelo de navegación de UWE (ver [Koch, 2008] y [Koch, 2012]) permite capturar parte de la esencia de la UIFCA pero incluye algunos conceptos que podrían dificultar el entendimiento de la notación por parte de los clientes si los mismos no están habituados a algunas terminologías técnicas (como caminatas guiadas, índices, menús). El resto de notaciones de modelado abstractas para aplicaciones web encontradas han sido consideradas como no fáciles de entender por los clientes, debido a que, o bien tienen conceptos técnicos críticos, o bien tienen conceptos técnicos menores pero no son notaciones visuales o con sintaxis concreta.

Vista estática de acceso a funcionalidades y contenido

Cada modelo *WAFCA* incluye una vista estática que se utiliza para describir cómo una vista de rol de usuario es organizada en elementos de granularidad más gruesa (ver meta-modelo en figura 2.4). Se definió una sintaxis concreta para esta parte del meta-modelo (ver figura 2.5), la cual puede ser vista como una pantalla que contiene regiones y elementos (para acceso). Asumimos que los clientes entienden y pueden producir estos tipos de artefactos, que lucen de manera similar a una maqueta de un sitio.

A continuación cada elemento del meta-modelo de la figura 2.5 es descrito en detalle.

Site View: representa un *user role site view*, con la definición brindada en la sección Background. Un *Site View* contiene una jerarquía de *Groupings* y *Group members*.

Grouping: representa una pieza de la interfaz gráfica usada para agrupar elementos, que pueden ser otros *Grouping* o *Group Members*. Los miembros de un *Grouping* pueden estar presente todos al mismo tiempo, (atributo *type=All*) o solo uno presente en un instante de tiempo (*type = Alternative*). Un *root grouping* es un *Grouping* en la raíz de la jerarquía. Los elementos de los *Grouping* son representados con rectángulos de diferentes figuras de acuerdo al tipo de *Grouping* (ver Figura 2.5). Algunos *Groupings* alternativos tienen condiciones sobre todos sus miembros (se usa la meta-variables *cond*). Tales condiciones son fórmulas proposicionales cuyas proposiciones tienen nombres de *Groupings/Group members* (una proposición es verdadera si y solo si su elemento correspondiente está presente).

Group member: representa los tipos de elementos de interfaz de los cuales se pueden componer los *Grouping*. Pueden ser elementos de acceso, elementos vacíos, elementos para realizar entradas y elementos de contenido.

Input: tipo de elemento para representar la provisión de alguna entrada en la aplicación.

Access: tipo de elemento para representar acceso a elementos de la aplicación, que pueden ser tareas o agrupamientos.

Task: representa acceso para a tarea, un caso de uso, un servicio o un comando.

Access to Grouping: representa acceso a navegación dentro un *emphGrouping*.

Empty: representa un conjunto vacío de elementos, es decir que no contiene nada.

Content: se utiliza para representar contenido. Puede ser contenido que incluye interacción con el usuario o contenido de solo lectura. Contiene un atributo llamado *periodicRefresh* de tipo *booleano* que hace referencia a si la información del elemento cambia periódicamente o no.

Content with Interaction: representa al tipo de contenido que permite interacción con el usuario.

Read Only Content: representa al tipo de contenido de solo lectura.

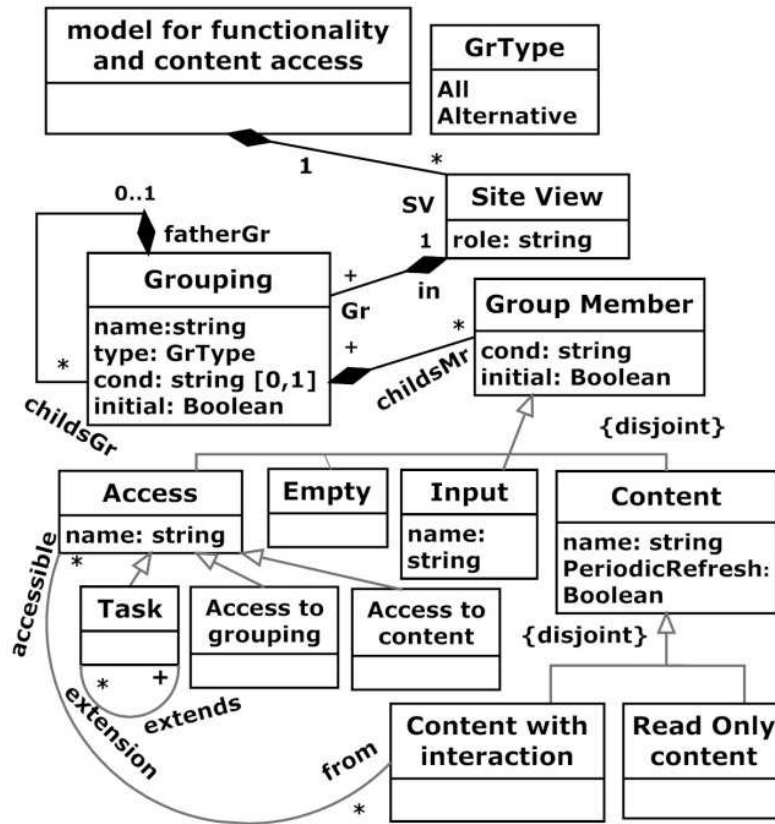


Figura 2.4: Meta-modelo para acceso a funcionalidades y contenido. Parte para expresar vista estática.

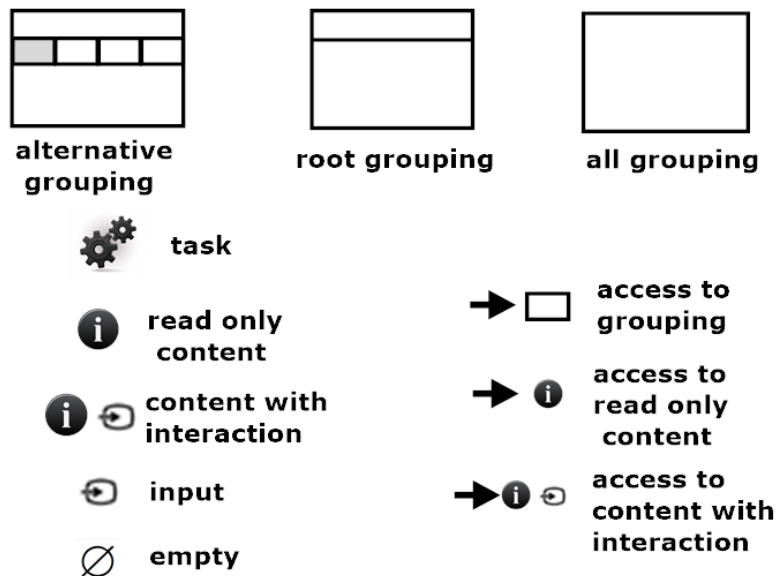


Figura 2.5: Sintaxis concreta para elementos de vista estática de modelo de acceso a funcionalidades y contenido.

Consideraciones de modelado. Para seleccionar un elemento *Access* dentro de un elemento *Content* se utiliza la meta-relación con roles *from* y *accessible*. Para acceder desde el interior de una tarea a las tareas de extensión (que no son necesarias para que exista la tarea extendida) utilizamos la meta-relación con los roles *extends* y *extension*. Cuando un *Grouping* de tipo alternativo *G* no está presente y si es presentado, es necesario decir que un hijo *E* de *G* es presentado por defecto; para expresar esto se incluye sobre *E* *initial = true*. Se representa gráficamente a un miembro *E* de *G* con *initial=true* con el rectángulo de *E* coloreado de gris.

Ejemplo La figura 2.6 muestra el modelo de vista estática para el grouping *User Agent* para una aplicación de correo electrónico online. Este grouping representa el agrupamiento raíz de la aplicación. *Work*, *Commands*, *Lists* son groupings alternativos. El grouping *Lists* contiene dos elementos de tipo *Content*: *inbox* y *sent mails*. El grouping *Commands* contiene el elemento de tipo *Task* *Refresh*, el grouping *Refresh* y el elemento *Empty*. Los elementos *compose* (para redactar y enviar correos), *search* (para buscar correos electrónicos en una lista) son ejemplos de tareas del grouping *User Agent*. Además, hay dos elementos de tipo *Access to grouping*: *settings* (representa el acceso al grouping de la configuración de la aplicación) y *Account group* (representa el acceso al grouping para la configuración de la cuenta de correo electrónico).

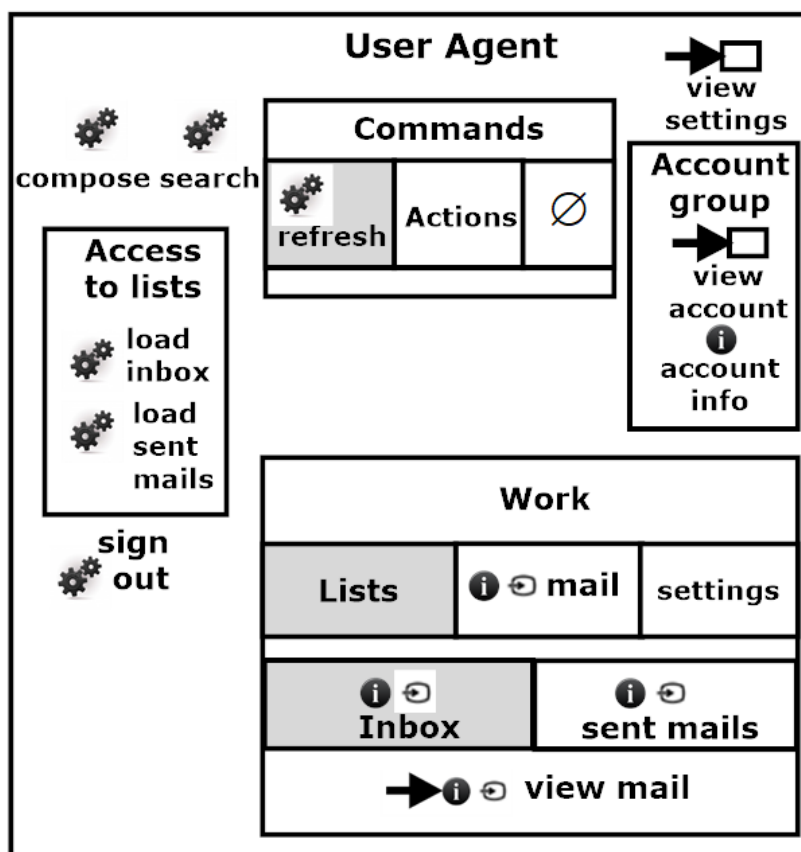


Figura 2.6: Ejemplo de modelo de vista estática de acceso a funcionalidades y contenido para grouping *User Agent* de aplicación de correo electrónico online.

Vista dinámica de acceso a funcionalidades y contenido

Para expresar requisitos para el cambio dinámico a los usuarios de las funcionalidades accesibles y elementos de contenido, *WAFCA* incluye una vista dinámica, la cual provee un conjunto de elementos de modelado que se pueden ver en la figura 2.7. Tales elementos son utilizados para representar un conjunto de requisitos de la forma: *<interacción con usuario u otro evento, respuesta del sistema>*, donde respuesta del sistema consiste de una o más acciones que modifican el conjunto actual de elementos de

contenido y funcionalidades accesibles.

A continuación se detalla el meta-modelo definido para la notación de modelado de vista dinámica del acceso a funcionalidades y contenido, y luego la sintaxis concreta del meta-modelo. Después, cada elemento del meta-modelo es descrito en detalle.

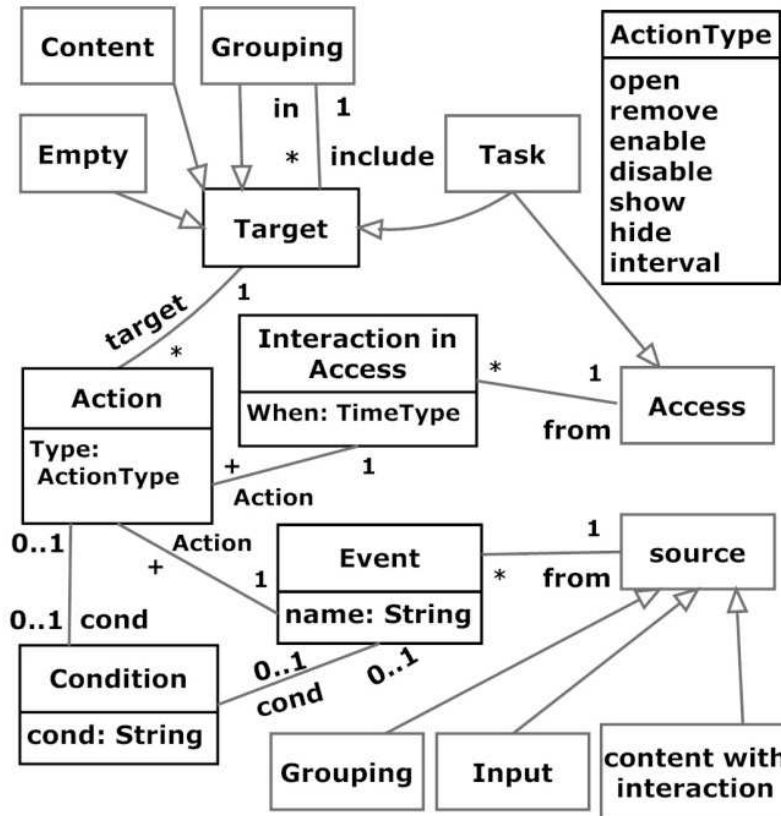


Figura 2.7: Meta-modelo para acceso a funcionalidades y contenido. Parte para expresar vista dinámica.

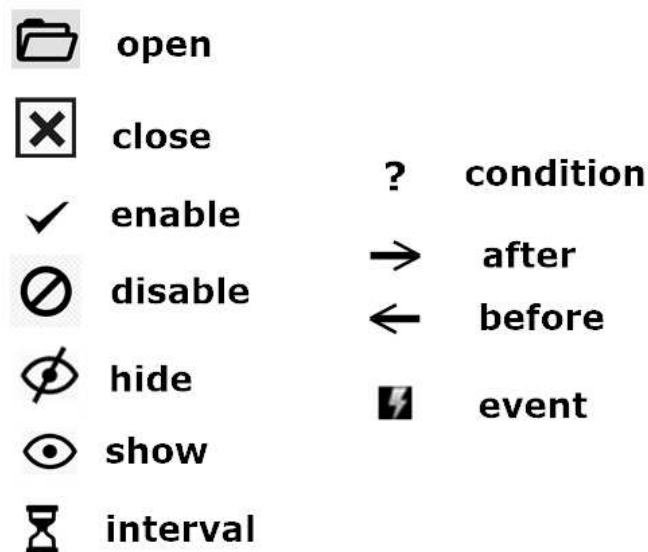


Figura 2.8: Sintaxis concreta para elementos de vista dinámica de modelo de acceso a funcionalidades y contenido.

Cada acción del sistema es modelada por el elemento *Action* y representa la respuesta del sistema a un requisito. Las acciones tienen un elemento destino, llamado *Target* (puede ser un elemento *Content*, un elemento *Grouping*, un elemento *Task* o un elemento *Empty*), y un tipo (atributo *type*) que puede tener los valores *open*, *remove*, *enable*, *disable*, *show*, *hide*, *interval*. El elemento destino es presentado solo durante un intervalo de tiempo.

Un requisito dice qué acciones sobre los destinos (targets) deben ser realizadas después de la aparición de algún evento. Los eventos son especificados con la clase del meta-modelo *Event*. Si un evento tiene asociada una condición para que las acciones asociadas al evento sean realizadas, ésta condición debe ser válida y es especificada mediante la clase *Condition*. Los elementos del tipo *Event* pueden ser: a) una interacción del usuario con un elemento *source* u otro evento asociado a un *grouping*; b) una selección de un elemento *Access* (*When=before*) o la finalización de una ejecución de un *Access* (*When=after*).

Un requisito se representa gráficamente con una flecha que tiene una o más puntas desde el elemento donde ocurre el evento a los destinos (*Targets*). El ícono de tipo de acción es ubicado cerca de la punta de la flecha. Los elementos de tipo *Event* son incluidos en el inicio de la flecha de requisitos. Una condición (elemento *Condition*) es representada con un signo de interrogación (?) y el texto de su descripción. Una condición asociada a un evento es colocada cerca del inicio de una flecha y una condición asociada a una acción es ubicada cerca de la punta de la flecha.

Supongamos que un *Target* es un *Grouping* *G*. Si *G* no está asociado con otros elementos *Target* (usando la asociación *include*) entonces *G* es presentado con los elementos predeterminados de sus agrupamientos alternativos; si no, se presentan los elementos *Target* asociados con *G* en lugar de los elementos predeterminados de sus agrupamientos alternativos. Un elemento *Target* asociado con *G* es representado con una flecha con punta en forma de rombo desde *G* al elemento *Target* asociado.

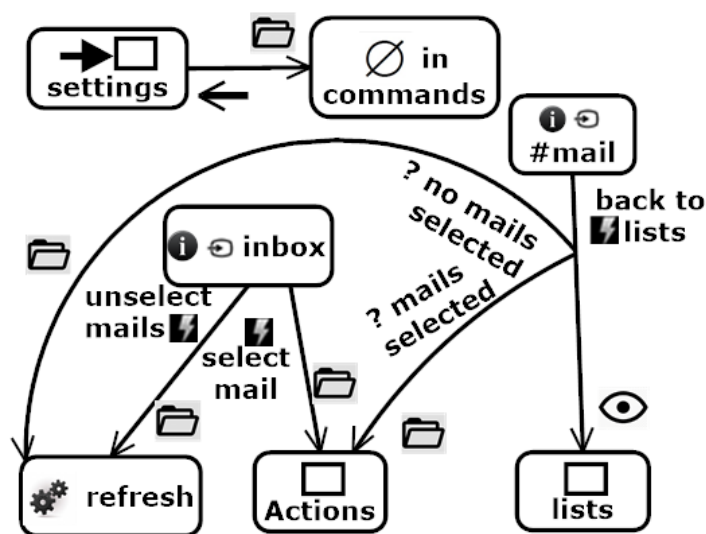


Figura 2.9: Ejemplo de modelo de vista dinámica de acceso a funcionalidades y contenido para algunos elementos del grouping *User Agent* de una aplicación para correo electrónico online.

Ejemplo La figura 2.9 muestra el modelo de vista dinámica para algunos elementos del grouping *User Agent* de una aplicación de correo electrónico online. En la parte superior del diagrama se puede observar que antes de presentar el grouping *settings*, se presenta al miembro *Empty* (del grouping alternativo *Commands*). Cuando el usuario anula la selección de todos los correos en la bandeja de entrada (*unselect mails*), se presenta la tarea *Refresh*, y cuando en la bandeja de entrada no hay correos seleccionados (*no mails selected*) y el usuario selecciona uno (*mails selected*), se abre el grouping *Actions*. Cuando el usuario elige volver a las listas (*back to lists*) en el content *mail*, se muestra el grouping *Lists*.

2.2.2. Notación para diseño abstracto de interfaz de usuario

Background

En este capítulo presentamos una notación que consiste de elementos de diseño abstractos para definir la interfaz de usuario de una aplicación web. Esta notación fue definida incrementalmente en dos iteraciones:

La primera iteración constó de la definición de la notación (ver [Casalánguida, 2013]) denominada *RIA Application Abstract Design (RIAAD)*, por sus siglas en inglés) y su fin era el diseño de interfaz de usuario para aplicaciones RIA. Esta primera versión de la notación incluyó algunos aportes a las notaciones existentes de diseño de interfaces de usuario abstractas para RIA:

- La consideración de elementos de interfaz gráfica editables (elementos básicos y estructuras de contenido);
- La abstracción de patrones conocidos en el dominio RIA, tales como breadcrumb, filtros de enlaces alfanuméricos, barra de navegaciones y una representación apropiada de campos de entrada de datos con sugerencias.
- La representación de elementos de interfaz de usuario para la edición de objetos multimedia (audio, video, imágenes) y documentos (presentaciones, hojas de cálculo, editores de documentos, entre otros).

RIAAD fue definida tomando como referencia las notaciones de interfaz de usuario de *UWE* (ver [Koch, 2008]), *UWE-R* (ver [Filho, 2009]) y *MARIA* (ver [Paternò, 2009]), a las cuales se les realizaron los agregados listados como aportes anteriormente.

La segunda iteración (ver [Casalánguida, 2015b]) significó la definición de una notación de interfaz de usuario abstracta denominada *RIAAD2* y se tomó como referencia *RIAAD*. A partir de *RIAAD* se realizaron algunas modificaciones y agregados con el fin de contemplar los adelantos tecnológicos en el campo, en particular el surgimiento de los frameworks responsivos (conjunto de librerías que permiten desarrollar diseños de interfaz de usuario de sitios web adaptables). Para dicho fin se examinaron una serie de frameworks de diseño web responsivos y por cada uno de ellos se aplicó el siguiente procedimiento: si ya existía un elemento en *RIAAD* que era una abstracción apropiada del elemento de interfaz gráfica del framework, entonces se mantenía dicho elemento; si existía un elemento de interfaz gráfica de *RIAAD* que podría generalizarse para obtener una abstracción apropiada del elemento del framework, entonces se hacía esta generalización; en caso contrario de los anteriores se definía un nuevo elemento para representar una abstracción del elemento del framework. Además se eliminaron algunos elementos de *RIAAD* que no eran lo suficientemente abstractos (demasiado específicos de patrones de código fuente –por ejemplo, textos con sugerencias y filtros de enlaces alfanuméricos–); se añadieron algunos elementos (ícono, botón y diálogo); se modificó la semántica de algunos elementos de *RIAAD*, y se cambió el nombre de algunos elementos de *RIAAD*.

En este trabajo, al tratar con interfaz de usuario para aplicaciones web en general (incluyendo RIAs y considerando frameworks responsivos), hemos dado a la notación el nuevo nombre de *Web Application Abstract User Interface Design (WAAUID)*, por sus siglas en inglés). Además hemos definido una sintaxis concreta para todos los elementos incluidos en la notación, esto aporta más claridad y legibilidad a la hora de visualizar modelos de la notación.

Meta-modelo para notación de diseño abstracto de interfaz de usuario para aplicaciones web

A continuación presentamos las clases del meta-modelo divididas en cuatro partes para mayor comprensión. La primer parte incluye las clases de más alto nivel, incluidos contenedores y distintas estructuras de acceso, de contenido y de interacción. La segunda parte completa algunos elementos de estructuras de acceso no incluidos en la primer parte. La tercer parte completa algunos elementos de estructuras de interacción. La cuarta y última parte incluye los elementos de más bajo nivel, denominados elementos básicos de interfaz de usuario, tales como elementos atómicos para mostrar contenido. Cada elemento del meta-modelo incluye una descripción del mismo.

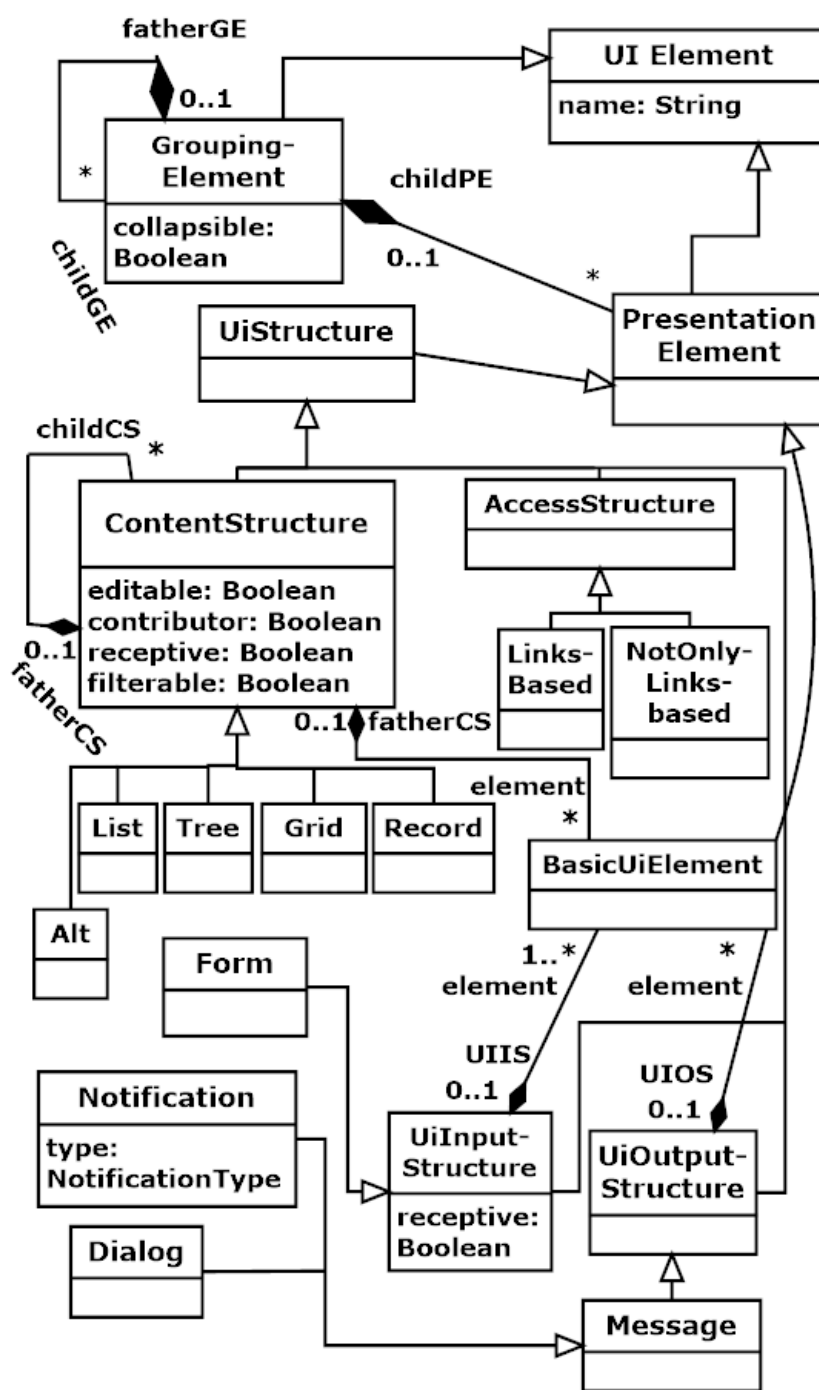


Figura 2.10: Meta-modelo para diseño abstracto de interfaces de usuario. Parte 1.

UI Element: elemento de más alto nivel, puede ser o bien un contenedor de elementos o bien un elemento de presentación. Tiene un solo atributo: *name*, que representa el nombre del elemento.

Grouping Element: elemento para agrupar otros elementos lógicamente sin restricciones de layout. Puede contener anidamientos a otros agrupamientos del mismo tipo o a elementos de presentación. Tiene un atributo de tipo *Boolean* llamado *collapsible* que especifica si el agrupamiento puede ser colapsado o expandido.

Presentation Element: representa elementos que serán presentados al usuario, los cuales pueden ser de dos tipos: estructuras de contenido o elementos básicos.

UiStructure: representa información o datos presentados al usuario de manera estructurada, ya sea a través de estructuras de contenido, de estructuras de acceso a contenido o de estructuras para ingresar o mostrar información.

ContentStructure(CS): representa un elemento de interfaz gráfica, el cual respeta una cierta estructura y es usado para presentar contenido. Un *CS* puede contener elementos selectores, es decir que pueden contener áncoras. Se consideran cinco tipos de *CS*: *List*, *Grid*, *Tree*, *Alt* y *Record*. Un *CS* puede ser *editable* (permite edición de ciertos contenidos) o no (es utilizado solamente para presentar contenido al usuario). Además tiene un atributo llamado *filterable* que significa que los elementos del *CS* pueden ser filtrados bajo alguna condición provista por el usuario o no; este atributo puede ser utilizado para listas, grids y trees. Para el caso de trees los filtros pueden considerar uno o más elementos básicos (*basicUiElements*) en los nodos. Además pueden ser *contributor* (sus elementos se pueden arrastrar y soltar) o no, y *receptive* (puede aceptar elementos arrastrados de otros) o no.

List: representa la presentación de una lista de elementos de contenido. Esto significa que una instancia de *List* contiene un conjunto de de instancias *BasicUiElement/CS* del mismo tipo. Un elemento *List* puede contener solo un *CS* –a través de *childCS*– o un *basicUiElement* –a través de *element*–.

Record: representa la presentación de un conjunto de, al menos dos, elementos de contenido. Esto significa que una instancia de *Record* contiene un conjunto de instancias de *BasicUiElement/CS*.

Grid: representa la presentación de varias filas de uno o más tipos. Esto significa que una instancia de *Grid* contiene un conjunto de instancias de *record* de uno o más tipos. Un *Grid* solo puede contener *records* –a través de *childCS*– para describir sus filas.

Table: representa la presentación de varias filas de un solo tipo. Esto significa que una instancia de *Table* contiene un conjunto de instancias de *record* de un único tipo. Una *Table* solo puede contener *records* –a través de *childCS*– para describir el tipo de sus filas.

Tree: representa la presentación de una estructura de tipo árbol, cuyos nodos incluyen contenido. Un árbol contiene nodos internos y nodos hojas. Cada tipo de nodo contiene un elemento de texto para el nombre del nodo y, opcionalmente, uno o más *basicUiElements*.

UIOutputStructure: representa un elemento de interfaz gráfica utilizado para presentar información al usuario. Contiene *BasicUiElements* con el atributo *typeOfEdition*=“no-editable”.

Message: representa una comunicación presentada al usuario a través de una notificación o un cuadro de diálogo.

Dialog: representa un cuadro de diálogo presentado al usuario incluyendo una solicitud de decisión (por ejemplo a través de botones “Aceptar”, “Cancelar”).

Notification: representa un mensaje de notificación de la ocurrencia del algún evento presentado al usuario. Incluye un atributo *type* del tipo *NotificationType*. Un *NotificationType* puede tener los siguientes valores: *Help* (notificación de ayuda), *Error* (notificación acerca de un resultado erróneo de una acción realizada), *Success* (notificación acerca de la ejecución exitosa de una acción), *Warning* (una notificación de advertencia al usuario), *Status* (estado de una tarea –por ejemplo, su progreso–), *External* (una notificación proveniente de una aplicación externa o servicio).

UIInputStructure: representa un elemento de interfaz de usuario utilizado para entradas del usuario. Una instancia de *UIInputStructure* contiene instancias de *BasicElement* con el atributo *typeOfEdition*=“input”.

Form: representa un formulario presentado para el ingreso de información por parte del usuario.

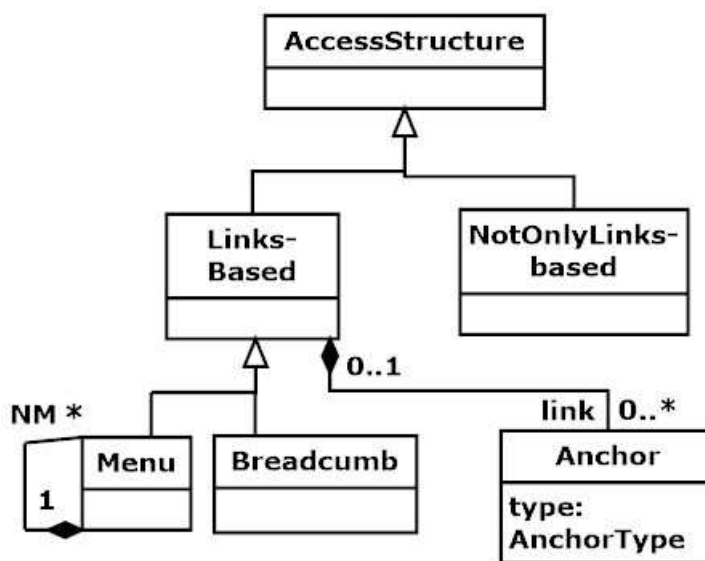


Figura 2.11: Meta-modelo para diseño abstracto de interfaces de usuario. Parte 2.

AccessStructure: representa un elemento de interfaz de usuario utilizado como una estructura de acceso a información. Pueden ser basadas en enlaces y no solo basadas en enlaces.

LinksBased: representa un agrupamiento de enlaces para acceder a otros elementos de interfaz de usuario o para realizar alguna acción. Algunas especializaciones de *LinksBased* son *Menu* y *Breadcumb*.

Menu: es un *LinksBased* que contiene dos o más áncoras. Un menú puede contener otros menú (es decir, menús anidados).

Breadcumb: contiene una lista de pasos, cada paso tiene un áncora. Los elementos *Breadcumb*s son usados para representar caminos de navegación cuyos nodos pueden ser visitados seleccionando pasos.

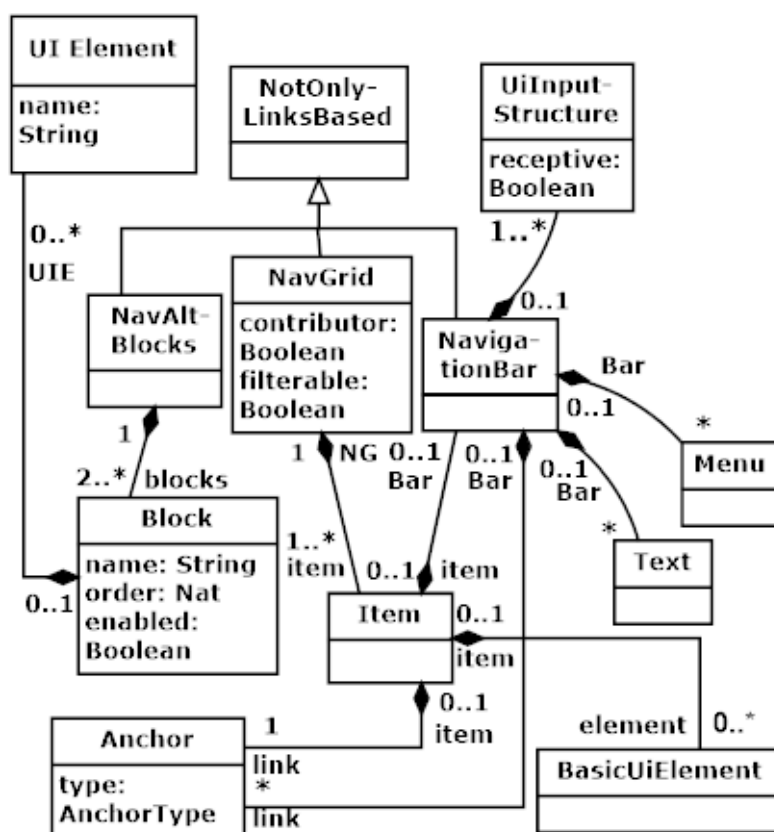


Figura 2.12: Meta-modelo para diseño abstracto de interfaces de usuario. Parte 3.

NotOnlyLinksBased: representa acceso a elementos de interfaz de usuario no basados exclusivamente en enlaces. Para algunas estructuras de tipo *NotOnlyLinksBased* el usuario puede tener que ingresar datos a través de estructuras de entrada –por ejemplo, formularios– para recolección de parámetros para generación de *UiStructure* o ejecución de funcionalidad. Se consideran 3 tipos de elementos *NotOnlyLinksBased*: *NavigationBar*, *NavGrid* y *NavAltBlocks*.

NavigationBar: representa una barra de navegación. Es decir un conjunto de áncoras, menús y elementos *UiInputStructure* (al menos debe haber un elemento de este tipo). Un *NavigationBar* puede contener más de un elemento del tipo *UiInputStructure*. Además, un *NavigationBar* puede contener algunos elementos de texto con el atributo `type="Not-Editable"`. Por lo general, una barra de navegación se comporta de la siguiente manera: después de la selección de un áncora o envío de formulario se muestra una estructura de acceso, un CS o un *UiInputStructure*; o se ejecuta un comando.

NavAltBlocks: representa un objeto que contiene una colección de bloques (elementos *Block*) en el cual solo un bloque puede estar visible a la vez. Cada elemento del tipo *Block* contiene cero o más *UiElements*, tiene un nombre y un orden. Además, un elemento del tipo *Block* un atributo del tipo *Boolean* con el nombre `enabled` que especifica si el bloque está habilitado para navegación o no. *NavAltBlocks* son una abstracción de navegadores Tab y navegadores del tipo Accordion, pero también pueden representar estructuras más genéricas. A lo sumo un bloque en un *NavAltBlocks* puede tener cero *UiElements*.

NavGrid (Navigation Grid): representa un elemento de interfaz de usuario que contiene un conjunto de ítems. Cada ítem respeta un esquema de ítems. En un *NavGrid* puede haber uno o más esquemas de ítems, y por cada esquema de ítems puede haber uno o más ítems. Cada esquema de ítems es especificado de la siguiente manera: un áncora correspondiente al ítem que será visualizado por un ítem, opcionalmente un elemento *navigationBar* para parámetros y/o acceso a funcionalidad y cero o más *BasicUIElements*. Tiene un atributo `filterable`, el cual hace referencia a si los ítems del *navGrid*

pueden ser filtrados bajo ciertas condiciones dadas por el usuario. Un caso particular de un *NavGrid* es una lista de navegación, donde existen distintos ítems respetando una única estructura de ítems. Algunos ejemplos de listas de navegación son: índices (solo *navigationBar* es excluido), inbox en Gmail (un *NavigationBar* que contiene un *UiInputStructure* solo con checkboxes, una áncora a un mail y algunos elementos básicos para resúmenes de conversaciones). Un *CS/NavGrid* puede ser un *contributor* (puede proveer elementos a un *CS/UiInputStructure*). Un *CS/UiInputStructure* puede ser *receptive* (puede recibir elementos de otros *CS/NavGrid*).

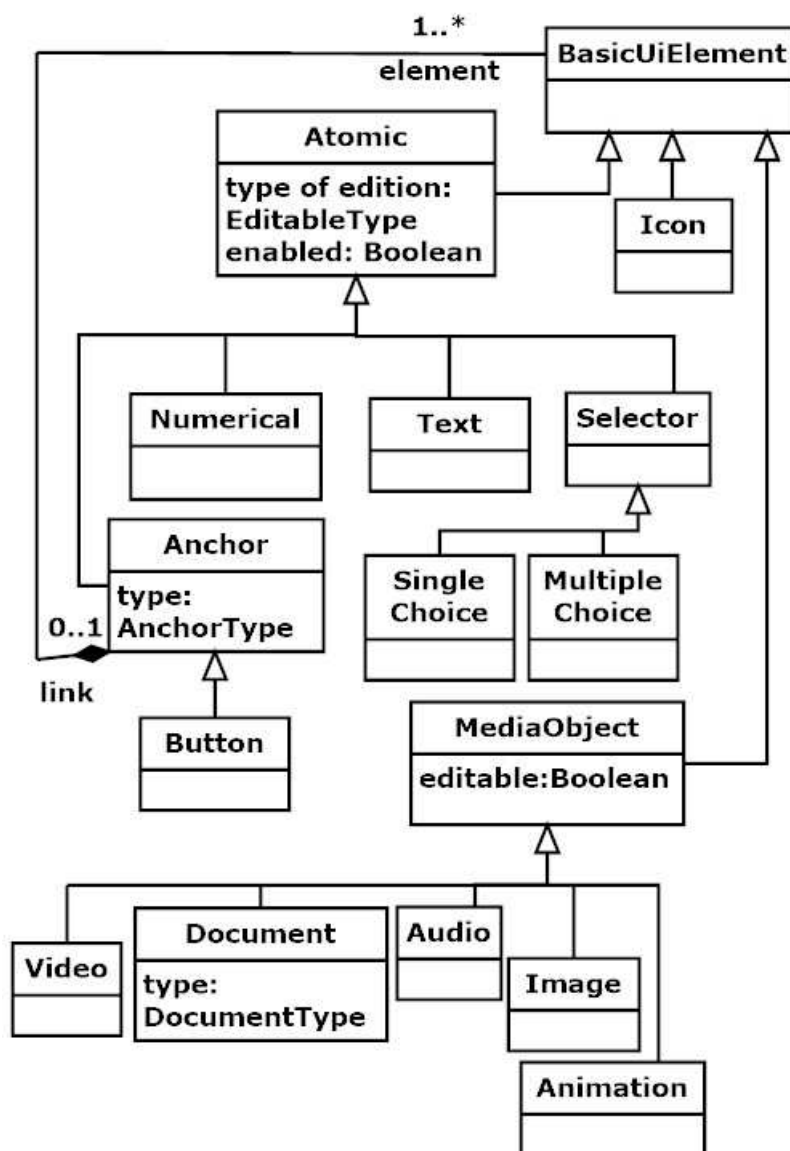


Figura 2.13: Meta-modelo para diseño abstracto de interfaces de usuario. Parte 4.

BasicUiElement: representa un elemento de interfaz de usuario básico de una interfaz, el cual existe solo o en grupos homogéneos. Pueden ser de tres tipos: atómicos, íconos o elementos de medios.

Atomic: engloba elementos básicos que pueden ser utilizados para diferentes propósitos de acuerdo a su atributo *type of edition*, el cual puede ser: *input* (para ingreso de información), *editable* (para edición de información), y *not editable* (para la presentación de información). Un elemento *Atomic* además tiene un llamado *enabled* que especifica si el elemento está habilitado o no para su interacción con

el usuario. Por ejemplo, para el caso de las áncoras este atributo dice si el enlace está habilitado o no para navegación.

Anchor: representa un elemento visible que puede ser seleccionado mediante un enlace. Contiene un atributo *type* del tipo *AnchorType*. Los valores posibles para un *AnchorType* son: *classicalLink* (enlace hacia otra página web o documento), *bookmarkLink* (enlace hacia otra posición en la misma página web) y *commandLink* (su selección inicia la realización de una acción o tarea). Un *Anchor* contiene uno o más *basicUiElements* de tipo diferente a *Anchor* y con *editableType*=“not-editable”.

Button: hereda de *Anchor* y representa un botón que se comporta como un *Anchor* cuando es presionado.

Numerical: representa un elemento numérico presentado al usuario ya sea para presentación, ingreso de números y para su edición, de acuerdo al tipo de edición que tenga.

Text: representa un elemento de texto presentado al usuario ya sea para presentación, ingreso de datos de texto y para su edición, de acuerdo al tipo de edición que tenga.

Selector: representa un elemento de interfaz de usuario para que el usuario seleccione una opción entre una o más opciones visibles.

Single Choice: representa un elemento de interfaz de usuario para que el usuario seleccione una opción o no. Un ejemplo de *Single Choice* puede ser un único elemento *CheckBox* o *Radio Button* de *HTML*.

Multiple Choice: representa un elemento de interfaz de usuario para que el usuario seleccione una opción entre varias opciones. Un ejemplo de *Multiple Choice* pueden ser los elementos *CheckBox* (más de un elemento) y *Select* (con varias opciones) de *HTML*.

Icon: es una representación gráfica de algo (una persona o un objeto) que es simbólica o una figura notable. Un *Icon* es un *BasicUiElement*. Ejemplos de *Icon* son: un *glyph* (es un símbolo, no una imagen), una imagen, y un ícono de aplicación (usualmente denominado *favicon*).

MediaObject: representa un objeto de medios, que puede ser: imagen, video, audio, animación o un documento. Un *MediaObject* puede ser editable o no. Para el caso que sea editable, esto significa que además del objeto de medios se incluyen los elementos necesarios para la edición.

Image: representa una imagen. Un ejemplo de este tipo de elementos es la etiqueta `` de *HTML*.

Video: representa un video. Un ejemplo de este tipo de elementos es la etiqueta `<video>` de *HTML 5*.

Audio: representa un audio de sonido. Un ejemplo de este tipo de elementos es la etiqueta `<audio>` de *HTML 5*.

Animation: representa una animación o presentación. Un ejemplo de este tipo de elementos es la etiqueta `<object>` de *HTML* incluyendo un objeto de animación.

Document: representa un documento, que puede ser de texto, una hoja de cálculo, entre otros tipos de documentos. Un ejemplo de este tipo de elementos es la etiqueta `<object>` de *HTML* incluyendo un documento.

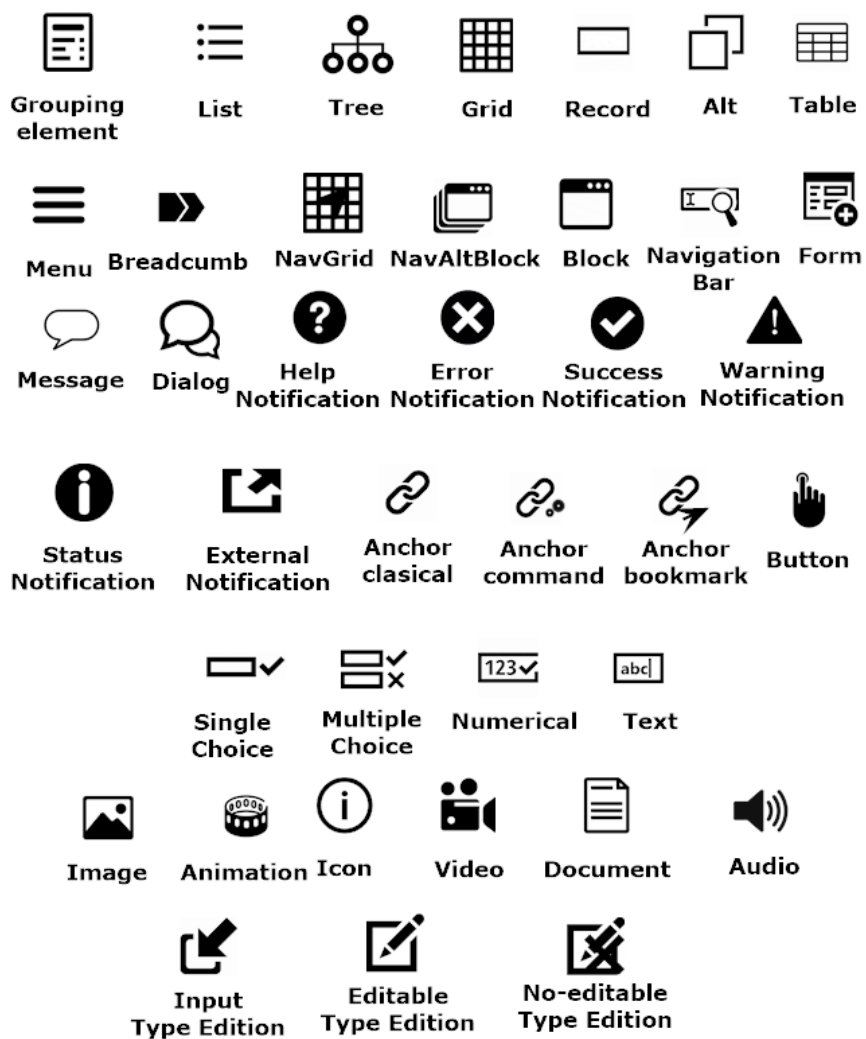


Figura 2.14: Sintaxis concreta para elementos del meta-modelo RIAAD.

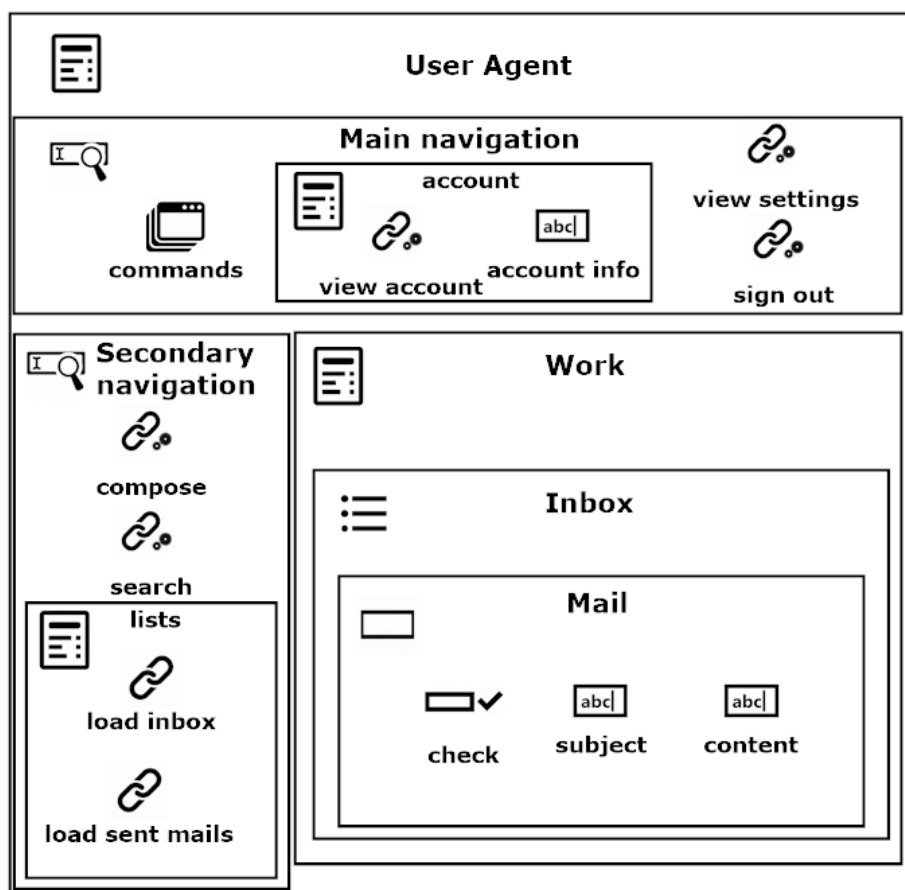


Figura 2.15: Ejemplo de modelo de interfaz de usuario abstracta para grouping *User Agent* de aplicación de correo electrónico online.

Ejemplo En la figura 2.15 se presenta el modelo de interfaz de usuario abstracta para una aplicación de correo electrónico online. El ejemplo se corresponde a la interfaz de usuario de la vista estática inicial del modelo estático de acceso y funcionalidades y contenido de la figura 2.6. El grouping y contenedor principal *User agent* representa la pantalla inicial de la aplicación y consta de tres elementos principales: dos elementos de tipo Navigation bar (*Main navigation* y *Secondary navigation*), que offician como principales navegaciones de la aplicación, y un elemento grouping *Work*, que representa el área de trabajo y visualización de contenido principal de la aplicación. *Main navigation* se compone de un elemento de tipo NavAltBlock llamado *commands* (con los comandos que el usuario puede realizar), dos elementos de tipo Command link (*view settings* y *sign out*, representando operaciones a nivel de aplicación) y un elemento grouping *account* (información y acceso a modificar información de usuario). *Secondary navigation* consta de dos elementos de tipo Command link (*compose mail* y *search*, operaciones sobre contenido) y un elemento grouping llamado *lists* (con enlaces de acceso a distintas listas de mails). El elemento *Work* inicialmente muestra un elemento List (*Inbox*), que representa el listado de correos electrónicos recibidos (elemento de tipo Record *Mail*).

Notación para definición de eventos sobre elementos de interfaz gráfica

En esta sección presentamos la definición de una notación de eventos aplicables a elementos de la notación *WAAUID*.

Un *evento atómico* (*AEP*, por sus siglas en inglés) consta de un *nombre*, un *origen* (referencia a una instancia de elemento de *WAAUID*), y sus *datos*. Se supone que en un momento dado de la ejecución de una aplicación web existe una secuencia de eventos atómicos que sucedieron y, además, que para cada evento atómico en la secuencia hay una marca de tiempo de su ocurrencia. La figura 2.16 muestra la clasificación considerada de eventos atómicos en este trabajo. Por cada tipo de evento se listan los

eventos contenidos en ese tipo (segunda columna) y los elementos de *WAAUID* que actúan como orígenes de tales eventos(tercera columna).

Un *patrón de eventos (EP, por sus siglas en inglés)* representa un conjunto de eventos respetando un patrón y puede ser usado para consultar la secuencia de eventos atómicos sucedidos (una lista de eventos que pertenecen a la secuencia de eventos puede respetar o no un EP). A continuación se provee una gramática BNF para patrones de eventos. Un *patrón de evento atómico* consta de: un nombre de evento, *origen* y, opcionalmente, *información* del evento. Un *patrón de evento compuesto (CEP, por sus siglas en inglés)* se describe por medio de operadores de composición (dados por medio de los no-terminales <OP> y <REP>) y eventos atómicos participantes. Los operadores de composición fueron tomadas de [Mbaki, 2008] y [Bry, 2006]. <EP> ::= <CEP> | <AEP> ,

```
<CEP> ::= <EP><OP><EP> | <REP><N><EP> ,
<OP> ::= AND | OR | SEQ , <N> ::= <nat>times,
<REP> ::= exactly | at least | at most,
<AEP> ::= <NAME>on <SOURCE>(with <DATA>)?
<NAME> ::= <str>, <SOURCE> ::= <str>, <DATA> ::= <str>
<str> es utilizado para strings alfanuméricos y <nat> para números naturales.
```

Event Type	Event name	Source of the event
User interface events	Press	Anchor, Form (submission)
	Over, moveOut	BasicUiElement, elements of CS.
	enter, edited	CS (element edition), BasicUiElement
	select	LinksBased, Selector, Selector in Form, Block in NavAltBlocks.
	remove, add, move	CS (modification)
	select to share	contributor CS/NavList
	Put	receptive CS/UiInputStructure
Data events	update, delete, insert,	relations, XML document.
Transaction	commit, abort, request	Transaction
Timer events	start, timeout	Timer
Service events	start, finish	web service
Developer defined	Given by the developer	The actual web application or an external application.

Figura 2.16: Clasificación de eventos atómicos.

Ejemplo Si tomamos como referencia la definición de interfaz de usuario abstracta para el grouping *User Agent* de aplicación de correo electrónico online de la figura 2.15, a manera de ejemplo se pueden definir los siguientes eventos:

Press on «anchor» load sent mails

Press on «anchor» load inbox

2.2.3. Taxonomía para describir actividades autónomas del sistema en diagramas de actividad

Como ya hemos presentado en la sección 2.1.1 (*Taxonomía para describir acciones de diagramas de actividad de Aplicaciones Web*), en este trabajo utilizamos distintos tipos de acciones para modelar la interacción del usuario con la aplicación web. En particular, las acciones de tipo «job» representan una *actividad autónoma* realizada por el sistema. Ya que en las aplicaciones web son muy variadas las actividades autónomas que el sistema puede realizar, si se quiere contar con más detalle sobre las mismas

con el objetivo de proveer al desarrollador más información, para así facilitar su tarea de implementación o para poder aplicar transformación de modelos de una manera más precisa, es útil contar con una notación que permita refinar tales actividades autónomas del sistema de una manera más concreta. Con este fin, en esta sección presentamos un perfil de estereotipos de acciones de diagramas de actividad de *UML* que puede ser utilizado para refinar algunos tipos de acciones de los diagramas de actividad que describen los casos de uso de requisitos del sistema. Además, hemos ejemplificado algunas situaciones típicas de modelado de actividades autónomas del sistema.

Para describir acciones de requisitos «*job*» con actividades, consideramos los siguientes estereotipos:

- «*insert*»: representa la inserción de elementos de datos, relaciones entre elementos de datos o de una vista de una base de datos (por ejemplo una base de datos relacional o un documento *XML*)
- «*delete*»: representa el borrado de elementos de datos, relaciones entre elementos de datos o de una vista de una base de datos (por ejemplo una base de datos relacional o un documento *XML*)
- «*update*»: representa la modificación o actualización de elementos de datos, relaciones entre elementos de datos o de una vista de una base de datos (por ejemplo una base de datos relacional o un documento *XML*)
- «*insertUI*»: representa la inserción de elementos en una estructura de contenido de interfaz de usuario.
- «*deleteUI*»: representa el borrado de elementos en una estructura de contenido de interfaz de usuario.
- «*updateUI*»: representa la actualización automática, por la aplicación, de información de una estructura de contenido de interfaz de usuario, un *UIBasicElement* o un *UIBasicElement* dentro de un *UIInputStructure*.
- «*changeUiIS*»: representa la alteración de una estructura de ingreso de datos (*UiInputStructure*) hecha automáticamente por la aplicación. Esto significa que se agregan o eliminan algunos elementos y el resto permanece como estaban.
- «*dataRetrieve*»: representa la obtención de información a través de alguna fuente dada, que puede ser un archivo de texto, una base de datos relacional, un documento *XML*, un servicio REST o una vista de una base de datos.
- «*getUIContent*»: representa la obtención de contenido desde uno o varios elementos de interfaz de usuario.
- «*UIPropertyChange*»: representa la alteración propiedades de elementos de interfaz de usuario (por ejemplo *accessible* –hace referencia a si el elemento es visible, audible u otro tipo; dependiendo de la modalidad–, *enabled* – el elemento se encuentra habilitado para interacción con los usuarios de la aplicación–, y *focus* –el elemento está en foco–). Este estereotipo incluye un valor etiquetado *property* que incluye el nombre de la propiedad como valor y otro valor etiquetado *value*, que incluye el valor de la propiedad a la cual se hace referencia como valor.
- «*validation*»: representa la validación de información o regla de negocios. Puede ser en la parte del cliente (validación de contenido de elementos de interfaz de usuario), en la parte del servidor o en un servicio web.
- «*dataProcess*»: representa el procesamiento de información. Algunos ejemplos son el procesamiento de información proveniente de base de datos o de un elemento de interfaz de usuario.
- «*generateEvent*»: representa la generación de un evento atómico sobre un elemento de interfaz de usuario. Incluye un valor etiquetado *expr*, con una expresión de *EP atómico* como valor.
- «*externalJob*»: representa la invocación de alguna tarea externa a la aplicación. Puede ser la invocación de un servicio web, el tratamiento de un evento generado por otra aplicación o la recolección de información no perteneciente a la aplicación (por ejemplo mashups).
- «*externalNotification*»: representa una notificación a la aplicación proveniente de una tarea externa. Por ejemplo puede ser un mensaje de notificación de un servicio web o la notificación de una aplicación externa de que un evento fue procesado.

Consideraciones de modelado. Para definir con detalle las actividades autónomas del sistema con la taxonomía presentada en esta sección se deben examinar las acciones con estereotipo «job» de las descripciones de casos de uso. Cada acción estereotipada con «job» está asociada a un diagrama de actividad describiendo la acción autónoma que se lleva a cabo, para esto se utiliza el perfil definido en esta sección y no debería representar complejidad llevar a cabo esta tarea si se toman en cuenta como referencias las definiciones de cada elemento del perfil y el nombre de la acción estereotipada con «job».

Ejemplos de uso de la taxonomía

A continuación se brindan algunos ejemplos de refinamientos de acciones de tipo «job» frecuentemente utilizadas y expresadas de manera genérica:

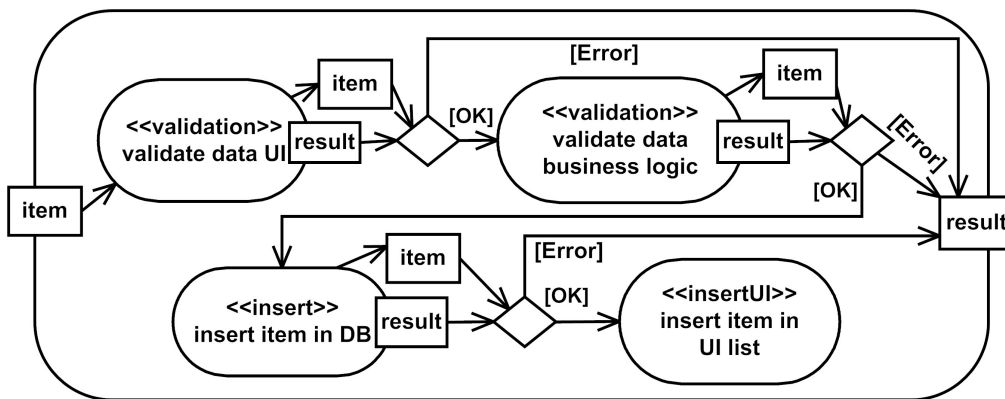


Figura 2.17: Ejemplo para refinamiento de «job» genérico *validate data and add item job*.

En la figura 2.17 se presenta la actividad que es el refinamiento de la acción de tipo «job» *validate data and add item job*, que representa la acción de validar información y agregar un ítem (de algún tipo de información) al sistema. Inicialmente, la información acerca del ítem es validada, luego se chequea si se violan ciertas reglas de negocios (utilizando la acción estereotipada con «validate» *validate data business logic*); luego, el ítem es insertado en la base de datos de la aplicación y, finalmente, se actualiza la presentación de la lista de ítems para reflejar la inserción del ítem (utilizando la acción estereotipada con «insertUI» *insert item in UI list*).

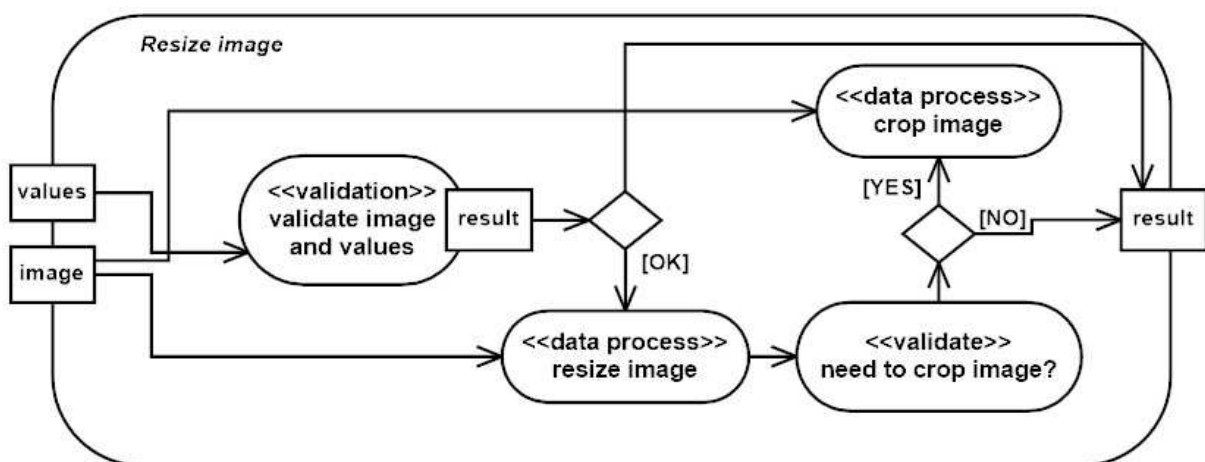


Figura 2.18: Ejemplo para refinamiento de «job» genérico *Resize image*.

La figura 2.18 muestra otro ejemplo de refinamiento de acción «job», en este caso para la acción *Resize image*, que hace referencia a una operación muy frecuente en las aplicaciones que incluyen algún

tipo de manejo de imágenes y que se encarga de modificar, en base a nuevos valores, las dimensiones de una imagen y, de ser necesario, además se delimita la imagen para que la redimensión no distorsione el aspecto visual de la imagen. Inicialmente se validan los valores ingresados para redimensionar la imagen. Si los valores son correctos, mediante la acción *resize image* estereotipada con «data process» se realiza la redimensión; a continuación, el sistema chequea si es necesario delimitar la imagen, en caso de serlo se realiza la delimitación mediante la acción *crop image* estereotipada con «data process».

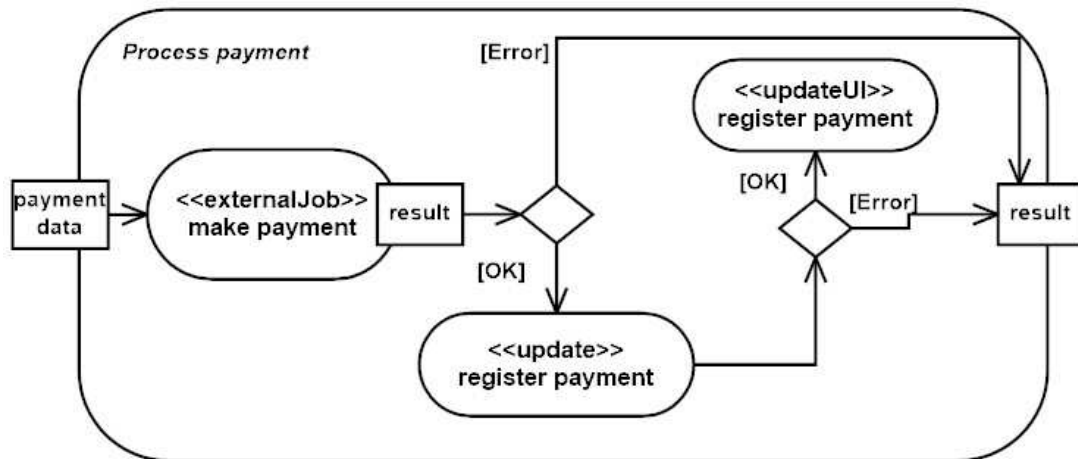


Figura 2.19: Ejemplo para refinamiento de «job» genérico *Process payment*.

En la figura 2.19 se incluye el refinamiento de la acción «job» *Process payment*, que es una operación comúnmente utilizada en sitios de comercio electrónico para procesar el pago de una compra. Inicialmente, mediante una operación externa al sistema (puede ser a través de un banco o de un sistema de pagos) se efectúa el pago tomando como entrada los valores del pago, si el resultado del pago es exitoso el sistema actualiza en base de datos la información de la compra (mediante la acción *register payment* estereotipada con «update»). Si este registro fue exitoso, el sistema modifica la información de la compra en pantalla (mediante la acción *register payment* estereotipada con «updateUI»). Si hay error, se retorna información del mismo.

Capítulo 3

Notaciones para modelado de dominio

3.1. Diagramas de Casos de Uso con variabilidades

3.1.1. Background

En [Bragança, 2007] se define una notación de casos de uso de UML que permite modelar variabilidades (lo cual la hace apta para modelar dominio). Para dicho fin, incluye los elementos *Variability*, *IncludeVariability* y *ExtendVariability*. El elemento *Variability* representa un conjunto de variantes (elementos que pueden estar presentes o no en una aplicación dada) y tiene los siguientes atributos: *name*, *min* y *max* (estos valores son números naturales que representan las cardinalidades mínima y máxima que pueden tener el conjunto de variantes elegidos). El elemento *Variability* es representado gráficamente con una nota adjuntada a sus variantes por medio de flechas. El elemento *IncludeVariability* es utilizado para especificar un conjunto de variantes que son dependencias del tipo «include» de casos de uso. El elemento *ExtendVariability* es utilizado para especificar un conjunto de variantes que son dependencias de tipo «extend» de casos de uso, las cuales tienen el mismo caso de uso base.

Además del trabajo de [Bragança, 2007], se han hallado otros trabajos en la literatura que permiten modelar variabilidades en diagramas de casos de uso de UML. En general, se han identificado los siguientes problemas en relación a diagramas de casos de uso de UML que permiten modelar variabilidades:

- **Problema 1:** En algunas notaciones (tales como [Bragança, 2007] y [Bühne, 2003]), se introducen algunos casos de uso “extra” no interesantes, solo por propósitos de representación de variabilidad; y tales tipos de casos de uso pueden proliferar en una línea de productos grande. Por ejemplo, en [Bragança, 2007], para el caso en el que se tienen casos de uso U_1, \dots, U_n y se quiere elegir un subconjunto de estos casos de uso con una cardinalidad máxima m y una cardinalidad mínima m' se crea un caso de uso U y se define un elemento *InclusionVariability* cuyos variantes son relaciones de tipo «include» entre U y los casos de uso U_1, \dots, U_n ; en otras palabras: U actúa como un “menú de selección” y tales casos de uso no son interesantes desde la perspectiva de diagramas de casos de uso. Para resolver este problema es necesario modelar variabilidades cuyos variantes se correspondan con casos de uso de manera que no se introduzcan casos de uso no interesantes.
- **Problema 2:** Los enfoques hallados en la literatura no brindan consideraciones explícitas para la definición de variabilidades cuyos variantes representen subsistemas o sub-subsistemas enteros involucrando casos de uso. Para resolver este problema es necesario especificar con qué elementos y de qué manera se puede lograr esto.
- **Problema 3:** En [Bühne, 2003] se modelan restricciones de dependencias en diagramas de casos de uso y se consideran restricciones de dependencias entre casos de uso variantes. Para ser aún más expresivo se deben definir restricciones de dependencias que tengan al menos un variante que represente un subsistema o sub-subsistema entero; esto es necesario ya que se encontraron de estos tipos de restricciones de dependencia en el caso de estudio desarrollado en este trabajo

correspondiente a un Sistema de Administración de Bibliotecas Online. Para resolver este problema es necesario resolver el Problema 2 y agregar nuevas consideraciones para definir restricciones de dependencia.

- **Problema 4:** Cómo definir un punto de variación cuyos variantes representan casos de uso base que son extendidos por el mismo de caso extensión; esto es necesario ya que se encontró un ejemplo de este tipo de variabilidad en nuestro caso de estudio de sistema de Administración de Bibliotecas Online.

En base a los problemas mencionados, el objetivo de esta sección, es definir una notación para modelar casos de uso de UML con variabilidades. Como metas para alcanzar este objetivo se pretende resolver los problemas 1, 2, 3 y 4; descritos anteriormente. Este objetivo contribuye a lograr la meta número 2 de la sección 1.4 del capítulo de Introducción.

En esta sección se realizaron extensiones al trabajo presentado en [Bragança, 2007] para modelar variabilidades en casos de uso de UML. Estas extensiones proveen la resolución a los problemas mencionados anteriormente.

3.1.2. Trabajo relacionado

La tabla 3.1 provee una comparación de los enfoques más relevantes que utilizan notaciones de diagramas de casos de uso con variabilidades que respetan los requisitos R1 hasta R6 y sus resultados son explicados en detalle a continuación de la tabla 3.1. Se decidió excluir a los enfoques de la literatura que no satisfacen al menos uno de estos requisitos.

	R1	R2	R3	R4	R5	R6
[Bragança, 2007]	good	good	no	no	no	no
[Bühne, 2003]	good	reg	no	no	no	reg
[Gomaa, 2004]	no	no	good	no	no	no
[Azevedo, 2012]	no	no	very good	reg	no	no
Our approach	good	very good	very good	very good	good	good

Figura 3.1: Comparación de notaciones de diagramas de casos de uso con variabilidades.

- **R1:** *Expresar cómo los variantes son seleccionados con flexibilidad y simplicidad.* [Bragança, 2007], [Bühne, 2003] y este trabajo satisfacen este requisito.
- **R2:** *No hacer uso de elementos de diagramas de casos de uso para proveer información de variabilidad.* (Puede ser nombre de un punto de variación o información acerca de cómo se seleccionan los variantes). En [Bühne, 2003] se usa el estereotipo «variant» para casos de uso. [Bragança, 2007] ha sido calificado como bueno (good) ya que no resuelve el problema 1.
- **R3:** *Evitar el Problema 1 (Sección 3.1.1);* idealmente sin pérdida de información de variabilidad. En [Bühne, 2003] no se evita el Problema 1, ya que para definir variabilidades se utiliza un caso de uso con una relación hacia un punto de variación asociado a los casos de uso variantes. En [Gomaa, 2004] se evita el Problema 1, pero no se tiene en cuenta la información sobre los variantes que pertenecen a cada variabilidad (el lector tiene que usar su sentido común). En [Azevedo, 2012] y este trabajo se satisface este requisito de manera óptima.
- **R4:** *Consideración de variabilidad con variantes representando subsistemas,* idealmente resolver el Problema 2 (Sección 3.1.1) respetando R3. Solamente en [Azevedo, 2012] y en este trabajo se satisface este requisito. En [Azevedo, 2012] se utilizan paquetes de casos de uso relacionados con una relación alternativa, pero en este trabajo se provee más flexibilidad para expresar cómo

los variantes que representan subsistemas (a través de paquetes de casos de uso variantes) son seleccionados.

- **R5: Resolver el Problema 4 (Sección 3.1.1).** Sólo en este trabajo.
- **R6: Consideración de restricciones de dependencia en diagramas de casos de uso;** idealmente incluyendo diferentes tipos de variantes y resolviendo el Problema 3. Solo en [Bühne, 2003] y en este trabajo se consideran restricciones de dependencia en diagramas de casos de uso. En [Bühne, 2003] se consideran restricciones de dependencia entra variantes de casos de uso (de diferentes tipos). En este trabajo se satisface este requisito de manera óptima.

3.1.3. Notación propuesta para Diagramas de Casos de Uso con variabilidades

En este trabajo, para modelar requisitos funcionales, se utilizan diagramas de casos de uso de [OMG, 2009]. Para diagramas casos de uso de dominio se realizó una adaptación al trabajo presentado en [Bragança, 2007] para modelar casos de uso de UML con variabilidades.

Elementos para describir variabilidad. En nuestro trabajo se toma el concepto de anotaciones utilizado en [Bragança, 2007] para definir variabilidades, pero se realizaron una serie de agregados que brindan la posibilidad de definir más tipos de variabilidades que se consideran útiles y necesarias.

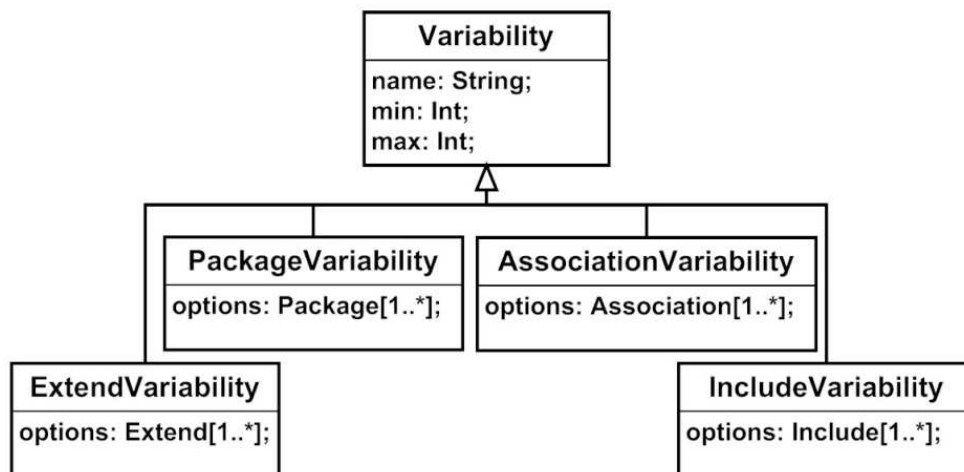


Figura 3.2: Meta-modelo para variabilidades en Diagramas de Casos de Uso.

Para el meta-modelo de anotaciones de variabilidad, se realizaron tanto incorporaciones como modificaciones al trabajo de [Bragança, 2007]. Se incorporaron los elementos *AssociationVariability* y *PackageVariability* con el fin de resolver los problemas 1 y 2, respectivamente. Se considera una definición más general del elemento *ExtendVariability* con el fin de resolver el problema 4 y se definen nuevos tipos de restricciones de dependencias entre variantes para resolver el problemas 3. Por lo tanto, se define:

- *ExtendVariability*: es utilizado para especificar variabilidades cuyos variantes son dependencias «extend» listadas en el atributo *options* de la meta-clase. No es necesario que todas las dependencias tengan el mismo caso de uso base. Para resolver el problema 4 se requiere que todas las dependencias tengan el mismo caso de uso extensión y diferentes casos de uso base.
- *PackageVariability*: es utilizado para especificar variabilidades cuyos variantes son paquetes de casos de uso listados en el atributo *options* de la meta-clase. Un paquete de caso de usos variante puede representar todo un sistema o subsistema variante, el cual puede ser seleccionado o no para un producto específico.

- *AssociationVariability*: es utilizado para especificar variabilidades cuyos variantes son asociaciones entre un caso de uso y un actor listadas en el atributo *options* de la meta-clase. Si se tienen casos de uso U_1, \dots, U_n asociados a un actor A y para los cuales se requiere seleccionar un subconjunto de estos casos de uso a través de una cardinalidad máxima m y una cardinalidad mínima n es necesario crear un elemento del tipo *AssociationVariability*, cuyos variantes son las asociaciones entre los casos de uso U_1, \dots, U_n y el actor A . Esto evita introducir casos de uso del estilo “selección de menú”, lo cual representa una solución al problema 1.

Restricciones de dependencia. Para resolver el problema 3 fue necesario definir restricciones de dependencia entre diferentes tipos de variantes en nuestra notación de diagramas de casos de uso con variabilidades. En la Figura 3.3 se representan las meta-clases para restricciones de dependencia entre variantes de casos de uso. Un elemento del tipo *Variant* consiste de un nombre de variabilidad y de un variante del diagrama de casos de uso (puede ser un elemento del tipo *Association*, un elemento *Package* o un elemento *Include*). Una restricción de dependencia puede ser del tipo *excludes* o del tipo *requires* y debe tener un nombre representativo. Cada restricción de dependencia tiene un elemento *dependent* (el dependiente) que puede ser un elemento del tipo *Association* o un elemento del tipo *Include* y que depende de un conjunto no vacío de variantes dentro del diagrama de casos de uso (obtenidos mediante la navegación de los roles *in* y *dependence on* de las meta-asociaciones correspondientes).

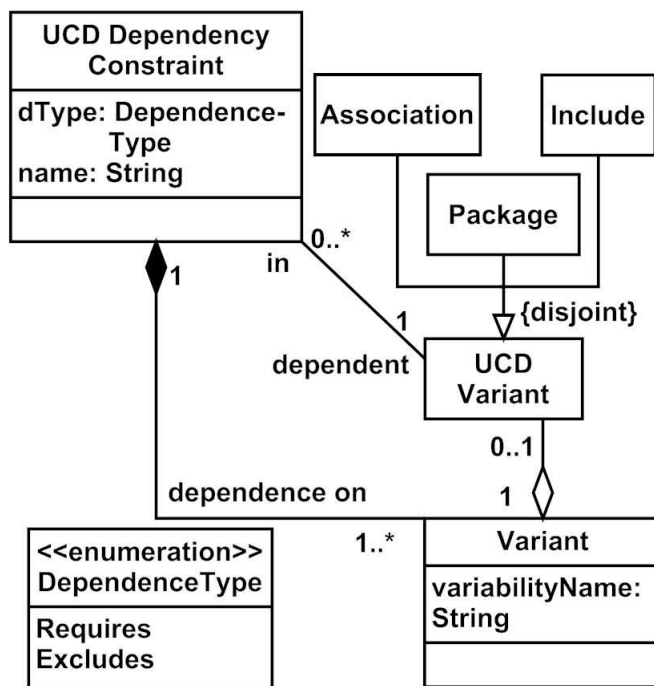


Figura 3.3: Meta-modelo para variabilidades en Modelos de Casos de Uso.

Ejemplos. La figura 3.4 muestra parte del paquete de casos de uso OPAC del sistema de Administración de Bibliotecas Online. Dentro del paquete OPAC hay varios elementos del tipo *PackageVariability*, por ejemplo: *PublicSearch?* (los variantes son paquetes: *Public Search* –cualquier usuario puede realizar búsquedas en OPAC– y *Private Search* –solo miembros de la biblioteca pueden realizar búsquedas en OPAC–) y *UserLogin* (incluye un paquete variante *MyAccount* –un usuario puede inspeccionar y alterar la información acerca de su cuenta y su actividad–, descrito en la Figura 3.5, opcional.)

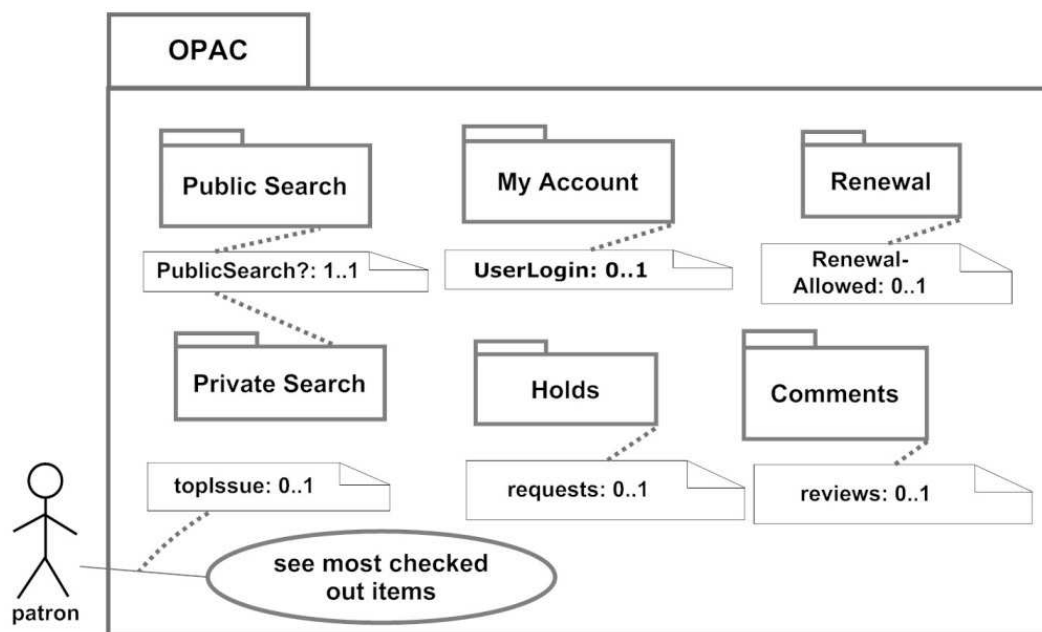


Figura 3.4: Parte del paquete de casos de uso OPAC del caso de estudio de Administración de Bibliotecas Online.

La figura 3.5 incluye parte del paquete My Account del caso de estudio de Administración de Bibliotecas Online. Entre los aspectos que se pueden destacar de este paquete, se incluye el paquete opcional Fines (esto representa otro nivel de paquetes a nivel opcional y justifica aún más por qué utilizar este tipo de variabilidades) y la restricción de dependencia en la asociación del tipo «include» entre el caso de uso *show account summary* y el caso de uso *show fines summary*.

La figura 3.6 muestra un ejemplo de variabilidad del tipo ExtendVariability, llamada *CreateIndexAccess* (min = 1, max = 1), cuyos variantes son relaciones «extend» entre el caso de uso extensión *add item* (para crear un ítem de la Biblioteca) y los casos de uso base *add record* (para catalogar un registro), *receive order* y *close basket*.

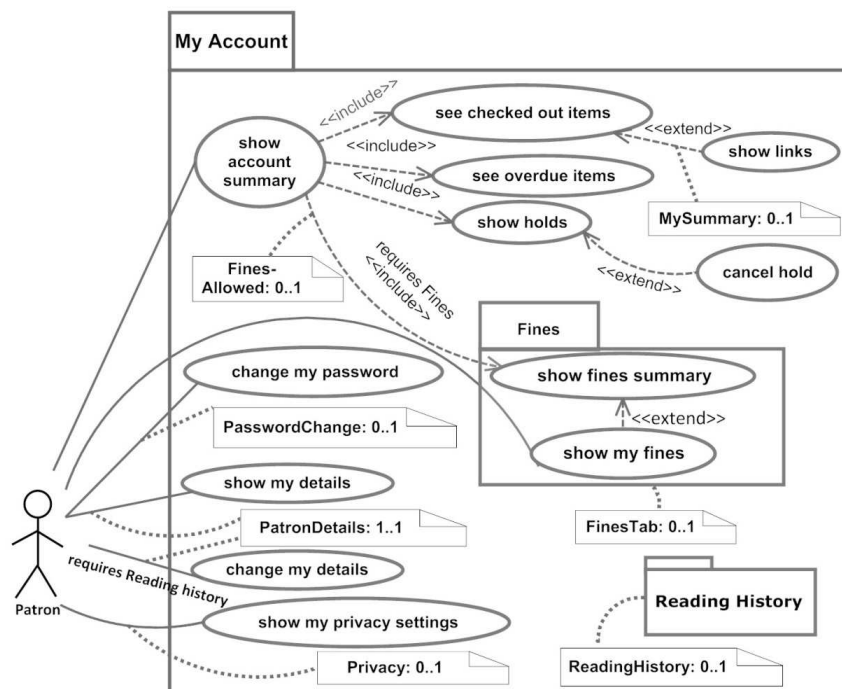


Figura 3.5: Parte del paquete de casos de uso My Account, incluido dentro del paquete de casos de uso OPAC.

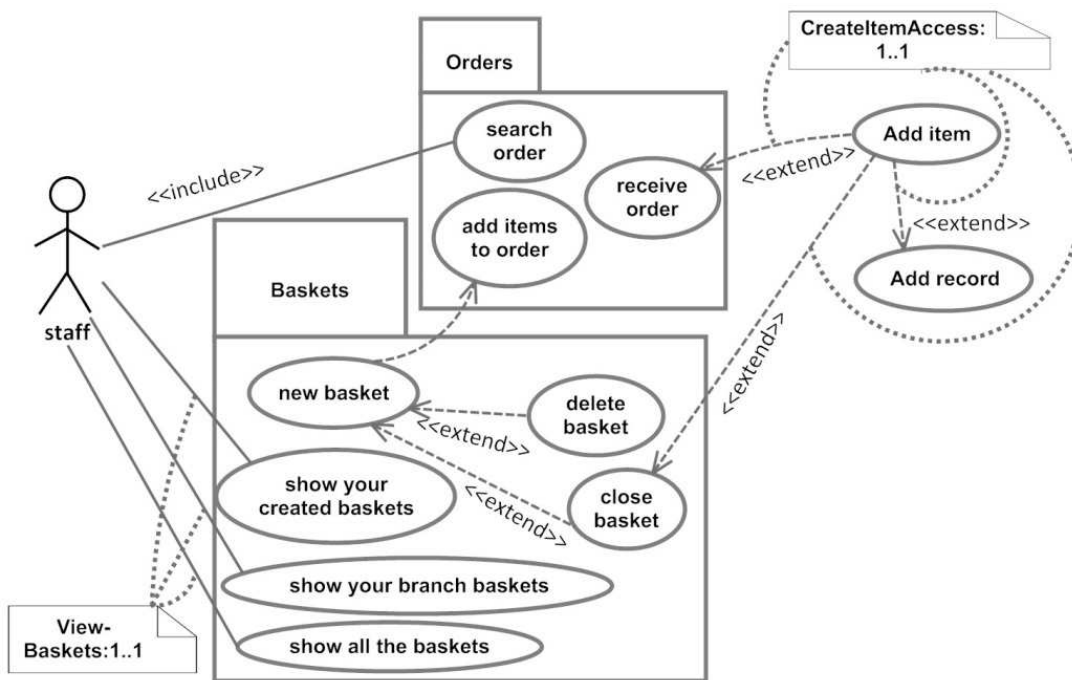


Figura 3.6: *CreateIndexAccess* ExtendVariability. Parte de paquetes de casos de uso Orders y Baskets.

3.1.4. Evaluación

Para analizar la propuesta de diagramas de casos de uso con variabilidades se utilizó el enfoque <problema, pregunta, resultado basado en una métrica>. Problema es alguno de los problemas de la sección 3.1.1.

Pregunta para el Problema 1: *¿Es demasiado alto el número de variabilidades que en otros enfoques introducen casos de uso no interesantes para modelar tales variabilidades?* Se encontró que un 62 % de variabilidades (que no son variabilidades paquetes) satisfacen esta propiedad.

Preguntas para el Problema 2: *¿Son bastante utilizados los variantes para paquetes de casos de uso?* Se encontró el 43 % de los paquetes de casos de uso son paquetes de casos de uso variantes. *¿La cantidad de variabilidades de paquete no alternativas es alta?* Se encontró que el 90 % de variabilidades de paquetes describen selecciones no alternativas de variantes (con min distinto de uno o max distinto de 1).

Además, se encontró un ejemplo usando la solución al Problema 4 y dos ejemplos usando la solución al Problema 3.

3.2. Diagramas de Actividad con variabilidades

3.2.1. Background

Para describir de manera detallada casos de uso de dominio, se utilizan *diagramas de actividad de UML* (ver [OMG, 2009]), con distintos agregados y extensiones que son necesarios para agregar variabilidad en los mismos.

Los diagramas de actividad de por sí, ya son útiles para describir variabilidades, ya que poseen elementos tales como nodos de decisión, nodos merge, nodos fork y nodos join; que los convierten en diagramas capaces de describir muchos tipos de variabilidades detectados (ver trabajo de [Korherr, 2007]). Si bien esta opción es válida para modelar variabilidades utilizando una notación conocida y sin ningún tipo de alteraciones, el uso de estos elementos propios de los diagramas de actividad para modelar variabilidad no basta para modelar otros tipos de variabilidades más complejas. Es por eso que en este trabajo se decidió utilizar, al igual que para casos de uso, el concepto de anotaciones de variabilidad adjuntadas a los elementos variantes de los diagramas de actividad.

Se han hallado algunos trabajos en la bibliografía que modelan variabilidades en diagramas de actividad. Sin embargo, se han identificado los siguientes problemas que no fueron considerados o no fueron resueltos apropiadamente en las notaciones existentes de diagramas de actividad con variabilidades:

- **Problema 5:** Existen situaciones en las cuales un variante de flujo de control que representa un comportamiento no puede ser modelado ni con una acción del tipo call behavior ni con un caso de uso inclusión.

Por ejemplo, en la Figura 3.7 se describe un comportamiento opcional llamado *Look for reservation* (chequear si un ítem está reservado), el cual es un variante de flujo de control dentro del caso de uso *reserve item* (reservar un ítem). Cuando un ítem no se encuentra reservado (en el diagrama la guarda *item not reserved*) es necesario continuar con la ejecución del caso de uso *reserve item*; caso contrario, es necesario terminar la ejecución de este caso de uso.

Este comportamiento no puede ser modelado ni con una acción del tipo call behavior ni con un caso de uso inclusión porque no se comporta como un llamado a una subrutina.

No se encontró en la literatura una caracterización matemática de tal tipo de variantes y una identificación de que este es un problema a resolver; tal caracterización debe ser utilizada para identificar y construir variantes de este tipo.

Si un cliente debe elegir este tipo de variantes, entonces necesitan ser mapeados a features críticas del modelo de features de la línea de productos que no pueden ser obtenidas a través de diagramas de casos de uso; en otras palabras, si la línea de productos tiene variantes de este tipo, no alcanza con transformar diagramas de casos de uso a modelo de features, es necesario mapear al menos un diagrama de actividad a features.

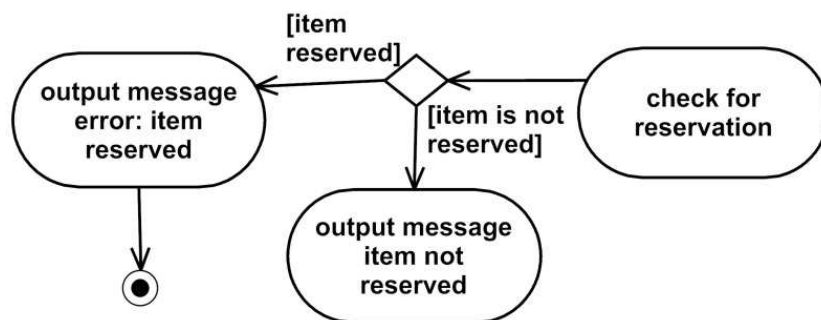


Figura 3.7: Comportamiento en Look for reservation.

- Problema 6:** Existen situaciones en las cuales un variante de flujo de control que representa un comportamiento no puede ser modelado ni con una acción del tipo call behavior ni con un caso de uso inclusión, y este variante necesita ser reutilizado por varios casos de uso.

En la Figura 3.7, se describe el comportamiento opcional *Look for reservation* que necesita ser reutilizados por los casos de uso *lend item*, *relend item* y *reserve item* del Sistema de Administración de Bibliotecas Online.

Debido a que los mecanismos de reutilización tradicionales (inclusión y acciones call behavior action) no pueden ser usados para este caso, es necesario definir un nuevo tipo de mecanismo de reutilización para permitir el reuso de este tipo de variantes en varios casos de uso.

- Problema 7:** Existen casos donde el mismo tipo de comportamiento es descrito varias veces con solo alguna variación mínima, dependiendo del contexto o el lugar donde aparece la descripción del comportamiento (ejemplos de contexto son: paquetes de casos de uso, casos de uso y definición de comportamientos reutilizables).

Por ejemplo, en la Figura 3.8 se describe un comportamiento para calcular fecha de vencimiento (*obtain due date*) que es usado por el caso de uso *renewing* (para renovar el préstamo de un ítem). En esta figura se observa una variabilidad llamada *SpecifyDueDate* con cardinalidad 1..1, cuyos variantes son dos elementos activity group: *enter due date* y *due date calculation*. El comportamiento *due date calculation* es utilizado para el caso de uso *checkout*, pero sin la parte inicial (la variabilidad *InitialDueDateCalculation* es removida).

Para este tipo de situación no se encontró una técnica de reutilización en la literatura, y es necesario proponer una.

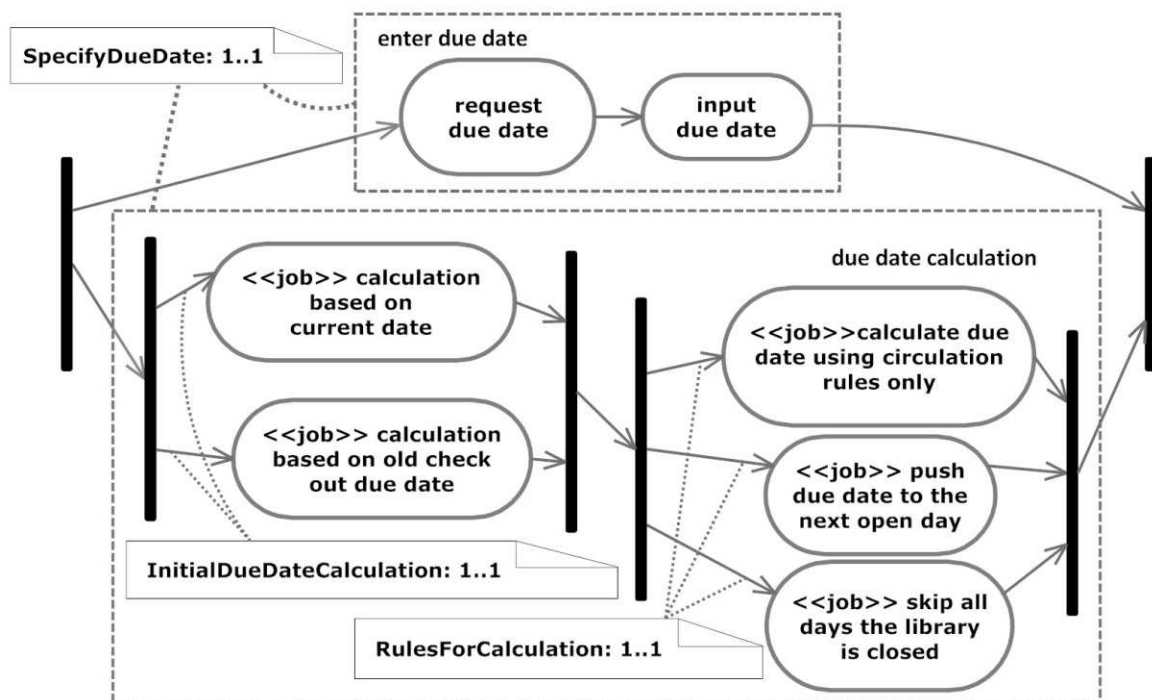


Figura 3.8: Comportamiento Obtain Due Date que es usado por el caso de uso renewing.

- **Problema 8:** el enfoque de [Schneider, 2007] considera mecanismo de parametrización de variabilidad en diagramas de actividad, donde cada miembro del conjunto de parámetros fue representado con elementos de modelado. Esta solución no es adecuada cuando la cantidad de elementos del conjunto de parámetros es elevada (se requiere un espacio físico excesivo), o el conjunto de parámetros es infinito; en el caso de estudio se hallaron algunos ejemplos de estos tipos de parámetros, por lo tanto, es necesario definir una forma de expresar conjuntos de parámetros concisamente.
- **Problema 9:** Sea S un conjunto de uno o más casos de uso, y sea C un conjunto de una o más condiciones. Se puede necesitar el siguiente requisito: al menos una condición de C debe ser válida para permitir la ejecución de cada caso de uso en S .
 Por ejemplo, en un Sistema de Administración de Bibliotecas Online el conjunto S consiste de varios casos de uso que realizan alguna acción sobre un libro (préstamo, devolución, entre otras) por un bibliotecario, y C consiste de una condición que dice *el bibliotecario debe ser un super-bibliotecario o pertenecer a la misma biblioteca que la biblioteca que posee el libro*; es necesario expresar que la condición de C puede ser elegida opcionalmente como una precondition para ejecutar cada caso de uso de S .
 Por lo tanto, es necesario desarrollar una técnica de modelado para esta situación.
- **Problema 10:** En la literatura se encontraron métodos que incluyen restricciones de dependencias solamente entre acciones variantes de diagramas de actividad, sin embargo, también es necesario permitir la definición de restricciones de dependencias entre distintos tipos de variantes de diagramas de actividad y variantes de diagramas de casos de uso. Por lo tanto, el problema es cómo especificar tales tipos de restricciones de dependencias dentro de diagramas de actividad. En el caso de estudio se encontraron varios ejemplos de estos tipos de restricciones de dependencia.

En base a los problemas mencionados, el objetivo de esta sección, es definir una notación para diagramas de actividad de UML con variabilidades. Como metas para alcanzar este objetivo se pretende resolver los problemas 5, 6, 7, 8 y 9. Este objetivo contribuye a lograr la meta número 2 de la sección 1.4 del capítulo de Introducción.

En esta sección se definió una extensión a la notación de diagramas de actividad de UML para modelar variabilidades, sin alterar el meta-modelo de UML. Esta extensión provee la resolución a los problemas mencionados anteriormente.

3.2.2. Trabajo relacionado

Se consideraron como trabajos relacionados a los enfoques que satisfacen al menos una de las siguientes condiciones: modelado de puntos de variación (ya sea con un variante opcional o con un conjunto de variantes alternativos), consideración de variabilidad en flujo de datos, consideración de restricciones de dependencia, consideración de conjuntos de parámetros, resolución del Problema 6, resolución del Problema 7 y resolución del Problema 9.

La tabla 3.9 provee una comparación de los enfoques más relevantes que utilizan notaciones de diagramas de actividad con variabilidades. Los requisitos A1 hasta A8 y sus resultados son explicados en detalle a continuación de la tabla 3.9.

	A1	A2	A3	A4	A5	A6	A7	A8
[Heuer, 2010]	good	reg+	no	no	no	no	no	reg
[Razavian, 2008]	reg	no	reg +	no	no	no	no	no
[Korherr, 2007]	no	no	no	no	no	no	no	reg-
[Schnieders, 2007]	no	no	reg	no	no	reg-	no	no
Our Approach	very good	very good	very good	good	good	very good	good	very good

Figura 3.9: Comparación de notaciones de diagramas de actividad con variabilidades.

- **A1: Flexibilidad y simplicidad para describir cómo los variantes son seleccionados en variabilidad de flujo de control.** En [Heuer, 2010], esta información es capturada por medio de expresiones booleanas por fuera del diagrama de actividad; en [Razavian, 2008] solo se permite definir variantes opcionales y alternativos. En este trabajo se satisface este requisitos a través de los atributos *min* y *max* de los elementos de variabilidad.
- **A2: No hacer uso de elementos del meta-modelo de diagramas de actividad para proveer información de variabilidad de flujo de control** (Ya sea puntos de variación, información o anotaciones de variantes). En [Heuer, 2010] las aristas de actividad pueden incluir nombres de variantes. En este trabajo se satisface este requisito por completo.
- **A3: Consideración de variabilidades en flujo de datos;** idealmente se quiere: flexibilidad para describir cómo se seleccionan los variantes, y no hacer uso de elementos del meta-modelo de diagramas de actividad para proveer información en variabilidad de flujo de datos. Solo se halló consideración de variabilidad en flujo de datos en [Razavian, 2008], [Schnieders, 2007] y en este trabajo. En [Razavian, 2008] y [Schnieders, 2007] se hace uso de elementos de diagramas de actividad para representar información de variabilidad (por ejemplo nodos parámetro, nodos de actividad, output pins). En [Schnieders, 2007] no se provee en el diagrama de actividad la información de cardinalidad para variabilidad en flujo de datos y en [Razavian, 2008] las variabilidades pueden ser del tipo de punto de variación alternativo (seleccionar un variante) o punto de variación opcional (elegir cero o un variante). En este trabajo no utilizamos elementos de diagramas de actividad para proveer información de variabilidad y consideramos un uso extenso de cardinalidades para definir variabilidades de flujo de datos.
- **A4: Resolver el Problema 6:** solo en este trabajo.
- **A5: Resolver el Problema 7:** solo en este trabajo.
- **A6: Consideración de uso de conjuntos de parámetros;** idealmente con resolución del Problema 8. En [Schnieders, 2007] se considera el uso de conjuntos de parámetros; sin embargo éstos no permiten todos los algunos tipos de variabilidades identificados y tampoco permiten definir conjuntos por comprensión, debido a que es necesario incluir un elemento para cada variante. En este trabajo se resuelve el Problema 8.

- **A7: Resolver el Problema 9:** solo en este trabajo.
- **A8: Consideración de restricciones de dependencia;** idealmente con resolución del problema 10. En [Korherr, 2007] se consideran solo restricciones de dependencia entre acciones. En este trabajo se resuelve el problema 10.

3.2.3. Notación propuesta para Diagramas de Actividad con variabilidades

En esta sección se presenta la notación definida en este trabajo para modelar diagramas de actividad de UML con variabilidades con el fin de resolver los problemas enumerados del 6 al 10. En primer término se brinda el meta-modelo para anotaciones de variabilidad que representan los distintos tipos de variabilidad que se pueden incluir en los diagramas de actividad y un breve resumen de la finalidad de cada clase de variabilidad. Luego se brindan algunas definiciones útiles para darle un marco más formal a diagramas para representar comportamientos. En base a estas definiciones, a continuación, se brinda detalle de cómo se han resuelto los problemas 6 y 7 en este trabajo. Finalmente, cada tipo de variabilidad es descrita con más detalle y ejemplos.

Meta-modelo para anotaciones de variabilidad en Diagramas de Actividad

Se definió un meta-modelo para anotaciones (que serán adjuntadas a elementos propios de diagramas de actividad) con la representación de los diferentes tipos de variabilidades identificadas en diagramas de actividad. Se consideran cinco tipos de variabilidades (ver Figura 3.10): *data flow variability*, *control flow variability*, *parameterSet*, *precondition variability* y *branch variability*.

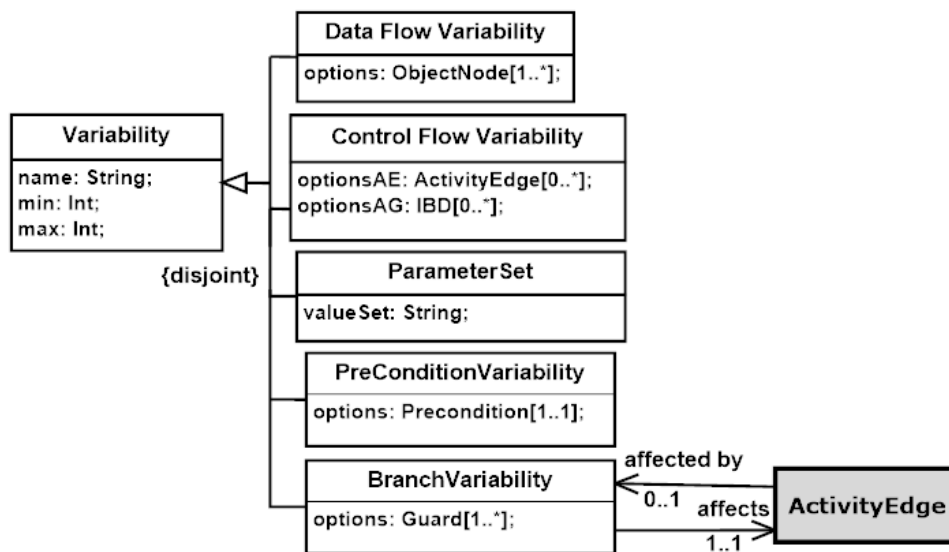


Figura 3.10: Meta-modelo para anotaciones de variabilidad en diagramas de actividad.

Una instancia de la clase *Data Flow Variability* representa una variabilidad en flujo de datos, cuyos variantes son elementos del tipo *ObjectNode* (por ejemplo pines o nodos de actividad de parámetro).

Una instancia de la clase *Control Flow Variability* representa una variabilidad en flujo de control, cuyos variantes pueden ser una arista de actividad dirigida a una acción o un diagrama representando comportamiento.

Una instancia de la clase *ParameterSet* representa un conjunto de parámetros definidos por comprensión.

Una instancia de la clase *PreConditionVariability* representa una variabilidad de condición, cuyo variante es un elemento de tipo *PreCondition* que representa una condición que se debe cumplir para un diagrama de actividad dado.

Una instancia de la clase *BranchVariability* representa una variabilidad de selección de alternativas en un nodo merge, cuyos variantes son elementos de tipo *Guard* que representan una guarda que puede

estar o no para una alternativa dada.

En general, los elementos anotados con alguno de los tipos de variabilidad anteriores representan los variantes. El nombre y la cardinalidad de la variabilidad están dados en la clase *Variability*.

Diagramas representando comportamiento

El propósito de esta sección es caracterizar los diagramas que representan comportamiento (que pueden ser reutilizables o no, que pueden ser variantes o no), los cuales no deben tener ni nodos aislados, ni dos o más componentes conectados (es decir, algunos sub-diagramas separados). Los diagramas de comportamiento son necesarios en este trabajo ya que:

- Describen *comportamientos variantes* de variabilidades en flujo de control, los cuales son representados en un diagrama de actividad o un diagrama de comportamiento.
- Describen *comportamientos reutilizables* que son reutilizados o bien en diagramas de actividad que describen casos de uso o bien en diagramas de comportamiento.
- Representan posibles soluciones a las ocurrencias de problemas 6 y 7.

Para evitar la existencia de nodos aislados se considera lo siguiente:

Definición 1: Un *diagrama con aristas de actividad* D satisface los siguientes requisitos:

1. D tiene los mismos tipos de nodos y aristas (con el mismo significado), a excepción de nodo inicial de actividad, que un diagrama de actividad.
2. Los extremos de una arista en D también están en D .
3. No hay nodos de actividad aislados en D (esto significa que cada nodo de actividad en D es un extremo de una arista o es una acción conectada a un pin que a sus vez está conectado a una arista).

Un *nodo no-objeto* es un nodo de actividad que no es un nodo objeto (del tipo *ObjectNode*).

Dado D , un diagrama con aristas de actividad:

Un *source activity node* de D es un nodo de actividad no-objeto que no está conectado en D con un

nodo de actividad predecesor no-objeto.

Un *sink activity node* de D es un nodo de actividad no-objeto que no está conectado en D con un nodo de actividad sucesor no-objeto.

Además, para evitar la existencia de dos o más componente conectados, se considera lo siguiente:

Definición 2: Un *diagrama de comportamiento* D es un diagrama que satisface las siguientes propiedades:

1. D es un diagrama con aristas de actividad.
2. Hay un solo *source activity node* en D , el cual es llamado $Source(D)$ que no es un nodo inicial de actividad.
3. Hay cero o un elemento *sink activity node* en D que no es un nodo final de actividad (en caso de existir es llamado $SinkNotFinal(D)$).
4. Se permite que haya nodos sink en D del tipo nodos finales de actividad.

Definición de mecanismo de reutilización para resolver los problemas 6 y 7

Solución Problema 7. Con el fin de resolver el problema 7, es necesario expresar que ciertas partes de un diagrama reutilizable (representando un comportamiento) son consideradas solamente para un contexto específico de reutilización (por ejemplo un conjunto de casos de uso, un conjunto de diagramas de comportamiento, un paquete de casos de uso). Para este propósito, se brinda una solución al problema 7 considerando lo siguiente:

Definición 3: un *execute behavior node* (EB) es un nodo descrito con un diagrama de actividad *D* que satisfice las siguientes propiedades:

1. *D* es un diagrama de comportamiento.
2. Puede haber nodos de decisión cuyas guardas sean contextos.

Un *contexto* es un conjunto de *elementos de contexto*, que pueden ser: paquetes de casos de uso (todos los casos de uso dentro del paquete), casos de uso y diagramas de comportamiento. Para cada *elemento de contexto* se utiliza la sintaxis: *tipo, nombre de elemento*, donde *tipo* de un elemento de contexto puede ser: *UC* (para caso de uso), *UCP* (para paquete de casos de uso), *IBD* (para diagrama de comportamiento inducido), *EB* (para execute behavior), *CBA* (para call behavior action).

Una guarda con un contexto *C* que está dentro de un digrama reutilizable *D* asociado a un nodo execute behavior *N* que se encuentra dentro de *A* chequea si la ocurrencia de *N* dentro de *A* es con el contexto *C*.

Solución Problema 6. El problema 6 puede ser resuelto utilizando un EB con la siguiente propiedad: el diagrama *D* que describe el EB no tiene guardas de contexto y es un diagrama de comportamiento que contiene al menos un *activity sink node* que es un nodo final.

Ejemplos. La figura 3.11 muestra un diagrama reutilizable para un EB llamado *Obtain due date* (se encarga de obtener la fecha de vencimiento de un ítem del a biblioteca); este diagrama es reutilizado en los casos de uso *renewing* y *check out*, respeta la definición 3, e incluye una variabilidad de flujo de control *specifyDueDate* (con *min = 1* y *max = 1*) con dos variantes que son diagramas de comportamiento: *enter due date* (el bibliotecario ingresa la fecha de vencimiento), y *due date calculation* (se calcula la fecha de vencimiento automáticamente y hay una guarda con un contexto *UC renewing* con el significado: la variabilidad de flujo de control *InitialDueDateCalculation* debe ser ejecutada cuando el caso de uso *renewing* está reutilizando el diagrama). El elemento variabilidad *RulesForCalculation* (con *min=1* y *max=1*) debe ser considerado independientemente del caso de uso usando el EB.

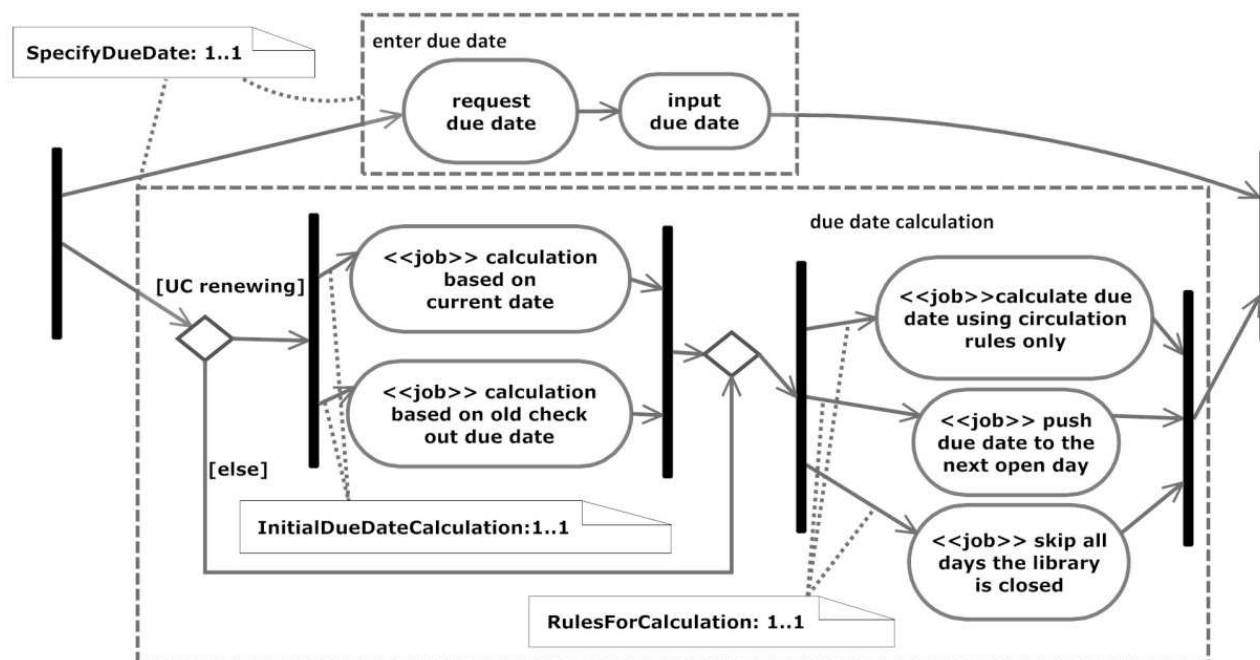


Figura 3.11: Diagrama de actividad para el elemento «execute behavior» *Obtain due date*.

Los nodos de decisión con guardas de contexto pueden ser utilizados dentro del diagrama de descripción de un EB para elegir entre dos o más caminos (de nodos y aristas). Por ejemplo, la figura 3.12 muestra un

diagrama para el «*execute behavior*» *suggest actions for records* (utilizado para incluir sugerencias en campos de entradas) que es reutilizado en cuatro casos de uso: *search catalog in the OPAC*, *show record in the OPAC*, *search the catalog in staff client*, y, *show the catalog in staff client*; este diagrama tiene varios nodos de decisión con guardas que son contextos. Por ejemplo, el nodo de decisión más a la izquierda del diagrama tiene dos contextos que son los paquetes de casos de uso *staff cliente* y *OPAC*. El resto de los nodos de decisión con contexto tienen casos de uso como contexto; la idea es que para cada caso de uso que reutiliza se elige un contexto diferente durante la ejecución del diagrama.

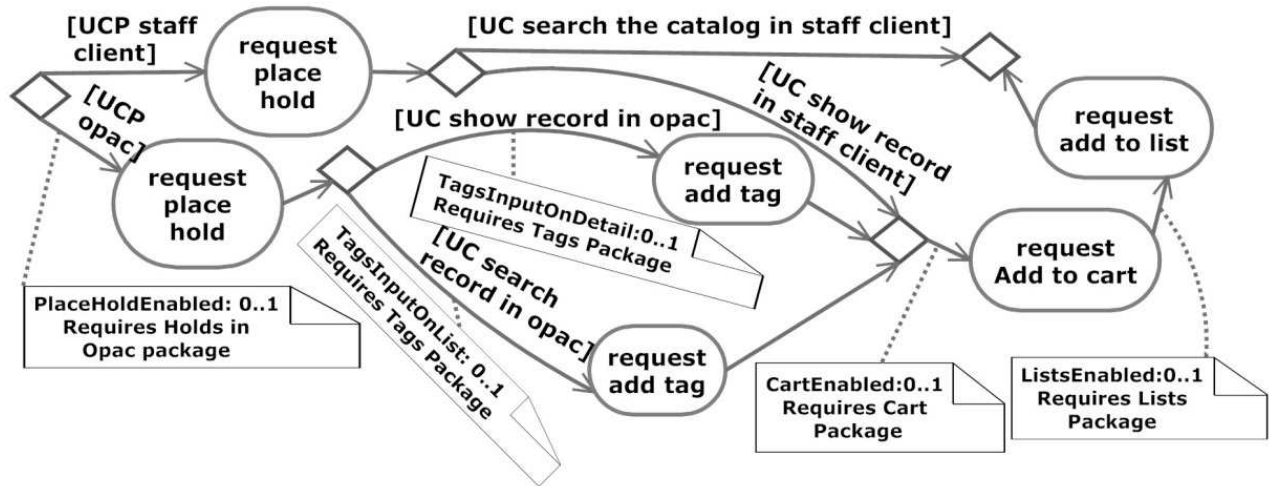


Figura 3.12: Diagrama de actividad para el elemento «*execute behavior*» *suggest actions for records*.

Los nodos de decisión con guardas de contexto pueden ser utilizados dentro del diagrama de descripción de un EB para decidir si un elemento variabilidad de flujo de control es considerado o no. Por ejemplo, la figura 3.13 muestra un diagrama para el elemento «*execute behavior*» *add item types, authorized value and 856u* que es reutilizado en dos casos de uso: *search catalog in the OPAC* y *show record in the OPAC*; este diagrama tiene guardas con contexto que son los casos de uso *search catalog in the OPAC* y *show record in the OPAC*.

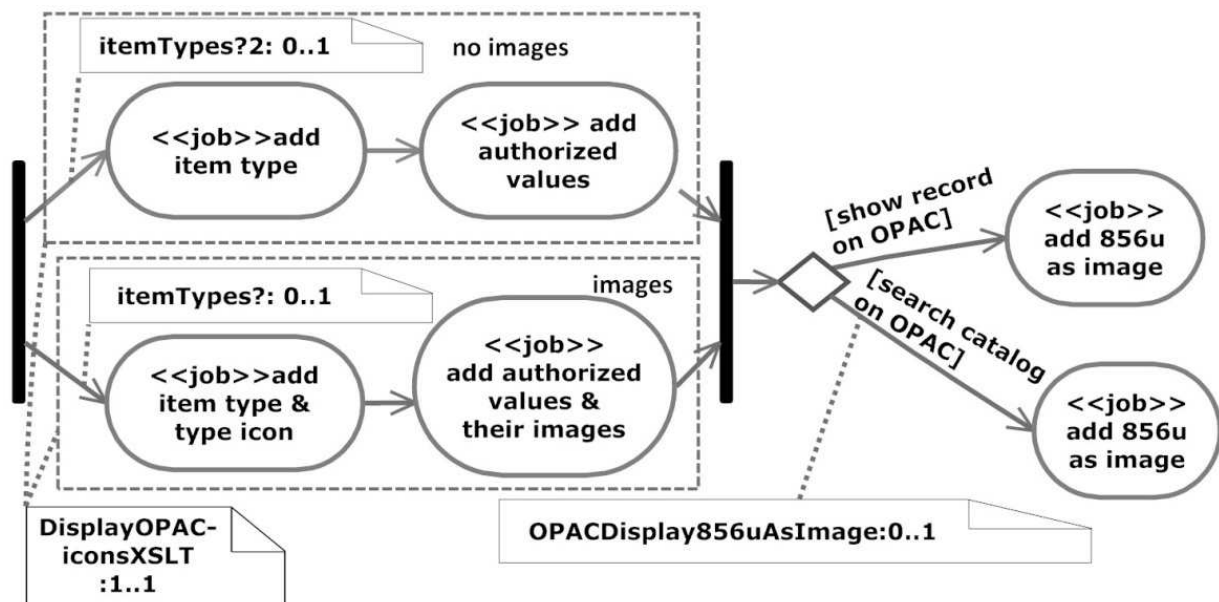


Figura 3.13: Diagrama de actividad para el elemento «execute behavior» *add item types, authorized value and 856u*.

Reglas para reutilizar un Execute Behavior Node. Para reutilizar un EN *N* dentro de un diagrama de actividad *A* se deben seguir las siguientes reglas:

1. *N* no debe tener pines en *A*; pero, de ser necesarios, se pueden conectar con *N* (por medio de aristas de flujo de datos) nodos de objeto (cuando el origen y/o los nodos sink no finales de la descripción de *N* consumen y producen tokens de datos).
2. *N* debe ser conectado solo con un nodo de actividad predecesor en *A* que no es un nodo de objeto, y si la descripción de *N* tiene un nodo sink no final, entonces *N* debe ser conectado solo con un nodo de actividad sucesor en *A* que no es nodo de objeto.

Es necesario definir, describir y reutilizar un EB con diagrama *D* cuando al menos una de las siguientes condiciones es válida:

1. *D* contiene nodos de actividad final que son utilizados para terminar la ejecución del diagrama de actividad que reutiliza (la relación de reutilización es transitiva, es decir, si un diagrama de actividad *A* reutiliza un EB *N* y la descripción de *N* reutiliza *D*, entonces *A* reutiliza *D*).
2. Es necesario usar en *D* nodos de decisión con guardas de contexto.

En el resto de los casos, se recomienda utilizar call behavior actions para reutilización.

Ejemplo. La figura 3.14 muestra el diagrama de actividad para el caso de uso *renewing* (para renovar el préstamo de un ítem). En este diagrama se puede ver el elemento «execute behavior» *obtain due date* que permite la reutilización del diagrama de la figura 3.11

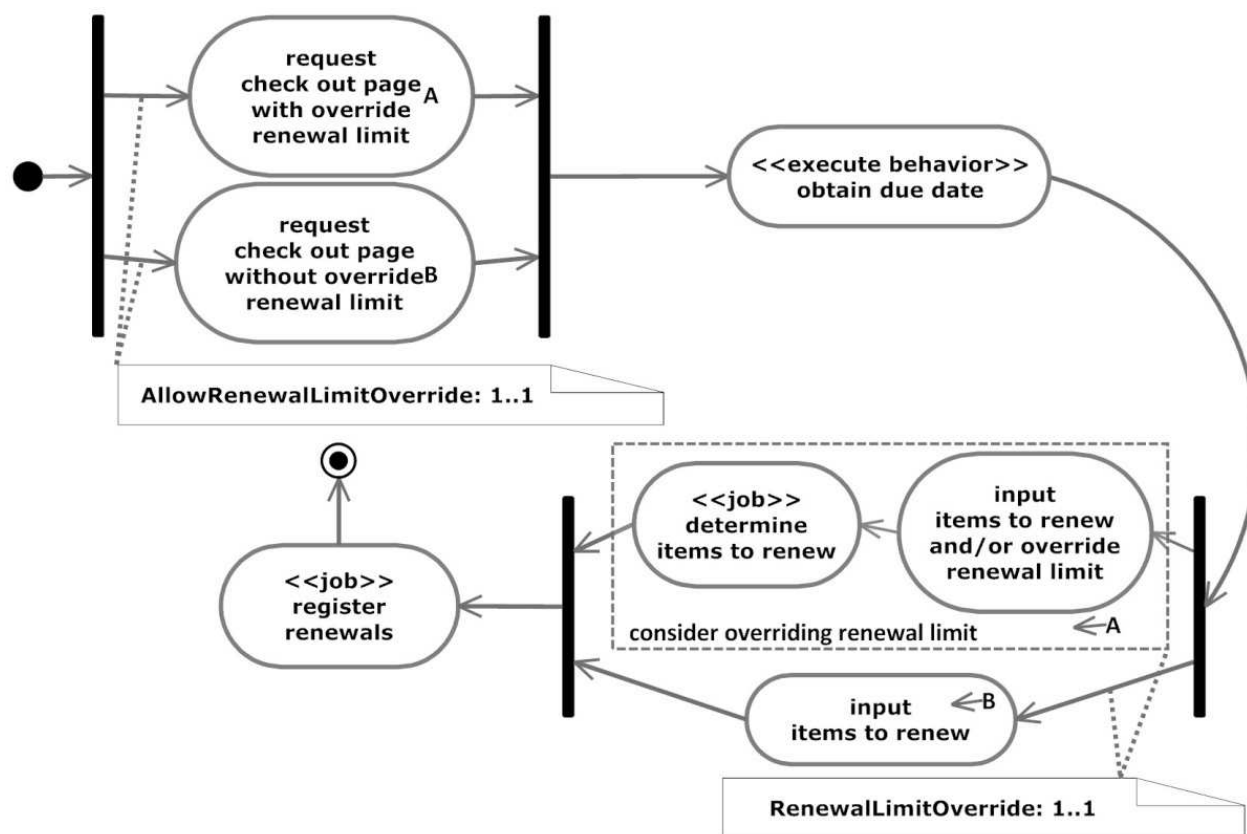


Figura 3.14: Diagrama de actividad para el caso de uso *renewing*.

Semántica para reutilizar un Execute Behavior Node. Se brinda una semántica informal para reutilizar un EB dentro un diagrama de actividad o de un diagrama de comportamiento.

La *expansión de un EB N* (N tiene asociado un diagrama de actividad D) con ocurrencia O dentro de A (que es un diagrama de actividad o un diagrama de comportamiento) –llamada *expansion(N, O)*– es un diagrama de actividad que es obtenido a partir de D considerando solo el contexto que contiene a O (es decir, en D los caminos correspondientes a los contextos que no contienen a O son borrados).

La semántica de A (que es un diagrama de actividad o un diagrama de comportamiento) es la *expansión de A* , que se computa a través del siguiente algoritmo:

expansion := A ;

Repeat

Find any occurrence O in *expansion* of an EB N

Replace in O the node N by *expansion(N, O)*

Until *expansion* does not contain an occurrence of an EB node

Se dice que un EB N *depende directamente* de un EB M si M ocurre dentro del diagrama asociado a

N . Se dice que un EB N es *dependiente* de un EB M si N depende directamente de M o hay un camino de dependencias desde N a M . Se dice que un EB es *recursivo* si depende de sí mismo.

El bucle del algoritmo definido anteriormente no finalizará hasta que todos los EB no sean recursivos.

Variabilidad de flujo de control

Uno de los tipos de variabilidades posibles en diagramas de actividad, es la variabilidad en flujo de control. Esta variabilidad puede incluir dos tipos de variantes: aristas (que apuntan a un nodo) y diagramas de comportamiento. Para darle un marco formal al uso de diagramas de comportamiento

como variantes, y con el objetivo de resolver el Problema 5, a continuación se brindan una serie de definiciones.

La clase del meta-modelo de anotaciones de variabilidad *Control Flow Variability* es utilizada para representar variabilidad en flujo de control.

Es fácil notar que un diagrama de comportamiento variante D dentro de un diagrama A (que puede ser un diagrama de actividad o uno de comportamiento) debe ser un subdiagrama completo de A (es decir, D debe contener todas las aristas de actividad y las relaciones de A que están relacionadas con nodos de D). Este requisito es expresado formalmente de la siguiente manera:

Definición 4: Un *subdiagrama inducido de D* es un diagrama X que satisface las siguientes propiedades:

1. X es un diagrama de comportamiento.
2. X contiene todas las aristas de actividad de A que unen nodos en X .
3. Si un pin p de X está relacionado con una acción a de X en A , entonces p también está relacionado con a en X .
4. Si una acción está asociado con un pin p de A que está en X , entonces p también está en X .

Además, un diagrama de comportamiento variante D dentro de un diagrama A debe estar conectado a nodos de A ; este requisito es especificado de la siguiente manera:

Definición 5: Un *diagrama de comportamiento inducido (induced behavior diagram - IBD -)* con respecto a A es un diagrama X que satisface las siguientes propiedades:

1. X es un *subdiagrama inducido* de A .
2. $Source(X)$ tiene solo un nodo predecesor no objeto fuera de X .
3. En X hay cero o un nodo de actividad con solo un nodo sucesor no objeto fuera de X ; tal nodo puede ser un nodo de decisión o un nodo de actividad sink.

Resolución Problema 5. Los variantes deseados para la resolución del problema 5 son caracterizados como variantes IBD que tienen al menos un nodo de actividad sink que es un nodo final.

Reglas para variantes en variabilidades flujo de control. A continuación se listan las reglas dadas para los tipos de variantes que pueden ser utilizados en variabilidades de flujo de control:

- *Un elemento de variabilidad de flujo de control con un variante opcional.* El variante puede ser:
 - Un IBD con un nodo de actividad sink que no es un nodo final.
 - Una arista de actividad apuntando a alguno de los siguientes elementos: una acción, un call behavior action o un EB.
- *Un elemento de variabilidad de flujo de control de selección entre más de un variante.* Un variante puede ser:
 - Un IBD.
 - Una arista de actividad apuntando a alguno de los siguientes elementos: una acción, un EB, un call behavior action o un nodo final de actividad.

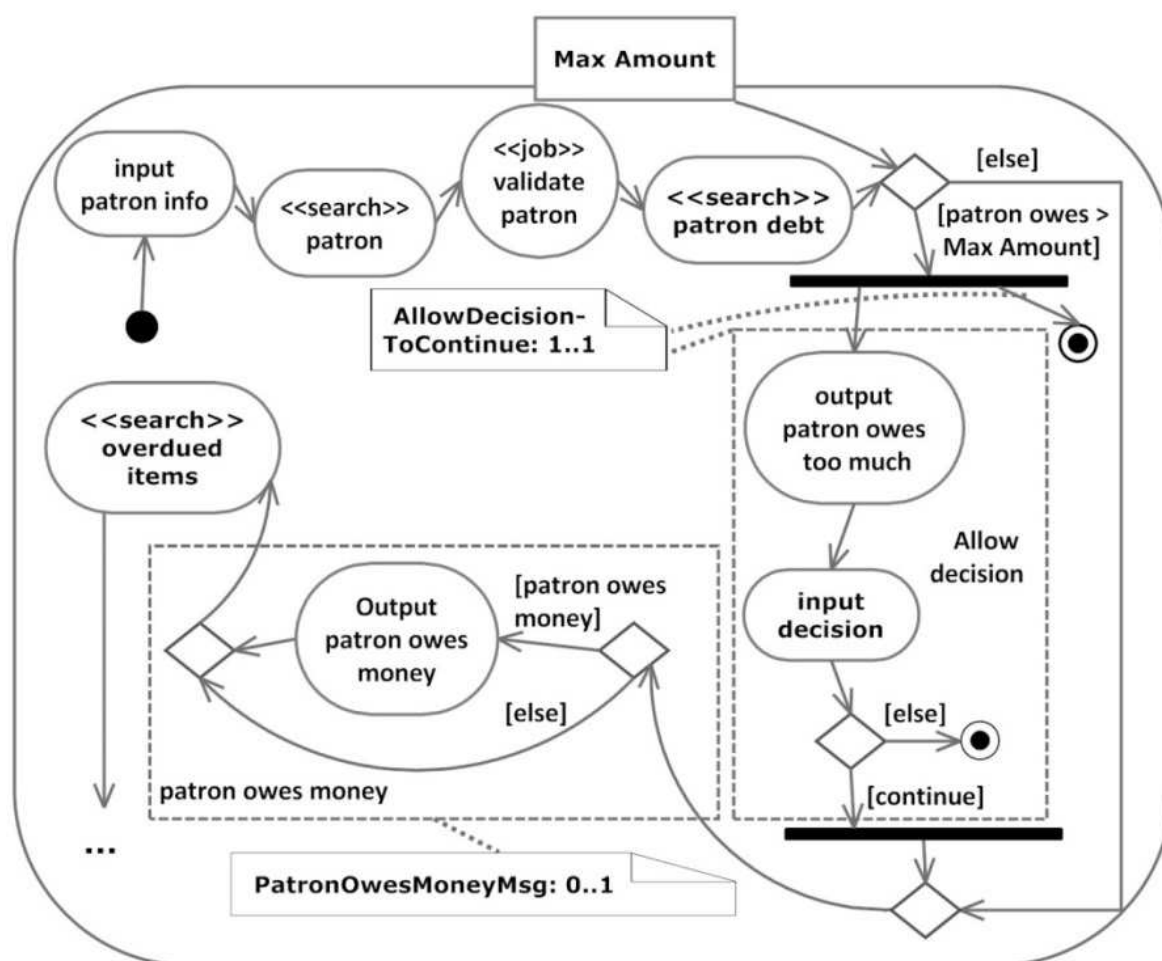


Figura 3.15: Parte de la descripción del caso de uso *check out*.

Ejemplo. La figura 3.15 muestra parte de la descripción del caso de uso *check out* (utilizado para efectivizar el préstamo de un ítem de la biblioteca). Para el caso en el que un cliente debe demasiado en préstamos se incluye la variabilidad *AllowDecisionToContinue*, que permite seleccionar entre dos variantes: el nodo final de actividad (se termina el proceso ya que el cliente está bloqueado) y un IBD llamado *Allow decision* (el problema 5 caracteriza situaciones como ésta); este IBD muestra un mensaje de que el cliente debe demasiado y permite al bibliotecario decidir si continúa con la operación de *check out* o no (observar el nodo final de actividad dentro del IBD). A continuación, se presenta la variabilidad *PatronOwesMoneyMsg* con el IBD variante *patron owes money*, el cual primero chequea si el cliente debe dinero; y si esta condición es verdadera muestra un mensaje informando esto.

Variabilidad de Conjuntos Parámetros

Como se puede ver en la figura 3.16, un conjunto parámetro (elemento *ParameterSet*) tiene asociado un conjunto de valores (expresados por medio del atributo *valueSet*); a partir de *valueSet* se puede elegir un subconjunto de valores con cardinalidad en el intervalo [min,max]. El valor del meta-atributo *valueSet* es expresado por medio de un conjunto de valores definido por extensión o por comprensión (esto es necesario para la resolución del problema 8). *ParameterSet* es utilizado para representar variabilidad.

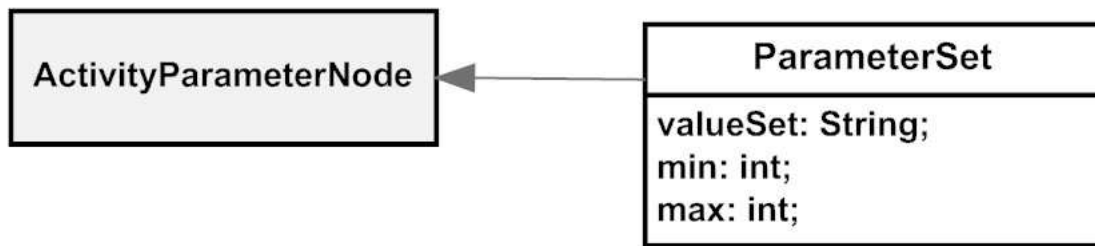


Figura 3.16: Definición de conjuntos parámetro.

Para representar un conjunto parámetro que forma parte de un diagrama de actividad se utiliza un elemento del tipo *ActivityParameterNode* que recibe un token con el subconjunto elegido de *valueSet*. Además, se debe describir a una call behavior action dentro de un diagrama de actividad que utilizan conjuntos parámetro con un nodo de actividad, y los conjuntos parámetro utilizados deben ser representados utilizando nodos de actividad parámetro (elementos del tipo *ActivityParameterNode*).

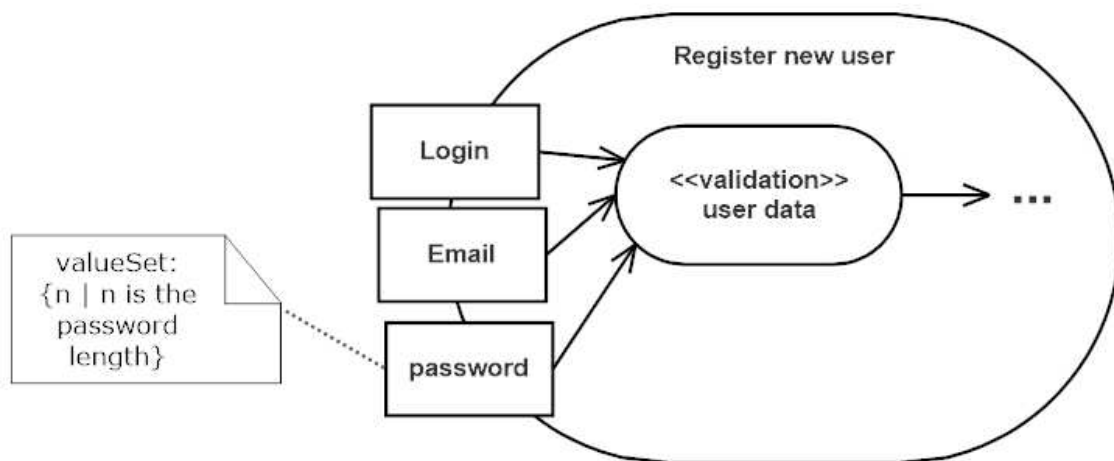


Figura 3.17: Ejemplo de variabilidad de conjunto parámetro para tarea de agregar un registro de nuevo usuario a una aplicación.

Ejemplo. La figura 3.17 muestra un posible uso de variabilidad de conjuntos parámetro. Se supone el caso de una actividad para registrar un usuario en una aplicación, donde los valores que se reciben son el nombre de usuario (nodo *Login*), la contraseña de usuario (nodo *password*) y el correo electrónico (nodo *Email*). El elemento de tipo *ActivityParameterNode password*, tiene asociada una variabilidad de parámetro, con el atributo *valueSet* representando la longitud de la contraseña permitida para una aplicación (esto puede variar y debe ser seleccionado con los atributos *min* y *max*).

Variabilidad de flujo de datos

Otro de los tipos de variabilidad que se pueden expresar en un diagrama de actividad, es la variabilidad en flujo de datos. Los diagramas de actividad tienen elementos del tipo *ObjectNode*. Un *ObjectNode* es un nodo de actividad abstracto que es utilizado para definir flujo de objetos o datos en una actividad. Las especializaciones de *ObjectNode* incluyen elemento pin, central buffer, parameter y expansion nodes. Para el fin de variabilidad en flujo de datos en este trabajo utilizaremos *ObjectNode* de manera más abstracta o bien alguna de sus especializaciones. En general, se utilizan las especializaciones pines (representan datos de entrada o salida en una acción), parameter o *ParameterSet* (especialización definida aquí de *ActivityParameterNode*).

La clase del meta-modelo de anotaciones de variabilidad *Data Flow Variability* es utilizada para representar variabilidad en flujo de datos.

Ejemplo. La figura 3.18 muestra parte de la descripción del caso de uso *search catalog in the OPAC* (utilizado para que los clientes busquen y visualicen ítems de la biblioteca), esta parte viene después de la búsqueda de ítems y considera la manera de formatear y presentar estos ítems al cliente. El caso de uso considera una variabilidad de flujo de control llamada *OPACXSLTResultsDisplay* para mostrar un ítem, que tiene dos IBD variantes: *normal view* (para mostrar los detalles del ítem sin ningún procesamiento previo), y *XSLT view* (para mostrar un ítem utilizando una plantilla de estilos XSLT); este variante consiste de agregar diferentes tipos de contenido a un fracción de XML (EB *add item type, authorized values, and 856u*), y de mostrar el resultado de la ejecución de la plantilla de estilos sobre este XML (*output result action*).

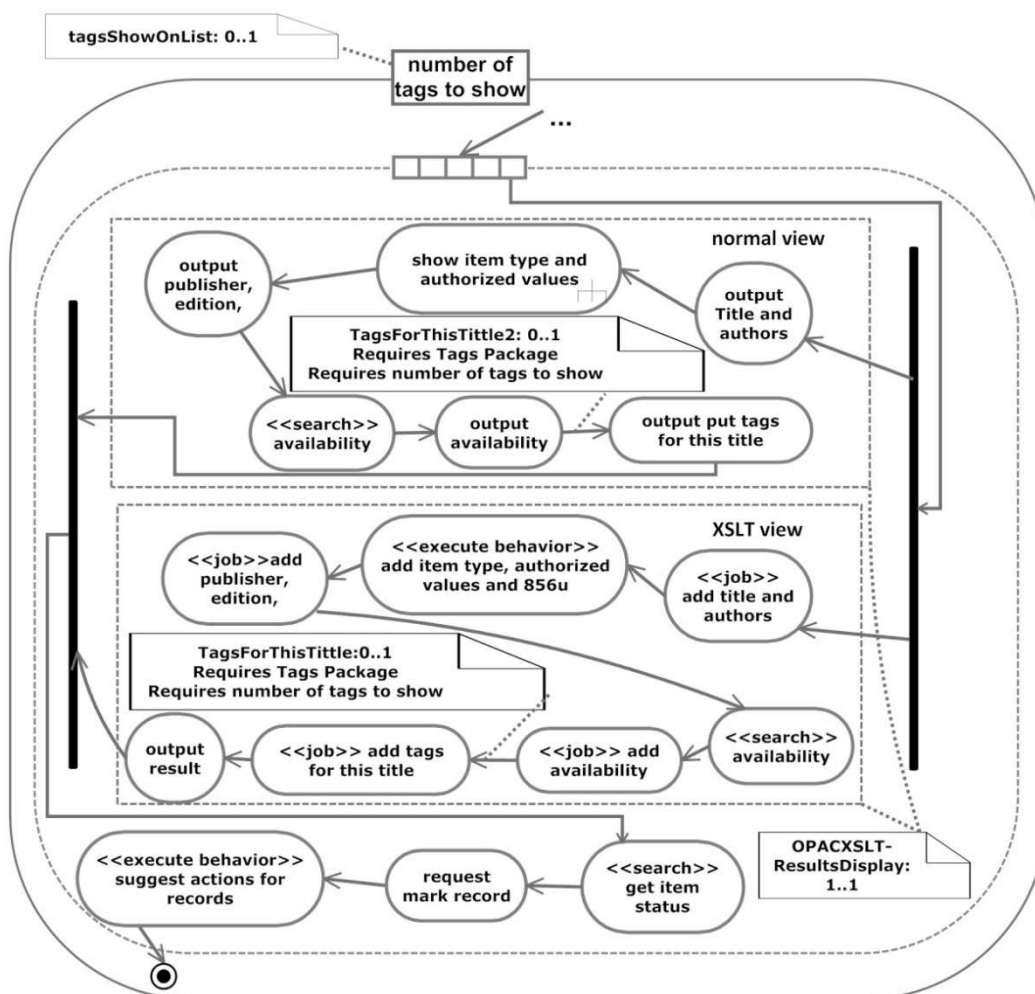


Figura 3.18: Parte de la descripción del caso de uso *search catalog in the OPAC*.

Variabilidad según condiciones

El objetivo de esta sección es la resolución del problema 9, la cual usa el concepto de variabilidad de condiciones. Una *variabilidad de condición* es un elemento de variabilidad cuyos variantes son una o más condiciones; si un elemento de variabilidad de condición tiene un solo variante, entonces su cardinalidad es 0..1, sino su cardinalidad debe ser 0..1 ó 1..1. Los elementos de variabilidad de condición se utilizan en las siguientes dos situaciones:

Situación 1: Variabilidad en pre-condiciones. Cuando una pre-condición variante seleccionada necesita ser válida antes de la ejecución de la descripción de casos de uso (por intermedio de un diagrama de actividad). Esta pre-condición puede ser necesaria o no, dependiendo la aplicación.

Esto se representa con una anotación con la información de variabilidad (*name* -nombre-, valores *min*

y *max*) y la de *pre-condición variante* y representa la solución al Problema 9. Para este fin se creó la meta-clase *PreConditionVariability*.

Ejemplo. La figura 3.19 muestra el elemento de variabilidad de pre-condición *independent branches*, que involucra a todos los diagramas de actividad del paquete de casos de uso llamado *Editing library objects by staff* (contiene casos de uso para edición y tratamiento de objetos tales como ítems, clientes). La pre-condición *staff = superlibrarian and staff belongs to the same library* hace decir que para que alguien pueda realizar las operaciones de los casos de uso debe ser un superbibliotecario o pertenecer a la misma biblioteca. Para simplificar la notación se agrega la pre-condición al paquete que contiene todos los casos de uso que serán descritos con diagramas de actividad.

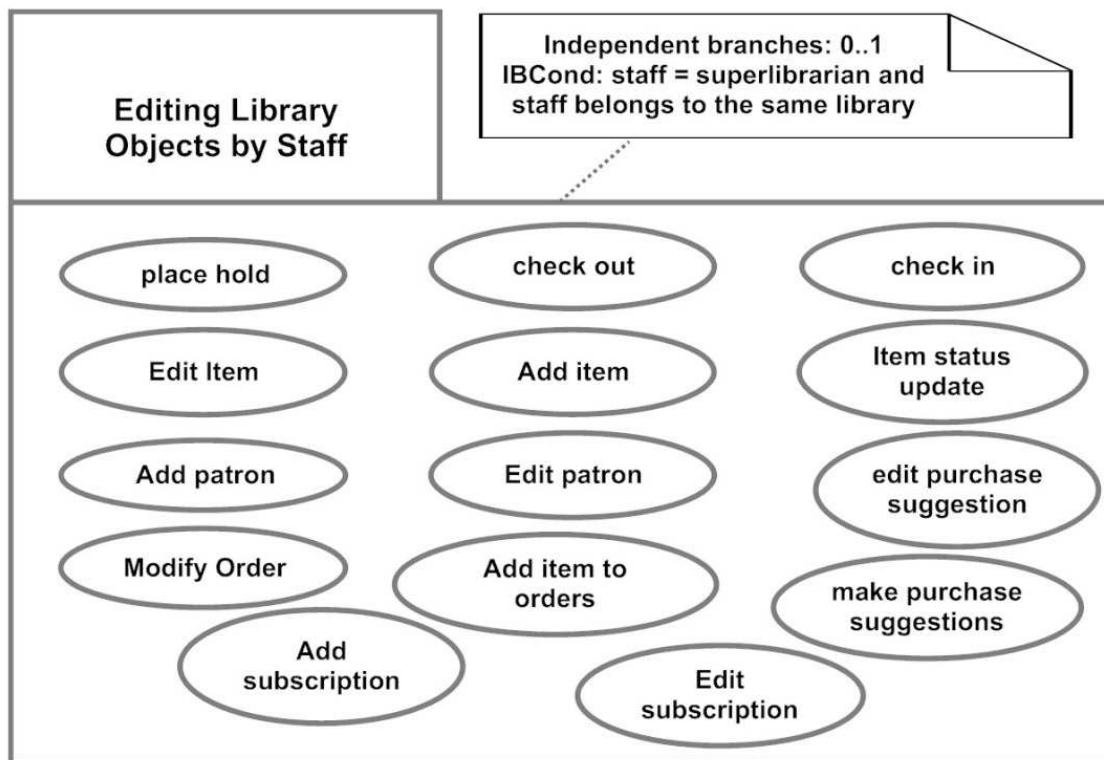


Figura 3.19: Elemento de variabilidad de condición *independent branches*.

Situación 2: Variabilidad en ramas de decisión. Cuando una condición que fue elegida de un conjunto de condiciones alternativas (para el producto variante) debe ser válida para la ejecución de una acción o un comportamiento.

Esta situación se describe con un nodo de decisión y una anotación en una de sus ramas que incluye la información usual de variabilidad (*name* -nombre-, *min* = 1 y *max* = 1) y un conjunto de condiciones alternativas.

Una situación común de variabilidad de condición consiste de una regla de negocios variable que se debe ser elegida para permitir la ejecución de una acción o un comportamiento.

No se halló este tipo de variabilidad en otros enfoques de diagramas de actividad con variabilidades. Para la resolución de este tipo de variabilidad se creó la meta-clase *BranchVariability*

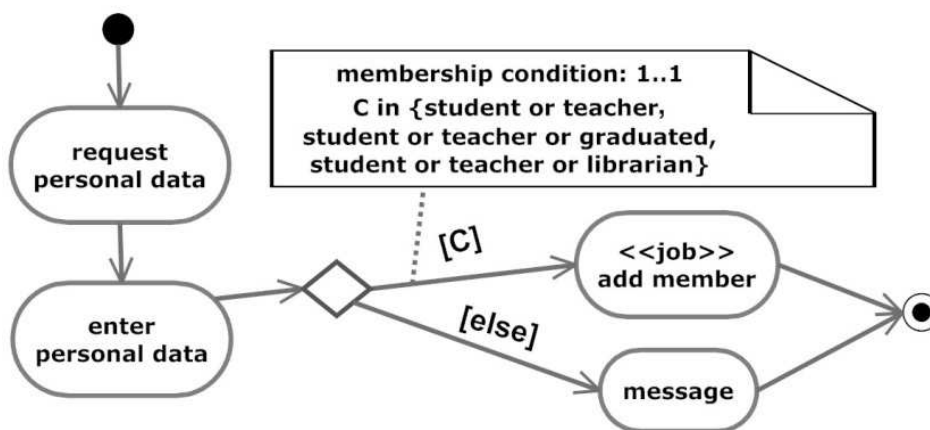


Figura 3.20: Regla de negocio para agregar un miembro a la biblioteca.

La figura 3.20 muestra un ejemplo de variabilidad de condición (*membership condition*) del tipo de la situación 2. Esta variabilidad se utiliza para la rama que dirige a la acción *addMember* (agregar un miembro a la biblioteca) y es utilizada para decir que solo una de las tres condiciones presentes debe ser elegida.

Restricciones de dependencia

En la Figura 3.21 se presentan las meta-clases para restricciones de dependencia que incluyen variabilidades de diagramas de actividad. Un variante (elemento de tipo *variant*) consiste de un nombre de variabilidad y de un variante de diagramas de actividad (elemento *AD variant*, que puede ser un elemento del tipo *ObjectNode*, un elemento *Condition* –meta-clase que se especializa con las meta-clases *PreCondition* o *Guard*–, un elemento *ActivityEdge* o un elemento *InducedBehaviorDiagram*) o de un variante de diagramas de casos de uso (elemento *UCD variant*, que puede ser una asociación o un paquete). Al igual que para casos de uso, una restricción de dependencia de diagramas de actividad puede ser de tipo *excludes* o *requires* y tiene un atributo *name*. Existen dos tipos de restricciones de dependencia que incluyen diagramas de actividad:

- *Restricciones de dependencia entre variantes de diagramas de actividad*: este tipo de restricciones tienen un elemento *dependent*, el cual puede ser un variante de flujo de control, una condición o un elemento *ObjectNode*. El elemento *dependent* depende de un conjunto no vacío de variantes dentro del diagrama de actividad (obtenidos mediante la navegación de los roles *in* y *dependence on* de las meta-asociaciones correspondientes).
- *Restricciones de dependencia entre un variante de diagramas de actividad y un variante de diagramas de casos de uso*: este tipo de restricciones tienen un elemento *dependent* que es un variante de flujo de control. El elemento *dependent* depende de un variante de diagrama de casos de uso (obtenido mediante la navegación de los roles *in* y *dependence on* de las meta-asociaciones correspondientes). Este tipo de restricciones de dependencias es definido para resolver el Problema 1.

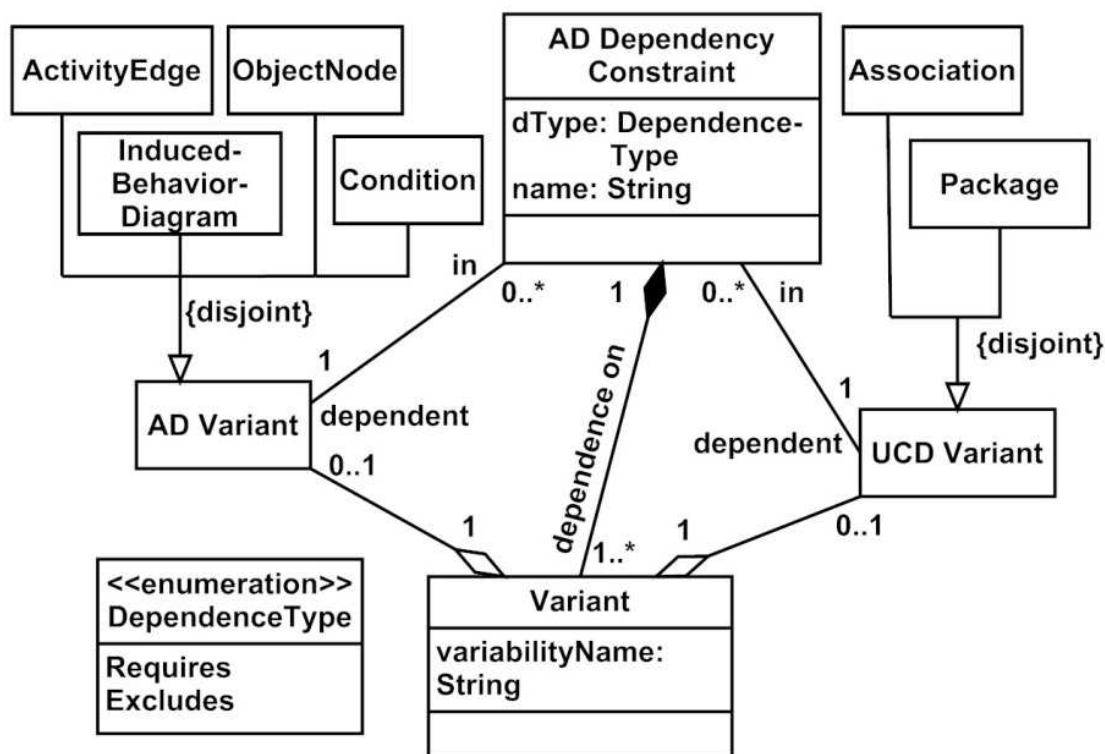


Figura 3.21: Restricciones de dependencia para diagramas de actividad y ejecuciones de comportamiento.

En la figura 3.14, el variante *renewalLimitOverride* contiene adentro otros dos variantes: el diagrama de comportamiento *consider overriding renewal limit* (permite al bibliotecario cambiar el límite de renovación si así lo desea), y la acción *input items to renew* (no permite cambiar el límite de renovación). La arista de actividad variante que apunta a la acción de tipo «output request» *check out page with override renewal limit* requiere del IBD variante *consider overriding renewal limit*. La arista de actividad variante que apunta a la acción de tipo «output request» *check out page without override renewal limit* requiere de la arista de actividad variante que apunta a la acción de tipo «input» *items to renew*.

En la figura 3.18 hay una restricción de dependencia de tipo *Requires* entre la arista de actividad variante que apunta a la acción de tipo «job» *add tags for this title* y el paquete de casos de uso variante *Tags*.

En la figura 3.12 el «execute behavior» *suggest actions for records* tiene una restricción de dependencia que dice que la arista de actividad variante que apunta a la acción de tipo «output request» *place hold* requiere del paquete de casos de uso *Holds in Opac*.

La figura 3.22 muestra una restricción de dependencia que dice que la arista de actividad variante que apunta a la acción de tipo «job» *move items to location CART* en el diagrama de actividad que describe el caso de uso *check in* requiere de la asociación variante entre el actor *timer* y el caso de uso *change from CART to permanent location* del paquete de casos de uso *Circulation*.

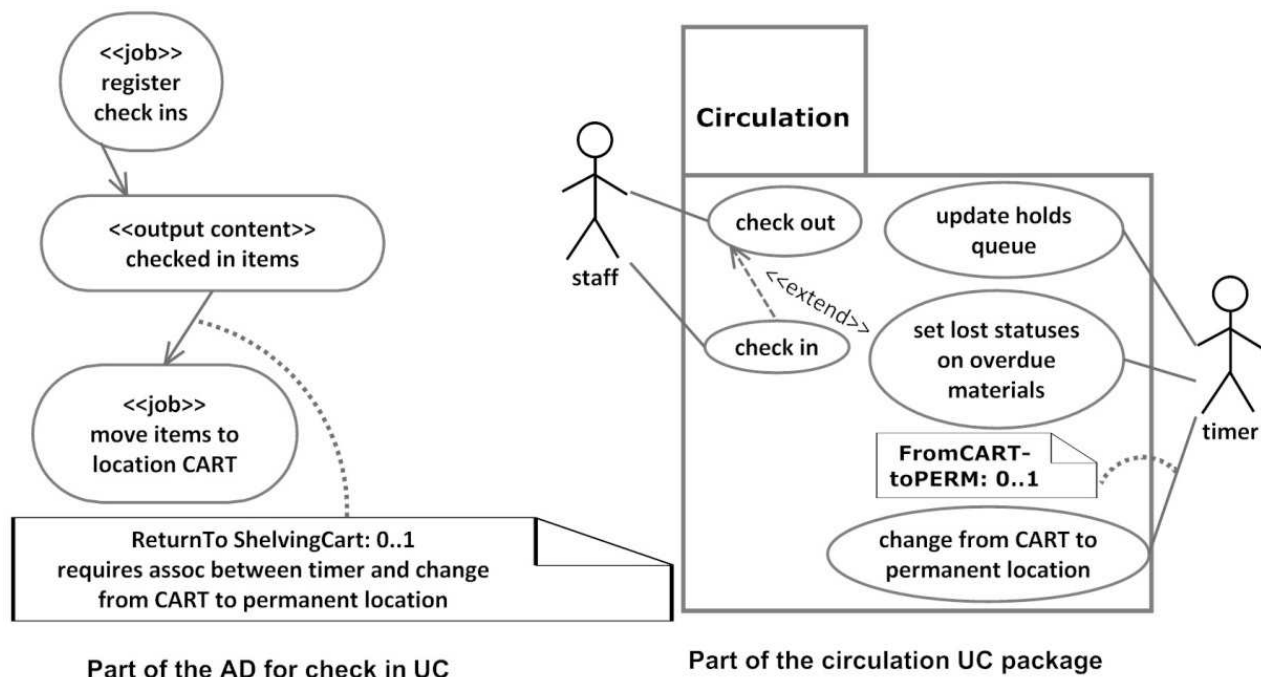


Figura 3.22: Restricción de dependencia entre una arista de actividad variante y un variante asociación de diagrama de casos de uso.

La figura 3.23 incluye una parte del caso de uso *place hold in staff client* donde la arista de actividad variante que apunta a la acción de tipo «job» *filter items belonging to other branches* depende de la condición variante *IBCond* del elemento de variabilidad de condición de la figura 3.19.

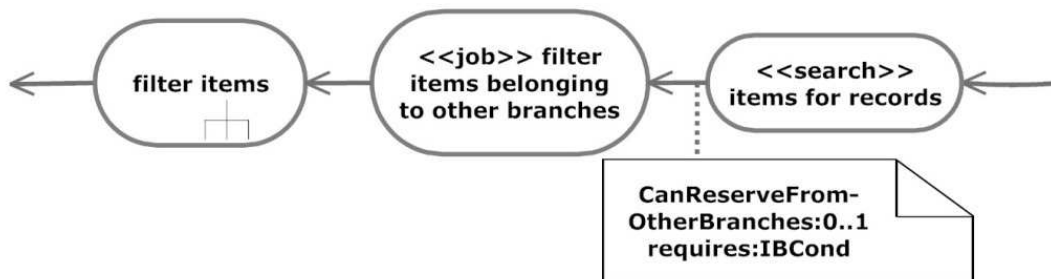


Figura 3.23: Restricción de dependencia entre una arista de actividad variante y un variante de condición.

3.2.4. Evaluación

Se define un caso de uso complejo como un caso de uso que tiene una descripción con varias variabilidades y restricciones de dependencia y al menos siete nodos del tipo: action, EB, call behavior action. Ya que el caso de estudio incluye demasiados casos de uso descritos con diagramas de actividad, en este análisis solo se consideraron casos de uso complejos. Hay un total de 11 casos de uso complejos.

Al igual que para diagramas de casos de uso con variabilidades, para analizar la propuesta de diagramas de actividad con variabilidades se utilizó el enfoque *<problema, pregunta, resultado basado en una métrica>*; donde *problema* es alguno de los problemas de la sección 3.1.1.

Para el *problema 5*: ¿Es demasiado escasa la cantidad de IBD con nodos de actividad final? No, ya que se halló que el 22% de IBD tienen un nodo de actividad final (esto es de un total de 36 IBD presentes en diagramas de actividad que describen casos de uso complejos).

Para el *problema 7*: ¿Es muy común que haya diagramas de actividad que describen casos de uso

complejos con nodos «`execute behavior`» cuyos diagramas incluyen guardas de contexto? Se encontró que el 57% de los casos de uso complejos incluyen nodos «`execute behavior`».

Para el *problema 8*: ¿Es elevada la cantidad de conjuntos parámetros que necesitan definir el conjunto de sus valores por comprensión? Se halló que este tipo de variabilidad representa el 72% de las variabilidades de parámetro en diagramas de actividad que describen casos de uso complejos.

Para el *problema 10*: ¿Es elevada la cantidad de restricciones de dependencia entre variantes de diagramas de actividad y variantes de diagramas de casos uso? Se encontró que un 54% de las restricciones de dependencia incluidas en los diagramas de actividad que describen casos de uso complejos satisfacen esta propiedad.

3.3. Framework NFR con variabilidades

3.3.1. Background

En el modelado de aplicaciones web se tiende a descuidar o a no tener en cuenta el modelado de requisitos no funcionales. Sin embargo, tener en cuenta modelado de requisitos no funcionales, y sus posteriores refinamientos, permite descubrir funcionalidades útiles e importantes para las aplicaciones web, además de colaborar a asegurar la calidad del software por medio de restricciones que se tienen que cumplir. En general, los enfoques que tratan solo con requisitos funcionales no tienen en cuenta varios asuntos del área de requisitos no funcionales, tales como:

1. El impacto de cualidades no funcionales sobre las partes funcionales de un sistema.
2. Cómo algunas metas contribuyen a otras metas.
3. La correlación entre metas de calidad.
4. Argumentaciones.
5. Priorización de una meta.
6. Procedimientos de evaluación.

Tanto como para para requisitos funcionales como para requisitos no funcionales, el paradigma de desarrollo SPL representa una forma de mejorar la productividad y la calidad de un producto a través de utilizar la ventaja de la reutilización sistemática de elementos de software. Una buena opción para aprovechar las ventajas del paradigma SPL en requisitos no funcionales es definir una notación para requisitos no funcionales que tenga en cuenta variabilidades.

En la bibliografía se hicieron algunos intentos por combinar requisitos no funcionales con desarrollo SPL. Algunos enfoques extienden modelos con elementos provenientes de requisitos no funcionales o atributos de calidad (ver [Benavides, 2005], [Tan, 2013], y [Bartholdt, 2010]), o con elementos provenientes de aspectos (ver [Tizzei, 2012], [Tan, 2013] y [Kulesza, 2005]). El trabajo presentado en [Khaliq, 2017] presenta un enfoque basado en MDE (Model Driven Engineering), donde en base a modelos en una notación NFR para productos variantes aplicando transformación de modelos se genera un modelo SPL utilizando una notación NFR para SPL. El punto en común de estos enfoques es que permiten expresar cómo una feature o elemento de modelado (representando o incluyendo una cualidad no funcional o aspecto) afecta a otros elementos provenientes de requisitos funcionales (asunto 1 del listado anterior); pero no tienen en cuenta los otros aspectos antes mencionados (asuntos de la lista desde 2 hasta 6).

El trabajo presentado en [Zhang, 2012] adapta nociones del framework NFR para identificar sub-atributos de calidad de una SPL, luego representa los modelos NFR para atributos de calidad incluyendo sub-atributos de calidad variantes en un modelo de features que es agregado al modelo de features proveniente de requisitos funcionales. El problema de este enfoque es que permite modelar variantes solo a nivel de sub-atributos de calidad y no representa variabilidad en el modelo propio de NFR, lo cual puede ser útil para validar requisitos no funcionales con el cliente de una manera independiente de requisitos funcionales.

El trabajo que se presenta en [Gonzalez-Huerta, 2012b] extiende el enfoque orientado a atributos de calidad (punto de vista de calidad) del enfoque multimodelo de [Gonzalez-Huerta, 2012] con requisitos no funcionales con variabilidades. Los requisitos no funcionales se identifican observando las relaciones

entre los atributos de cualidad y luego son relacionados a los atributos de calidad y son medidos (de ser posible). El modelo no presenta una notación visual para los requisitos no funcionales con variabilidades ni tampoco presenta mapeo de variabilidades a otra notación como modelo de features, lo cual (al ser una notación textual) hace más difícil la validación con el cliente. Además, no se tienen en cuenta los ítems 4 y 6 del listado anterior.

El objetivo de este trabajo es la definición de una notación NFR con variabilidades (meta número 1 de la sección 1.4 del capítulo de Introducción). Como metas de esta sección (sub-metas de la meta número 1 de Introducción) se desea: que la notación cumpla con los asuntos 1 a 6 del listado anterior, tanto en el ámbito de aplicaciones web como de otros tipos de sistemas; que la notación permita modelar variabilidades de manera visual; y que contemple variabilidades en todos los niveles de abstracción.

En este trabajo hemos extendido el framework NFR (que posee notación visual) por medio de agregado y modificación de elementos para expresar variabilidad, los cuales también pueden ser representados visualmente.

3.3.2. Trabajo relacionado

Como trabajo relacionado hemos seleccionado a los enfoques que tratan con atributos de calidad o requisitos no funcionales, utilizan una notación de atributos de calidad o NFR para definir variabilidad y tienen en cuenta de alguna manera los aspectos de los ítems 1 hasta 6 de la sección 3.3.1 de Background.

Con las condiciones antes mencionadas, el único trabajo que cumple con las mismas es el trabajo de [Gonzalez-Huerta, 2012b]. En este enfoque se extiende el meta-modelo de punto de vista de calidad del enfoque multimodelo que integra atributos de calidad en MDE (Model Driven Engineering) de [Gonzalez-Huerta, 2012] con elementos para modelar requisitos no funcionales. Los requisitos no funcionales son identificados inspeccionando los atributos de calidad del modelo y son relacionados a los mismos. Los requisitos no funcionales son descritos a través de lenguaje natural o en OCL y cada uno de ellos incluye una medida (fórmula descrita en lenguaje natural). Los requisitos no funcionales pueden estar relacionados a features o componentes funcionales. En este enfoque se trabaja principalmente con atributos de calidad y los requisitos no funcionales son especificados a través de fórmulas que deben ser respetadas. Como limitaciones de este trabajo se pueden mencionar: que, al ser una notación textual, puede presentar más dificultad para validación con el cliente y, además, no tiene en cuenta de manera óptima, los ítems 4 y 6 del listado de la sección 3.3.1 de Background.

3.3.3. Notación propuesta para Framework NFR con variabilidades

En este trabajo se propone realizar una extensión al *framework NFR* de [Chung, 2000] que permite modelar variabilidades. Para extender el framework NFR con variabilidades se realizaron dos agregados al meta-modelo: una clase para el elemento *Variability* y dos tipos de descomposiciones al elemento de tipo enumerado *DecompKind, orVariability* y *andVariability* (ver figura 3.24). El elemento *Variability* tiene un nombre (a través del atributo *name*) y una cardinalidad de variabilidad, dada por medio de dos números enteros, los atributos *min* y *max*. Una *operacionalización* o un *softgoal NFR* puede participar en una o más variabilidades.

Se definieron las siguientes restricciones de variabilidad:

- Un *softgoal NFR* o una *operacionalización* no puede tener dos variabilidades con el mismo nombre. A continuación se expresa esta restricción en lenguaje OCL:

```
Context NFRSoftgoal inv: self.variability->size() >1 implies
```

```
self.variability->forall(v1, v2 | v1.name <>v2.name)
```

- Si dos elementos, ya sea un *softgoal NFR* u *operacionalización* tienen una variabilidad en común entonces ellos deben ser “hijos” del mismo padre. A continuación se expresa esta restricción en lenguaje OCL:

```
Context NFRSoftgoal inv: NFRSoftgoal.allInstances()->forall(n1, n2 |
```

```
n1.variability->size() >0 and n2.variability >0
```

```
and n1.variability->exists(v1 | n2.variability->exists(v2 |
```

```
v1.name = v2.name)) implies n1.oclaAsType(Elements).contribution[parent] =
```

```

and n2.oclAsType(Elements).contribution[parent] and
n1.oclAsType(Elements).contribution[parent].ocIsTypeOf(decompositionCT))
Context Operationalizing inv: Operationalizing.allInstances()->

forall(o1, o2 | o1.variability->size() >0 and o2.variability >0 and
o1.variability->exists(v1 | o2.variability->exists(v2 | v1.name = v2.name)) implies
o1.oclAsType(Elements).contribution[parent] = and
o2.oclAsType(Elements).contribution[parent] and
o1.oclAsType(Elements).contribution[parent].ocIsTypeOf(decompositionCT))

```

No es posible definir una variabilidad que incluya un subconjunto de softgoals de una descomposición *AND/OR* porque la semántica de esta descomposición dice que todos los posibles softgoals de una descomposición *AND* u *OR* deben estar presente en cualquier miembro de la familia que satisface la descomposición. Por esta razón, en este trabajo definimos dos nuevos tipos de descomposición para permitir variabilidades:

- ***ANDVariability***: es una descomposición cuyos componentes pueden estar en una o más variabilidades y todos los softgoals seleccionados de acuerdo a tales variabilidades son necesarios para satisfacer el padre de la descomposición.
- ***ORVariability***: es una descomposición cuyos componentes pueden estar en una o más variabilidades y al menos uno de los softgoals seleccionados de acuerdo a tales variabilidades es necesario para satisfacer el padre de la descomposición.

Consideraciones de modelado. Una variabilidad se representa con una anotación de *UML* que incluye los softgoals variantes. El elemento *ANDVariability* es representado de la misma manera que una descomposición *AND* pero con el agregado de la anotación de variabilidad conectado a cada descomposición variante. El elemento *ORVariability* es representado de la misma manera que una descomposición *OR* pero con el agregado de de la anotación de variabilidad conectado a cada descomposición variante. Se llama opcional al softgoal *X* que solo participa como variante en una variabilidad *V*, la cual posee cardinalidad $[0..1]$ y tiene como único variante al softgoal *X*. Se representa un softgoal opcional incluyendo una signo de pregunta en la nube del softgoal. Si los elementos de una variabilidad con cardinalidad $a..b$ incluyen todos los miembros de un *ANDVariability* u *ORVariability*, entonces se omite la anotación de variabilidad y se incluye “[a..]” junto a la descomposición.

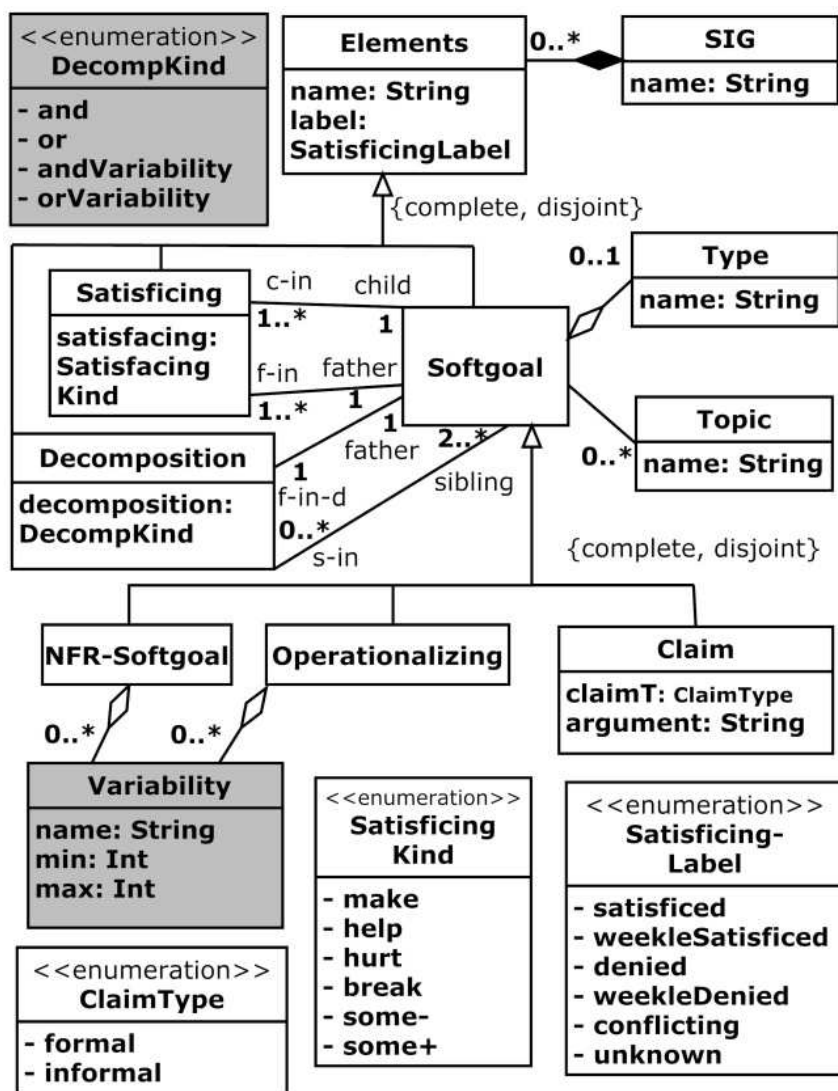


Figura 3.24: Extensión al meta-modelo del framework NFR para modelar variabilidades.

Ejemplo. En la figura 3.25 se incluye un *SIG* con variabilidades para el requisito no funcional *Accesibilidad*. Debido a que el objetivo en este trabajo es mostrar como se extienden los modelos *SIG* para familia de aplicaciones, en este ejemplo solo se descomponen ciertas partes. El softgoal *Accesibilidad* se descompone usando descomposición del tipo *andVariability* en *Visual*, *Audtivia* y *Cognitiva*. Suponemos que las aplicaciones van a ser desarrolladas solo para un tipo de problemas de accesibilidad, por lo tanto, estos softgoals se encuentran en una variabilidad con cardinalidad 1..3, lo cual significa que uno de los tres tipos de problemas de accesibilidad debe ser seleccionado en etapa de especialización.

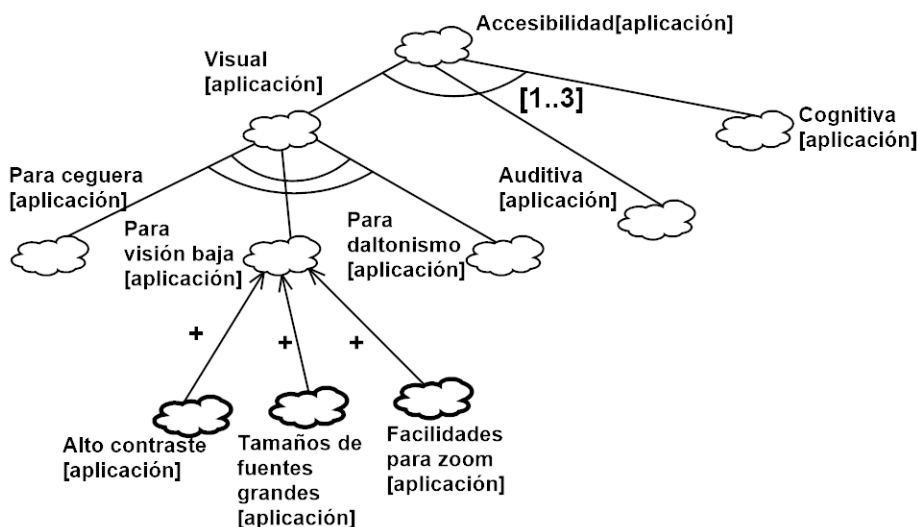


Figura 3.25: Parte de SIG con variabilidades para el requisito no funcional de Accesibilidad.

3.3.4. Configuración del SIG de dominio

El proceso de configuración de SIG de dominio trata sobre la selección y eliminación de variabilidades de un SIG con variabilidades para así obtener un SIG para una aplicación en particular. Una vez configurado el SIG, analizando los requisitos brindados por el cliente para la aplicación, puede ser necesario actualizar e incorporarle o modificar nuevos elementos al mismo. Si bien por razones de tiempo y volumen de trabajo, en este trabajo no nos concentramos en brindar un proceso de desarrollo de familias de aplicaciones web que incluya a los SIG del framework NFR, los mismos pueden ser observados como ayuda para tomar decisiones en cuanto a requisitos funcionales.

Partiendo del SIG con variabilidades de la familia de aplicaciones elaborado utilizando la notación definida en el capítulo 3.3, entre el analista y los actores involucrados en el desarrollo de la aplicación deben decidir, observando en detalle los requisitos no funcionales de la aplicación, los elementos que van a ser parte del SIG de la aplicación. Para llevar a cabo la especialización de un SIG con variabilidades a un SIG de una aplicación hemos definido un procedimiento que se detalla a continuación.

3.3.5. Procedimiento de configuración de SIG de dominio

Se denota con $DG(S)$ al grafo formado por un softgoal S y sus descendientes (contribuciones y otros softgoals) en un SIG. Se denota con $DGO(S)$ al conjunto obtenido de remover de $DG(S)$ todos los $DG(T)$, tales que T es un descendiente de S y T que contribuye a un softgoal fuera de $DG(S)$.

Se obtiene el SIG de una aplicación aplicando sistemáticamente (hasta que el SIG no incluya ninguna variabilidad) las siguientes reglas (no es necesario que sean aplicadas en orden):

1. Si un softgoal S es opcional y S no ha sido seleccionado (se descarta a S para la aplicación a especializar) entonces se debe eliminar $DGO(S)$ y las relaciones de S con su padres.
2. Si una variabilidad de tipo ANDVariability con padre P incluye una variabilidad V , entonces se debe remover V . Por cada variante M de V que no sido seleccionado para la aplicación se debe remover $DGO(M)$ y la relación entre P y M . Los variantes seleccionados serán los descendientes de P y se relacionan a través de una descomposición AND.
3. Si una variabilidad de tipo ORVariability con padre P incluye una variabilidad V , entonces se debe remover V . Por cada variante M de V que no sido seleccionado para la aplicación se debe remover $DGO(M)$ y la relación entre P y M . Los variantes seleccionados serán los descendientes de P y se relacionan a través de una descomposición OR.

4. Si V es una variabilidad con sus variantes contribuyendo en algún grado a un softgoal padre P , entonces se debe remover V . Por cada variante M de V que no sido seleccionado para la aplicación se debe remover $DGO(M)$ y la relación entre P y M .

Ejemplo. En la figura 3.26 se brinda el SIG con variabilidades para el requisito no funcional *Usabilidad* para el subsistema OPAC del caso de estudio de familia de aplicaciones de biblioteca online (se realizan algunas operaciones a partir de búsquedas de ítems). El softgoal raíz del SIG se denomina *Usability* y se descompone utilizando una descomposición *andVariability* en 3 softgoals llamados *Errors*, *Efficiency* y *Learnability*. Estos softgoals están presentes dentro de una variabilidad con cardinalidad 2..3 (al menos dos de los tres softgoals deben ser seleccionados en la etapa de especialización). Las operacionalizaciones *Use search history* (almacenar y visualizar historial de la búsquedas realizadas), *Show related data* (mostrar ítems relacionados al ítem visualizado) y *Use reading history* (almacenar y visualizar historial de los ítems visualizados) que contribuyen al softgoal *Efficiency* son opcionales, por lo tanto pueden ser seleccionadas o no. La operacionalización obligatoria *Use help* se descompone usando descomposición del tipo *orVariability* en las operacionalizaciones *General index of help*, *Tips of help* y *Agent help*. *General index of help* y *Agent help* son softgoals variantes incluidos en una variabilidad con cardinalidad 1..1 (esto significa que uno de los dos debe ser seleccionado en tiempo de especialización). La operacionalización opcional *Suggest fields* también contribuye al softgoal *Use help*.

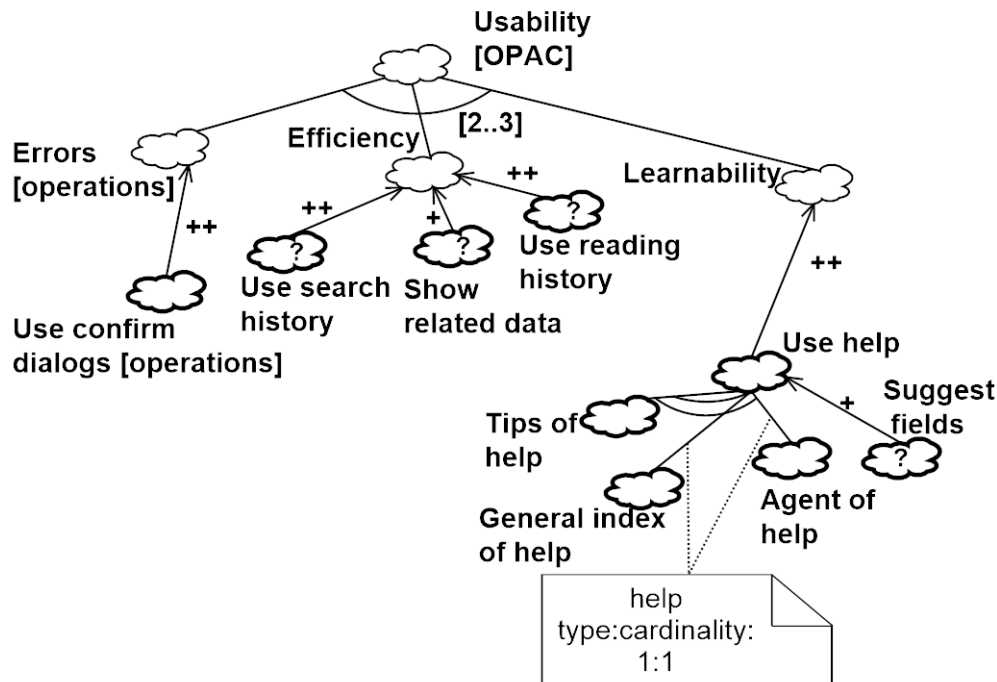


Figura 3.26: SIG con variabilidades para Usabilidad en subsistema OPAC del Sistema de Administración de Bibliotecas Online.

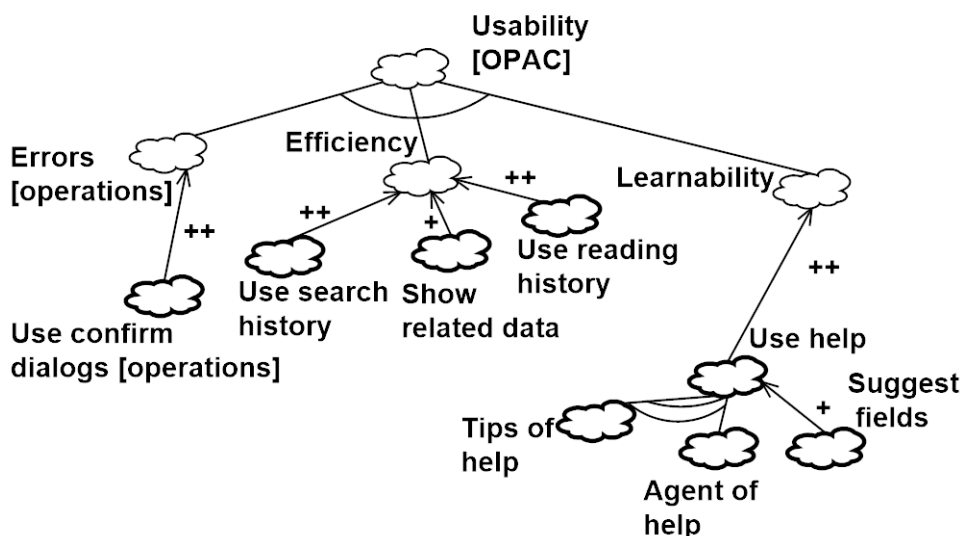


Figura 3.27: SIG especializado para Usabilidad en subsistema OPAC del Sistema de Administración de Bibliotecas Online.

En la figura 3.27 se muestra el resultado de la aplicación de las reglas del procedimiento presentado en este capítulo para la obtención de una posible especialización del SIG con variabilidades de la figura 3.26. En primer lugar se resuelve la variabilidad del tipo *andVariability* que involucra a los softgoals *Errors*, *Efficiency* y *Learnability* y que tiene cardinalidad 2..3. Aplicando la segunda de las reglas del procedimiento se decide seleccionar los 3 softgoals. Aplicando la tercera regla del procedimiento se resuelve la variabilidad de tipo *orVariability* que involucra a las operacionalizaciones *General index of help* y *Agent help* con cardinalidad 1..1, donde se resuelve elegir a la operacionalización *Agent help*. Aplicando la primera de las reglas del procedimiento se resuelve seleccionar todas operacionalizaciones opcionales incluidas en el diagrama, las cuales pasan a ser operacionalizaciones de la aplicación, estas son *Use search history*, *Show related data*, *Use reading history* y *Suggest fields*.

3.3.6. Evaluación

Para analizar la propuesta de requisitos no funcionales con variabilidades se utilizó el enfoque *<pregunta, resultado basado en una métrica>*.

La primer pregunta planteada es: *¿Es realmente interesante considerar variabilidades en el ámbito de NFR?* Sí, esto lo muestra que en los ejemplos y aplicación al caso de estudio realizados, de un total de 6 descomposiciones, 5 de ellas incluyeron variabilidades.

La segunda pregunta planteada es: *¿Las variabilidades en NFR tienden a darse siempre en el mismo nivel de abstracción?* No, de los ejemplos realizados de un total de 5 variabilidades, 3 se dieron a nivel de descomposiciones de NFRs y 2 a nivel de operacionalizaciones; lo cual comprueba que permitir la definición de variabilidades en cualquier nivel de abstracción de un SIG NFR es de utilidad.

3.4. Modelos de features

3.4.1. Background

Los modelos de features son una notación para capturar requisitos variantes y compartidos de una línea de productos por medio de la representación de una estructura jerárquica de features. En general, son utilizados para: especificar los resultados del análisis de dominio, validar los requisitos de una línea de productos con el cliente, guiar el desarrollo líneas de productos o componentes de las mismas y proveer una base para realizar la configuración de productos de una línea de productos.

Los modelos de features surgieron en el año 1990 con FODA (Feature-oriented domain analysis, ver [Kang, 1990]), y a partir de allí, se convirtieron en la notación más difundida, utilizada y aceptada

para modelar características de líneas de productos o familia de aplicaciones. Estos modelos de features se representaban por medio de una estructura de tipo árbol que incluían un elemento concept (raíz con el nombre del sistema) y elementos features; que representaban un aspecto prominente o distintivo visible al usuario, una cualidad o una característica de un sistema de software o conjunto de sistemas. Las fetures se relacionaban jerárquicamente a través de relaciones de composición. Estas relaciones entre features indicaban que una feature hija requería que su feature padre esté presente. Los tipos de relaciones de composición eran AND (que indicaba que cuando la feature padre está presente, todas las hijas tienen que estar presentes) y XOR (que indica que sólo uno de los hijos debe estar presente si el padre está presente).

A finales de los 90's, aparecieron algunos trabajos (por ejemplo [Kang, 1998] y [Griss, 1998]) que extendieron la notación de features de FODA. Entre las incorporaciones más importantes y trascendentes, se destacan: los nombres de las features aparecen en cajas; los modelos de features no son necesariamente árboles, pueden ser grafos acíclicos dirigidos; se agrega el operador de composición OR (que indica que al menos uno de los hijos debe estar presente si el padre está presente); se le da una representación gráfica a las restricciones de dependencia, a través de flechas con líneas de punto entre features dependientes.

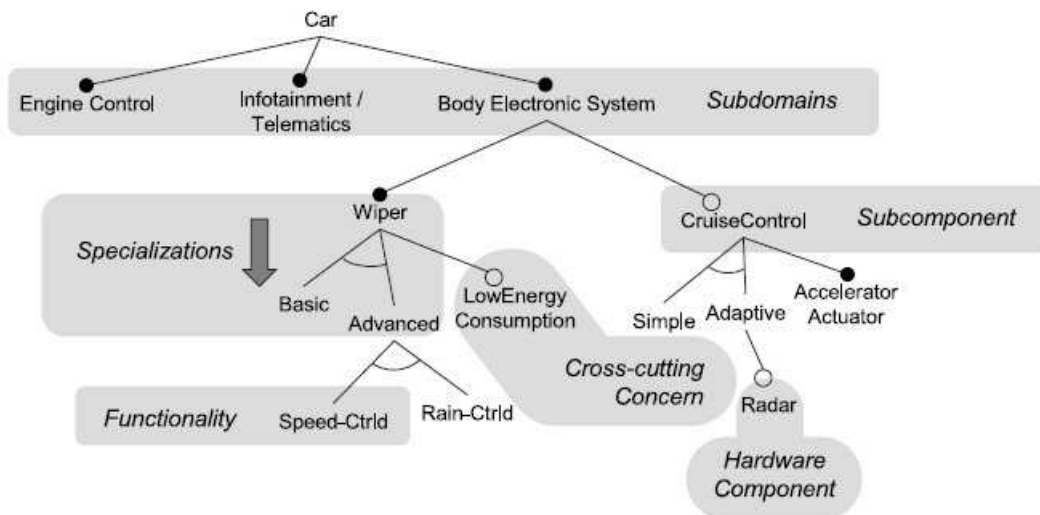


Figura 3.28: Ejemplo de diagrama de features para una familia de automóviles utilizando relaciones tradicionales.

A partir del año 2000, las principales extensiones que se realizaron para modelo de features tuvieron que ver con la manera de descomponer features padres en features hijas (ver [Riebisch, 2002]). Estas extensiones incorporaron nuevos tipos de descomposiciones (alternative, optional, mandatory), el uso de cardinalidades y agrupamiento de features con cardinalidades para representar descomposiciones.

En los últimos años surgieron nuevas extensiones a los modelos de features, pero estas en general se basan en necesidades específicas de un dominio o ámbito de aplicación y no han sido ampliamente utilizadas. Algunas de las extensiones surgidas son: posibilidad de que una feature hija cuente más de una feature padre, distintas relaciones de descomposición de features, uso de atributos para features, representación de distintos niveles de features, otros tipos de relaciones de dependencia (por ejemplo help y suggest).

En este trabajo utilizamos modelos de features basados solo en relaciones de descomposición y cardinalidades. Las relaciones de descomposición pueden ser de dos tipos: *SingleRelation* (relaciona una feature padre con una feature hija con una cardinalidad $r..1$, donde r puede ser 0 o 1) y *GroupRelation* (relaciona una feature padre con un conjunto de features hijas a través de una relación $n..m$, donde m debe ser mayor que 1 y representa el número de features hijas y n es la cantidad de features que deben ser seleccionadas). Además, ya que el concepto de feature ha ido variando con el correr del tiempo, hemos decidido utilizar el concepto de feature de [Pohl, 2005], donde se dice que “una feature es una característica o rasgo que una instancia de producto individual de una línea de productos puede poseer o

no”.

En la siguiente sección se presenta una notación definida para modelos de features, de acuerdo a necesidades propias de este trabajo y a mejoras de notaciones existentes.

3.4.2. Notación propuesta para modelo de features

En este trabajo hemos definido una notación propia para diagramas de features (un diagrama de features es una representación gráfica de un modelo de features), basada mayormente en notaciones existentes. Ya que una de las principales metas de este trabajo es la construcción automática de un modelo de features (a través de transformación de modelos) a partir de los modelos de requisitos de dominio en *UML*, fue necesario construir un meta-modelo particular para modelos de features que facilite las transformaciones a realizar, de acuerdo a la notación de variabilidades para requisitos de dominio en *UML*. Para dicho fin se tomaron como referencia los meta-modelos de los trabajos de [Kapova, 2009] y [Bragança, 2007]. En general se optó por la estructura y relaciones entre features del meta-modelo de [Kapova, 2009], con el agregado del concepto de referencias de [Bragança, 2007]. Además se agregó el concepto de feature parámetro para representar variabilidades de diagramas de actividad del tipo *ParameterSet*. A continuación se detallan las características del meta-modelo definido para diagramas de features, y luego, en la figura 3.29 se puede observar el meta-modelo de features definido en este trabajo:

- Se hace distinción de feature raíz.
- Las variabilidades son manejadas con relaciones que incluyen atributos de cardinalidad (llamados *minCardinality* y *maxCardinality*) y con features parámetro.
- Una relación de composición puede ser una relación de grupo, cuando *maxCardinality* > 1, lo cual significa que hay más de un variante en la variabilidad; o una relación simple, cuando *maxCardinality* = 1, lo cual significa que hay un único variante en la variabilidad. La omisión de cardinalidad significa que hay una relación simple con cardinalidad 1..1 entre la feature padre y la feature hija y se dice que la feature hija es *obligatoria*.
- A diferencia de otros modelos de features no se hace distinción entre feature opcional y feature obligatoria, esto se maneja con relaciones entre features (con atributos de cardinalidad) y con features parámetro.
- Una relación utilizada para representar una variabilidad puede incluir un nombre para expresar el significado de la variabilidad. El motivo de esto es que, en algunas ocasiones, a partir de los nombres de los hijos de la relación no queda muy en claro el significado de la variabilidad.
- Una feature parámetro representa una feature que tiene un conjunto de valores posibles (atributo *valueSet*), a partir de los cuales un subconjunto de valores deben ser seleccionados utilizando cardinalidades a través de los atributos *minCardinality* y *maxCardinality*.
- Se permite que una feature pueda tener más de un padre. El motivo para esto es la traducción de las relaciones *extend* e *include* de casos de uso a modelos de features.
- Se tienen en cuenta dependencias entre features. Las mismas pueden ser del tipo *requires* o *excludes*. Estas dependencias son denotadas con una flecha de líneas de puntos entre la feature dependiente y las otras features. Para etiquetar el tipo de variabilidad se adjunta una etiqueta a la flecha con los posibles valores 'R' (para *Requires*) y 'E' (para *Excludes*).
- Ya que una feature puede solo ser derivada (a través de relaciones de composición) en otras features, para permitir que una feature pueda participar en varias relaciones de composición se introdujo el concepto de referencia (elemento *Reference*) a una feature. Las referencias son denotadas con una línea doble entre la feature padre y la feature referenciada hija.
- Los elementos *Node* y *Relation* son meta-clases abstractas.

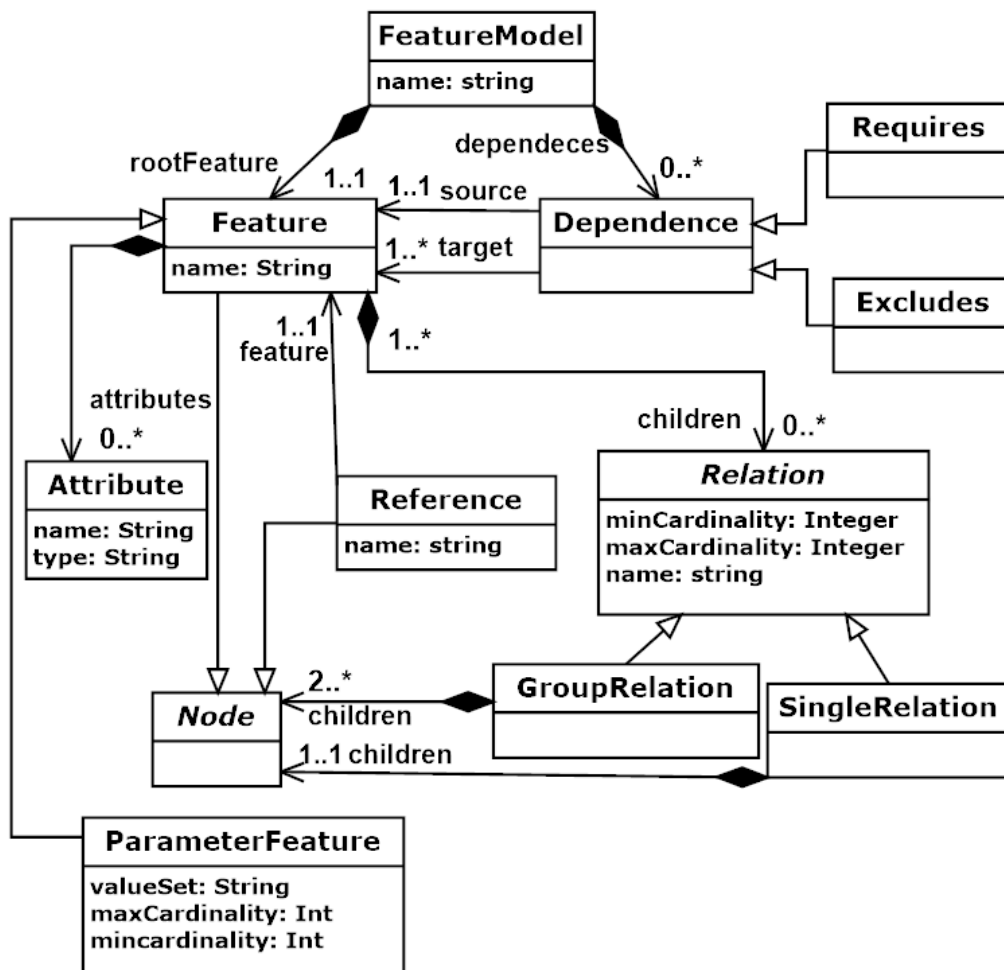


Figura 3.29: Meta-modelo de features.

Ejemplo. La figura 3.30 muestra una parte (incluyendo las features de más alto nivel) del caso de estudio de Sistema de Administración de Bibliotecas Online. Este modelo incluye una relación de grupo con $\text{minCardinality}=1$ y $\text{maxCardinality}=1$ entre la feature *OPAC* y las features *Public Search* y *Private Search* features. Esto quiere decir que de las dos posibilidades (*Public Search* y *Private Search*) hay que seleccionar una en la definición de una aplicación.

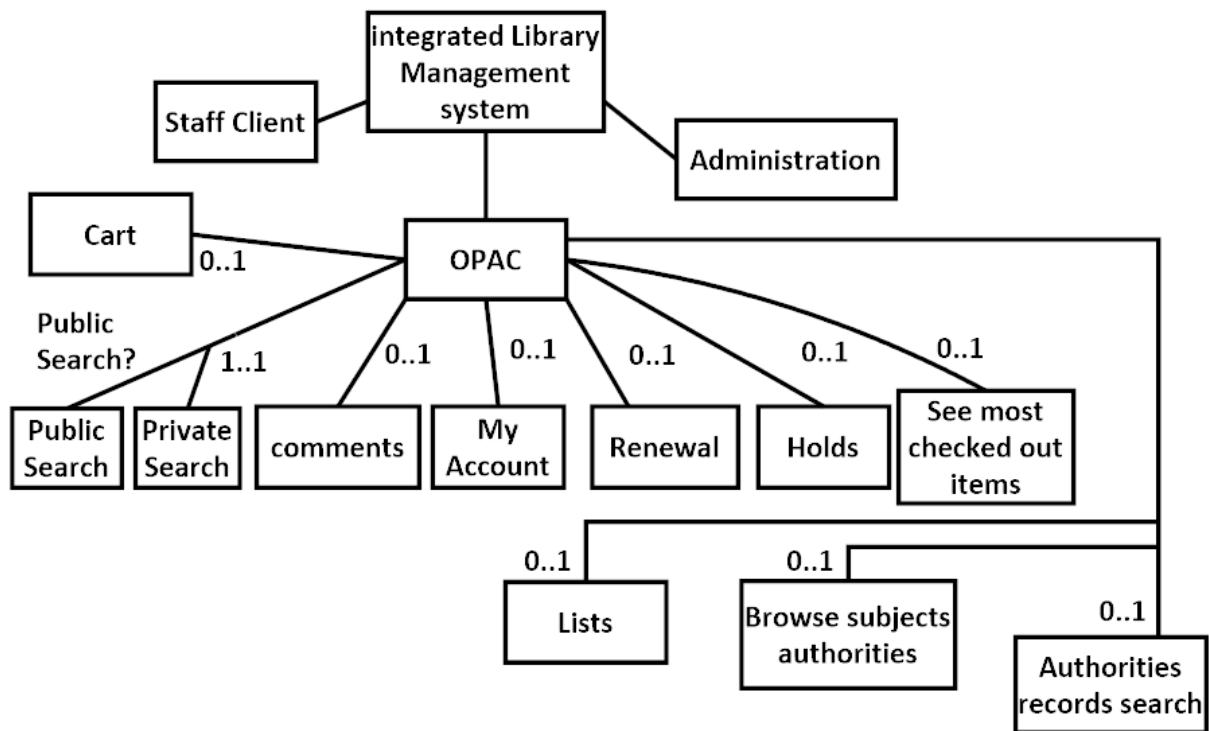


Figura 3.30: Diagrama de features para para el caso de estudio Sistema de Administración de Bibliotecas Online, incluyendo las features de más alto nivel del subsistema OPAC.

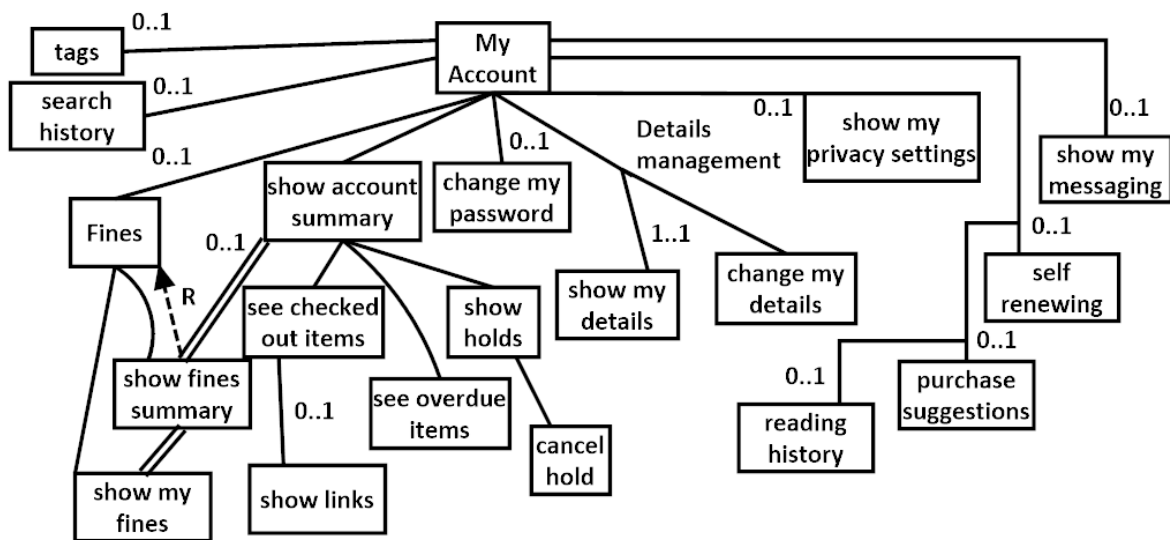


Figura 3.31: Diagrama de features para los niveles más altos del paquete My account perteneciente al subsistema OPAC del caso de estudio Sistema de Administración de Bibliotecas Online.

La figura 3.31 muestra una parte (incluyendo las features de más alto nivel) del paquete de casos de uso *My account* (para manejo de cuentas de usuario de la biblioteca) del caso de estudio de Sistema de Administración de Bibliotecas Online. Entre los aspectos más destacables de este ejemplo se incluye una relación simple entre la feature *My Account* y la feature *change my password*, con $\text{minCardinality}=0$ and $\text{maxCardinality}=1$, lo cual quiere decir que la funcionalidad *change my password* es opcional y puede ser seleccionada o no, dependiendo de la aplicación. También se puede observar que la feature *show my fines*

feature está referenciada por la *feature Show fines summary*, haciendo uso de una relación simple. Esto es necesario ya que la *feature show fines summary* ya estaba relacionada a la *feature Fines*. Además se puede observar una restricción de dependencia del tipo Requires entre la *feature show fines summary* y la *feature Fines*.

3.5. Modelos de interfaz de usuario abstracta con variabilidades

3.5.1. Background

La importancia de modelar aspectos de interfaz de usuario haciendo uso de los beneficios de SPL para aplicaciones web es un asunto que ha sido evidenciado claramente en la bibliografía. En [Lee, 2006] se dice que “*no está claro cómo modelar features variantes para problemas específicos de la interfaz de usuario en líneas de productos de software basadas en la web. Esta es una barrera importante que se debe superar si el desarrollo de líneas de productos de software de productos basadas en la web aprovecha las mayores ventajas de la reutilización de software: mayor calidad, menor esfuerzo o menor tiempo de comercialización.*”

Distintos enfoques en la bibliografía se han ocupado de modelar variabilidades de interfaz de usuario en familia de aplicaciones web (ver [Pleuss, 2010], [Gabillon, 2013], [Gabillon, 2015], [Sottet, 2015] y [Fadhilillah, 2018]); pero, en general, en estos enfoques se propone definir *features* variantes en modelos de *features* para interfaz de usuario y mapeo de esas *features* a modelos abstractos propios de interfaz de usuario. Estos trabajos no tienen en cuenta variabilidad en modelos específicos de interfaz de usuario (ya sean concretos o abstractos), lo cual representa un problema debido a que los especialistas en interfaz de usuario no están acostumbrados a construir modelos de *features*. Además, el proceso de identificación de variabilidades de interfaz de usuario en base a requisitos se puede ver favorecido si se trabaja con modelos específicos de interfaz de usuario, ya que se piensa en términos de elementos de interfaz de usuario y no en términos de *features*.

Hemos hallado un solo trabajo en la bibliografía que presenta notación de interfaz de usuario con posibilidad de incluir variabilidades (ver [Lee, 2006]). Los problemas de este trabajo son que no tiene en cuenta en la definición de su notación de interfaz de usuario los últimos adelantos tecnológicos (widgets RIA, frameworks responsivos) y la expresividad para representar variabilidades es muy escasa (solo incluye para este fin relaciones entre clases de tipo OR/XOR y no se incluyen restricciones de dependencias entre variantes).

En base a los problemas detectados en la bibliografía, el objetivo de este trabajo es producir una notación de interfaz de usuario abstracta de aplicaciones web con variabilidades. Como metas para este objetivo se desea: que la notación de interfaz de usuario abstracta de aplicaciones web tenga en cuenta los últimos adelantos tecnológicos y que la expresividad para denotar variabilidades sea adecuada. Este objetivo permite alcanzar la meta número 6 de la sección 1.4 del capítulo de Introducción.

En este trabajo hemos extendido la notación para interfaz de usuario abstracta para aplicaciones web definida en la sección 2.2.2 por medio de agregados para expresar variabilidad, los cuales también pueden ser representados visualmente.

3.5.2. Trabajo relacionado

Como trabajo relacionado hemos seleccionado a los enfoques que presentan variabilidad en modelos específicos de interfaz de usuario, ya sean modelos abstractos o modelos concretos.

En base a este criterio de selección, el único trabajo que cumple con esto es el de [Lee, 2006]. En el enfoque presentado en [Lee, 2006] se define una notación llamada WUIML que extiende el meta-modelo de UML 2.0 con estereotipos de clases para modelar interfaz de usuario de aplicaciones web, estereotipos de dependencias para modelar interfaz de usuario de aplicaciones web y clases del meta-modelo de diagramas de actividad para modelar acciones variantes y estereotipos para modelar. Al ser un enfoque anterior a avances tecnológicos significativos (por ejemplo aparición de HTML5, aplicaciones RIA, frameworks responsivos), en la notación para modelar interfaz de usuario de aplicaciones web no se tiene en cuenta algunos widgets y elementos que fueron surgiendo como consecuencia de dichos avances tecnológicos. Otro problema a destacar de este enfoque es su escasa expresividad para representar variabilidades, solo a través de relaciones entre clases de tipo OR/XOR y sin brindar la posibilidad de definir restricciones de dependencia entre variantes. En este trabajo la variabilidad se puede expresar haciendo uso de

cardinalidades y elementos de interfaz de usuario abstracta variantes.

3.5.3. Una propuesta de notación para interfaz de usuario abstracta con variabilidades

Para construir interfaces de usuario abstractas, en este trabajo se utilizan modelos que respetan el meta-modelo *WAAUID* definido en la sección 2.2.2. Por la manera en que fue diseñado este meta-modelo, los modelos de interfaces de usuario son construidos a través de abstracciones de contenedores, los cuales se componen (utilizando relaciones de composición) de otras abstracciones de contenedores o bien de abstracciones de elementos de interfaz de usuario (listas, elementos de entrada, formularios, menús, entre otros). Esta forma de construir modelos de interfaz de usuario abstracta brinda la posibilidad de definir variabilidades en distintos niveles de interfaz de usuario (haciendo uso de estas relaciones de composición).

Para la definición de variabilidades en interfaz abstracta de usuario abstracta se definió un meta-modelo (ver figura 3.32) que, de manera análoga al concepto de anotaciones de variabilidad definido en requisitos, incluye una meta-clase llamada *VariabilityUI*. Esta clase representa un elemento de variabilidad de interfaz de usuario abstracta e incluye un conjunto de uno o más variantes (un elemento de interfaz de usuario abstracta de cualquier tipo), esto se representa gráficamente adjuntando una anotación de UML de variabilidad al diagrama, con su correspondiente nombre, cardinalidad y asociaciones a los elementos de interfaz de usuario abstracta que forman parte de los variantes.

Además se incluyen clases para representar restricciones de dependencia entre variantes, estas restricciones pueden ser del tipo *Requires* o del tipo *Excludes*. La restricción de dependencia de tipo *Requires* significa que un variante origen (relación *dependent*, elemento de interfaz de usuario abstracta) debe ser seleccionado en una proceso de selección de variantes de variabilidades si y solo si el variante destino (relación *dependenceOn*) es seleccionado. La restricción de dependencia de tipo *Excludes* significa que un variante origen (relación *dependent*, elemento de interfaz de usuario abstracta) debe ser seleccionado en una proceso de selección de variantes de variabilidades si y solo si el variante destino (relación *dependenceOn*) no es seleccionado.

A continuación se presenta el meta-modelo definido para describir variabilidades en modelos de interfaz de usuario abstracta:

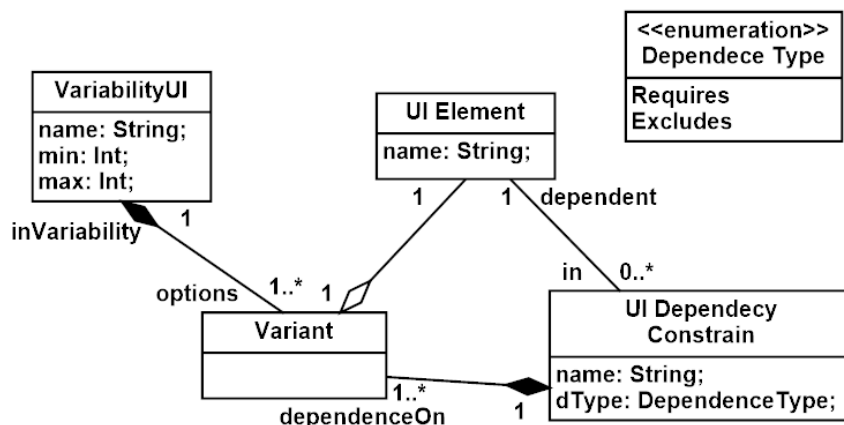


Figura 3.32: Meta-modelo para anotaciones de variabilidad en modelado abstracto de interfaz de usuario abstracta.

3.5.4. Algunas situaciones genéricas de variabilidad en interfaz de usuario identificadas

De acuerdo a situaciones de interfaz de usuario identificadas en el caso de estudio y al análisis de interfaces de usuario en distintas aplicaciones web que pueden formar parte de una familia de aplicaciones, hemos identificado algunos tipos de variabilidades genéricas comunes. Si bien este apartado no pretende categorizar todas las posibles situaciones de variabilidades en interfaces de usuario, presenta algunas

situaciones que se pueden presentar con frecuencia en interfaces de usuario de familia de aplicaciones web y que motivan aún más la notación definida en esta sección.

A continuación se detallan las situaciones genéricas identificadas como posibles variabilidades en interfaces de usuario:

- **Elementos de interfaz de usuario opcionales:** se hace referencia a distintos elementos de interfaz de usuario abstracta que pueden estar presentes o no en construcciones de interfaces de usuario abstractas, dependiendo si son necesarios o no para determinada aplicación. Pueden ser contenedores conteniendo elementos de interfaz o simplemente elementos de interfaz que no son contenedores. Se modelan con una variabilidad opcional con cardinalidad 0..1.

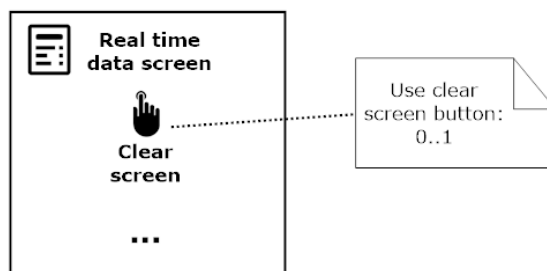


Figura 3.33: Variabilidad opcional en modelado abstracto de interfaz de usuario abstracta: botón para limpiar pantalla.

La figura 3.33 incluye un ejemplo que presenta una funcionalidad de interfaz de usuario opcional. Dicha funcionalidad limpia la pantalla de datos actuales para una mejor visualización por parte del usuario en una aplicación que continuamente está visualizando nuevos datos en tiempo real (como por ejemplo un chat). Esta funcionalidad estará habilitada o no en una aplicación particular si se selecciona la variabilidad asociada al elemento de tipo **Button** *Clear screen* en la aplicación que se esté realizando la configuración. Esto significa que el elemento de tipo **Button** estará disponible solo en aquellas aplicaciones en las que se haya seleccionada la variabilidad.

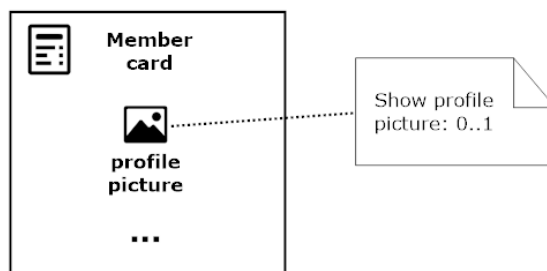


Figura 3.34: Variabilidad opcional en modelado abstracto de interfaz de usuario abstracta: foto de perfil opcional.

La figura 3.34 incluye otro ejemplo de variabilidad de interfaz de usuario opcional, que en este caso presenta un elemento del tipo **Image** opcional, que representa la imagen de perfil de una tarjeta de miembro de una aplicación. Algunas aplicaciones por razones de carga podrían optar por no incluir una imagen de perfil.

- **Variabilidad en patrones de interfaz de usuario:** este tipo de variabilidad hace referencia a la presentación de elementos de interfaz de usuario para los cuales es posible aplicar uno o más patrones de interfaz de usuario definidos sobre ellos. Se puede modelar con una variabilidad de selección de variantes entre varios. Algunos ejemplos de patrones comunes sobre elementos pueden ser: paginación o scroll continuo sobre listas de ítems; menú vertical dropdown, menú horizontal dropdown o menú de tipo acordeón; conjunto de imágenes en formato de galería o conjunto de imágenes en formato de diapositivas.

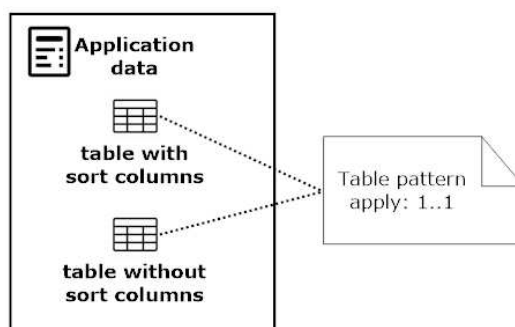


Figura 3.35: Variabilidad en patrones de interfaz de usuario: tablas con distintos patrones.

La figura 3.35 muestra un elemento contenedor llamado *Application data* en donde se visualiza información de la aplicación a través de una tabla. Este contenedor incluye dos elementos de tipo table, *table with sort columns* y *table without sort columns*. Ambos elementos son variantes de la variabilidad de cardinalidad 1..1 *Table pattern apply*, la cual significa que se debe elegir entre uno de los dos elementos de tipo table, uno con el uso del patrón que ordena los datos de las columnas y otro sin el uso de dicho patrón.

- **Variabilidad en construcciones visuales de ítems o records:** este tipo de variabilidad de interfaz de usuario trata la posibilidad de mostrar un ítem o record de datos del sistema de distintas maneras de acuerdo a distintas necesidades de las aplicaciones. No se hace referencia a variabilidad en los datos almacenados por la aplicación sino en la manera en que éstos son presentados al usuario. Por ejemplo para algunas aplicaciones se puede optar por incluir un ícono en un record de información y para otras no.

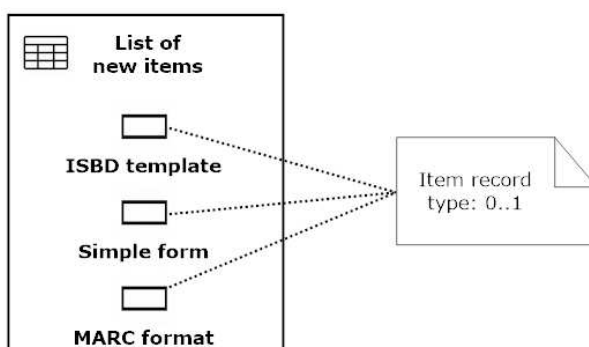


Figura 3.36: Ejemplo de variabilidad de alternativas de tipos de records en ítems.

La figura 3.36 muestra un ejemplo de variabilidad en la construcción de records para visualizar nuevos ítems de datos de una aplicación biblioteca. El elemento de tipo table *List of new items* muestra ítems de datos de una biblioteca respetando un formato de record que puede ser de tres tipos distintos, dependiendo de la necesidad de la aplicación a configurar. Esta posibilidad de selección determina el nivel de detalle bibliográfico que se visualizará en la página de detalle de un ítem. El record del tipo *simple form* muestra el ítem en un formato gráfico, el record de tipo *MARC format* muestra un ítem respetando el formato *MARC21* y el record del tipo *ISBD template* muestra un ítem que respeta el formato *ISBD (International Standard Bibliographic Description, AACR2)*. A través de la variabilidad *Item record type* con cardinalidad 1..1 se debe seleccionar uno de los tres formatos posibles de record que mostrará la aplicación.

- **Variabilidad en distintas composiciones de interfaz de usuario:** este tipo de variabilidad de interfaz de usuario se trata de seleccionar entre varias posibilidades de elementos de interfaz de usuario abstracta para un mismo fin. Pueden ser composiciones a través de contenedores o elementos

simples. Si bien existen numerosas posibilidades de este tipo de variabilidades, algunos ejemplos de este tipo de variabilidad pueden ser utilizar un botón o un áncora para un enlace determinado o utilizar un menú o una composición de enlaces y botones.

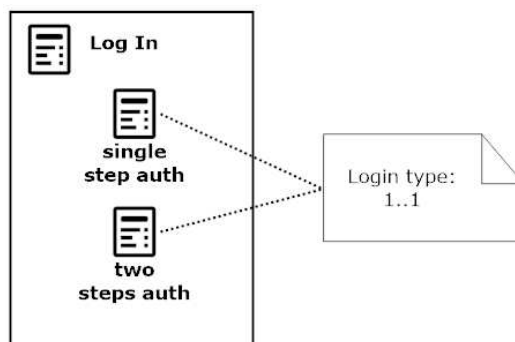


Figura 3.37: Ejemplo de variabilidad de selección de distintas composiciones de interfaz de usuario abstracta.

La figura 3.37 muestra un ejemplo de variabilidad de distintas posibilidades de composiciones de interfaz de usuario abstracta. El ejemplo muestra un contenedor llamado *Log In* que incluye dos contenedores con distintas posibilidades para que el usuario se autentique en la aplicación a través de una cuenta de usuario. Para este fin se brindan dos posibilidades, la primera ingresando nombre de usuario y contraseña en una misma pantalla y la segunda ingresando en primer lugar nombre de usuario para luego ingresar el password en otra pantalla distinta. Esta variabilidad no hace referencia a métodos de ingreso a la aplicación, sino a cómo ingresar los datos por parte del usuario, por lo cual es una variabilidad de interfaz de usuario. Para modelar esta situación se utiliza la variabilidad *Login type* con cardinalidad 1..1 que incluye dos contenedores, uno por cada tipo de posibilidad. El contenedor *Two steps* a su vez incluye dos contenedores para modelar las dos pantallas secuenciales en las cuales se va a ingresar el nombre de usuario y la contraseña. El contenedor *Single step* incluye otro contenedor con los elementos necesarios para modelar el ingreso del nombre de usuario y la contraseña en una misma pantalla.

- Variabilidad en visualización de una información de medida en distintos formatos:** existen ítems de información que representan mediciones en algún formato, que en base a una fórmula estática de conversión, puede ser trasladados a otro formato. No todas las aplicaciones pueden necesitar que se visualice el ítem en más de un formato, dependiendo del origen del público visitante, de cuestiones de layout u otros aspectos. Algunos ejemplos típicos de este tipo de variabilidades son visualizar la temperatura en grados Celsius, en grados Fahrenheit u ambos; visualizar mediciones de altura en metros, centímetros o en pies.

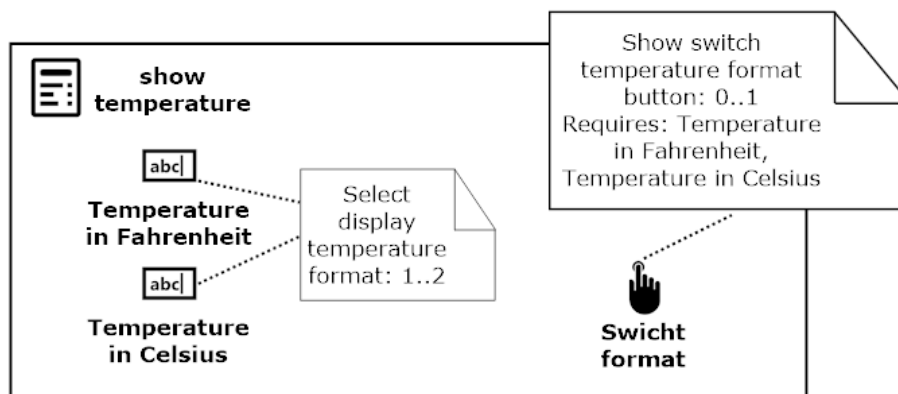


Figura 3.38: Ejemplo de variabilidad en visualización de una información de medida en distintos formatos: formatos de temperatura.

La figura 3.38 muestra un ejemplo de variabilidad en visualización de una información de medida en distintos formatos para el caso de una aplicación que permite mostrar la temperatura de un determinado lugar en dos formatos: grados Celsius y grados Fahrenheit. Se incluyen dos elementos de tipo Text de solo lectura, uno para cada formato, los cuales son variantes de la variabilidad *Select display temperature*; que tiene una cardinalidad 1..2 y que significa que al menos uno de los dos variantes debe ser seleccionado. También se incluye un elemento de tipo Button opcional llamado *Switch format* que permite visualizar un formato a la vez en pantalla y que es opcional; es decir, es variante de la variabilidad *Show switch temperature format* de cardinalidad 0..1, que incluye una referencia de tipo Requires a los dos elementos de tipo de tipo Text correspondientes a los formatos de visualización. Esto significa que para seleccionar dicha variabilidad opcional es necesario que se hayan seleccionado ambos tipos de formatos en la variabilidad *Select display temperature*.

Evaluación

Para analizar la propuesta de modelador de interfaz de usuario abstracta con variabilidades se utilizó el enfoque *<pregunta, resultado basado en una métrica>*.

La pregunta planteada es: *¿Es realmente interesante considerar variabilidades en el ámbito de interfaz de usuario abstracta?* Sí, para el caso de estudio realizado en este trabajo se encontraron cuatro variabilidades opcionales propias de interfaz de usuario. Además, en la sección 3.5.4, se identificaron cinco situaciones comunes en interfaces de usuario, con sus respectivos ejemplos, que conviene que sean modeladas con variabilidades.

La siguiente pregunta planteada es: *¿Es necesario considerar referencias entre variantes en interfaz de usuario abstracta?* Sí, el ejemplo de la figura 3.38 incluye una situación de una aplicación web donde un variante requiere que dos variantes sean seleccionados. Este ejemplo puede replicarse en situaciones similares de interfaces de usuario.

Parte II

Desarrollo de modelos de dominio de familias de aplicaciones web y generación de modelo de features

Capítulo 4

Proceso de desarrollo de modelos de dominio y generación de modelo de features

4.1. Background

Para el desarrollo de familia de aplicaciones, generando automáticamente modelo de features, existen dos enfoques principales: desarrollo basado en productos variantes, donde se cuenta con una buena cantidad de productos sobre un mismo dominio y a partir de ellos se extraen variabilidades y características en común para constituir una familia de aplicaciones ([Mefteh, 2015], [Alves, 2008] y [Weston, 2009]); y desarrollo de familia de aplicaciones desde cero, donde partiendo de requisitos y de una categoría de aplicaciones bien definida se especifican y detallan las variabilidades y características en común ([Bragança, 2007] y [Casalánguida, 2012]).

A la hora de seguir un proceso de desarrollo para familia de aplicaciones, la primera inquietud que surge es ¿qué enfoque seguir y bajo qué condiciones se debe seleccionar uno de los dos enfoques?

Creemos que a partir cualquiera de los dos tipos de enfoques se pueden obtener buenos resultados, pero dependiendo de la naturaleza del problema a desarrollar puede ser más conveniente un enfoque a otro. A continuación listaremos las condiciones que creemos más favorables para desarrollo a partir de productos variantes:

- Se cuenta con la implementación de numerosos productos de software variantes similares utilizando técnicas de reutilización ad-hoc tales como copiar-pegar-modificar [Ziadi, 2012]. Al contar con esto y utilizando dichas técnicas (o la conocida técnica de clone and own), nuevos productos pueden ser generados sin grandes costos en cuestiones de tiempo y dinero.
- Una compañía no puede afrontar un desarrollo de una línea de productos de software desde cero y, por lo tanto, utiliza elementos existentes (código fuente, artefactos de diseño) [Beuche, 2009]. Esto es debido a que desarrollar una SPL desde cero es una tarea altamente costosa ya que implica el desarrollo de los artefactos de software de dominio [Clements, 2002].
- No se cuenta con expertos en desarrollo de artefactos de dominio. Para el desarrollo de modelos de dominio desde cero se requiere pericia [Mefteh, 2015], conocimiento y experiencia [Wang, 2009].
- El desarrollo será realizado por una organización pequeña. En [Al-Msie'Deen, 2014], se dice que en la práctica, la mayoría de las organizaciones pequeñas no desarrollan una línea de productos de software desde cero.
- Ya se cuenta con una línea de productos de software y se desea extender la misma en base a nuevos productos. En este caso, utilizando alguna técnica como clone and own, se puede crear una nueva SPL que incluya los nuevos productos. En [Zhang, 2012] se habla de esta técnica.

A continuación se listan las condiciones que creemos más favorables para desarrollo desde cero de familias de aplicaciones:

- Se cuenta con un software exitoso y de gran escala, que requiere de múltiples configuraciones para su implementación. En [Bosch, 2011] se da la noción de que un producto exitoso puede ampliarse a una familia de productos. En este caso, se pueden definir los requisitos de la familia de aplicaciones realizando ingeniería inversa.
- Se cuenta con un desarrollo de productos open source de gran escala de manera colaborativa, donde a solicitud de usuarios y desarrolladores se van incluyendo funcionalidades, y muchas de ellas, opcionales. En [Linden, 2008] se presenta un trabajo en el cual en base a adaptaciones de un software open source se construye infraestructuras de líneas de productos para tales sistemas.
- Se cuenta con un especialista de dominio con vasta experiencia en elaboración de artefactos de dominios. Como se menciona en [Wang, 2009] para desarrollar artefactos de dominio es necesario contar con analistas con conocimiento y experiencia en dominio.
- El desarrollo será realizado por una organización de mediana o gran escala. Ya que el desarrollo desde cero es considerado una tarea costosa [Clements, 2002] y que, en la práctica, no es adoptada por las organizaciones pequeñas [Al-Msie'Deen, 2014], es un tipo de desarrollo más adecuado para organizaciones con más recursos.
- Ya se cuenta con una línea de productos de software y se desea extender la misma en base a nuevos requisitos. En este caso, solo es necesario extender o modificar los artefactos de dominio en base a los nuevos requisitos para generar una nueva línea de productos de software.

Ya que este trabajo ha sido pensado principalmente para que pueda ser utilizado por personas que vienen desarrollando con modelos de UML en las etapas de requisitos (a partir de documentos de requisitos), sin la necesidad de conocimientos avanzados en modelos de features y modelos con variabilidades; y, además, hemos encontrado algunas situaciones interesantes de estudio (cómo generar un modelo de features de aceptable calidad sin necesidad de especialistas, cómo darle un marco de línea de productos a aplicaciones open source de gran volumen -como el caso de estudio- y cómo definir un proceso que contemple evolución y cambios en familia de aplicaciones) hemos optado por el enfoque de desarrollo desde cero.

Además de los modelos de dominio de la familia de aplicaciones en UML (ya sean generados desde cero o a partir de productos variantes) es necesario tener un modelo de features de la familia de aplicaciones, ya que los modelos de features son orientados al reutilizador y los modelos de UML son orientados al usuario y, así se tenga toda la información de variabilidad en los modelos de UML, es necesario el modelo de features ya que los clientes suelen no entender modelos de UML (ver [Casalánguida, 2012]).

En este trabajo hemos decidido utilizar un enfoque que, a partir de modelos de dominio con variabilidades, genera automáticamente el modelo de features de la familia de aplicaciones por los siguientes motivos: *a.* en [Wang, 2009] se dice que la calidad de construir un modelo features depende fuertemente de los conocimientos y experiencia de la persona que ocupa el rol de analista en el dominio, por lo tanto, será valioso si existe un enfoque que pueda minimizar la participación de los analistas en la construcción del modelo de features, al automatizar actividades claves en la construcción de un modelo de features. Por este argumento, aquí buscamos minimizar o, hasta eliminar, la participación del rol de analista en la construcción del modelo de features de la familia de aplicaciones; *b.* mantener la consistencia entre los modelos de dominio y el modelo de features de la familia de aplicaciones.

Otro asunto importante en cualquier tipo de desarrollo, más aún cuando se cuenta con varios tipos de modelos y para el ámbito de familia de aplicaciones, es la elaboración de un proceso detallado de desarrollo. Para este trabajo, es necesario definir un proceso de desarrollo donde se visualicen las etapas necesarias para el desarrollo de los modelos de dominio y la generación del modelo de features de la familia de aplicaciones. Este proceso debería incluir todas las actividades que se deben realizar a lo largo del proceso y los artefactos como resultado de esas actividades.

Hemos hallado en la bibliografía algunos trabajos de familia de aplicaciones que utilizan el enfoque de desarrollo desde cero con modelos de UML con variabilidades en la etapa de requisitos y generación automática de modelo de features (ver [Bragança, 2007] y [Bragança, 2008]). En [Bragança, 2007] se presenta un proceso que tiene en cuenta modelo de caso con variabilidades, la generación automática de modelo de features y la posterior configuración de un producto de la familia. En [Bragança, 2008] se presenta un proceso global que incluye los artefactos producidos en cada etapa del ciclo de vida de desarrollo de aplicaciones y se presenta un capítulo para generar modelo de features a partir de casos de uso con variabilidades. En general en los procesos presentados en estos trabajos (salvo [Bragança, 2008])

no se presenta un proceso de desarrollo con suficiente nivel de detalle, no se tiene en cuenta evolución en el marco de proceso de desarrollo, no se brinda un ámbito de aplicabilidad del proceso ni se consideran modelos de interfaz de usuario con variabilidades para generar modelos de features.

Se pretende que el proceso a definir para desarrollo de modelos de dominio de familia de aplicaciones y generación progresiva de modelo de features satisfaga los siguientes requisitos: 1. Genere el modelo de features a partir de todos los tipos de modelos de dominio definidos; 2. Tenga en cuenta evolución y cambios en los requisitos de familia de aplicaciones. En [Benlarabi, 2015] se dice que los requisitos de los clientes evolucionan continuamente. Por lo tanto, la SPL debe ser adaptada para hacer frente a nuevas necesidades de los clientes; 3. Considere modelo de interfaz de usuario abstracto con variabilidades. Esto brinda la posibilidad que el modelo de features generado incluya distintos niveles de features con consideraciones de requisitos (a través de casos de uso), requisitos detallados (a través de diagramas de actividad) y de interfaz de usuario (modelo de interfaz de usuario abstracto). No hemos hallado otro trabajo que tenga en cuenta interfaz de usuario para generar automáticamente modelo de features y consideramos que esto es importante porque las interfaces de usuario actuales cada vez brindan más posibilidades de desarrollo a través de widgets e implementaciones distintas para un mismo fin (por ejemplo con frameworks responsivos); 4. Incluya suficiente detalle de cada actividad del proceso global donde se brinde cada artefacto producido y utilizado en la misma. Se busca que al leer el proceso queden totalmente claras las etapas y artefactos a producir. 5. Brinde un ámbito de familias de aplicaciones donde sea adecuado utilizar este proceso.

El objetivo principal de esta parte del trabajo es producir un proceso de desarrollo de modelos de dominio que utilice Model Driven Development para generar un modelo de features de manera automática a partir de modelos de dominio de requisitos en UML y de modelo de dominio de interfaz de usuario abstracta. Como metas para este objetivo, se plantea que el proceso de desarrollo satisfaga los requisitos planteados en el párrafo anterior. Este objetivo contribuye a alcanzar las metas número 3 y 4 de la sección 1.4 del capítulo de Introducción.

En este capítulo se brinda una descripción detallada del proceso desarrollo de modelos de requisitos y diseño de familia de aplicaciones web y de las actividades que se llevan a cabo durante el proceso, y además, se brinda un listado de desarrollos open source que pueden ser considerados como familia de aplicaciones.

4.2. Trabajo relacionado

Se tuvieron en cuenta como trabajos relacionados a los enfoques que incluyen generación automática de modelo de features de una familia de aplicaciones (sin necesidad de revisión manual del modelo de features) a partir de modelos de dominio de requisitos y diseño, en particular que al menos incluyan modelos de casos de uso de UML con variabilidades para generar modelos de features.

	Modelos de dominio contemplados	Más de una manera de definir variabilidades	Generación de FM contempla todos los modelos de dominio	Proceso con suficiente detalle	Proceso contempla evolución de la familia	Se brinda ámbito de aplicabilidad
[Bragança, 2007]	Casos de uso	No	Sí	No	No	No
[Bragança, 2008]	Casos de uso / Diagramas de actividad/ Modelos de clases	Sí, una por modelo	No	Si	No	No
Este trabajo	Casos de uso/ Diagramas de actividad/ Modelo de Ui abstracto	No	Sí	Sí	Sí	Sí

Figura 4.1: Comparación entre enfoques seleccionados para procesos de generación de modelos de features a partir de modelos de dominio..

En la tabla comparativa de la figura 4.1 se incluyen los trabajos de [Bragança, 2007], [Bragança, 2008] y el trabajo de aquí. En cuanto al primer ítem de la tabla *Modelos de dominio contemplados*, hace referencia a modelos de dominio con variabilidades incluidos en el proceso de definición de dominio de familia de aplicaciones. El trabajo de [Bragança, 2007] solo contempla modelos de casos de uso. El trabajo de [Bragança, 2008], además de casos uso incluye diagramas de actividades y modelos de clases. En este trabajo se contemplan casos de uso, diagramas de actividad y modelos de interfaz de usuario abstracta (a través de modelos de clases). El segundo ítem de la tabla, *Más de una manera de definir variabilidades*, hace referencia a si cada modelo de dominio tiene su propia notación de variabilidades, lo cual hace más dificultoso para el diseñador de dominio aprender ya que tiene que aprender más de una notación de variabilidades. Solo en el trabajo de [Bragança, 2008] se requiere utilizar una notación de variabilidad por cada tipo de modelo de dominio que se define. El tercer ítem de la tabla, *Generación de FM contempla todos los modelos de dominio*, tiene que ver si el modelo de features es generado utilizando como entrada todos los modelos de dominio definidos; solo el trabajo [Bragança, 2008] no genera automáticamente el modelo de features utilizando todos los modelos de dominio, debido a que solo tiene en cuenta diagrama de casos de uso con variabilidades para generar el modelo de features. El siguiente ítem de la tabla, *Proceso con suficiente detalle*, trata sobre si los procesos brindados en esos trabajos cuentan con el suficiente detalle que se debería considerar en un proceso completo; es decir, explicitando artefactos, etapas y descripción de cada etapa; solo [Bragança, 2008] y el trabajo de aquí cumplen afirmativamente con este ítem. El siguiente ítem de la tabla, *Proceso contempla evolución de la familia*, habla acerca de si la definición del proceso de dominio de familia de aplicaciones es construido de una manera que favorece a la evolución de la familia, proveyendo iteraciones sobre el dominio; solo el trabajo presentado aquí muestra esa consideración en la definición de su proceso. El último ítem de la tabla, *Se brinda ámbito de aplicabilidad*, tiene que ver con que si el proceso de producción del dominio de familia de aplicaciones está delimitado a un ámbito de tipos de familias de aplicaciones, lo cual hace más fácil distinguir qué tipos de familia de aplicaciones son más óptimas para ser desarrolladas con el proceso; solo el trabajo presentado aquí presenta esta consideración.

4.3. Descripción del proceso

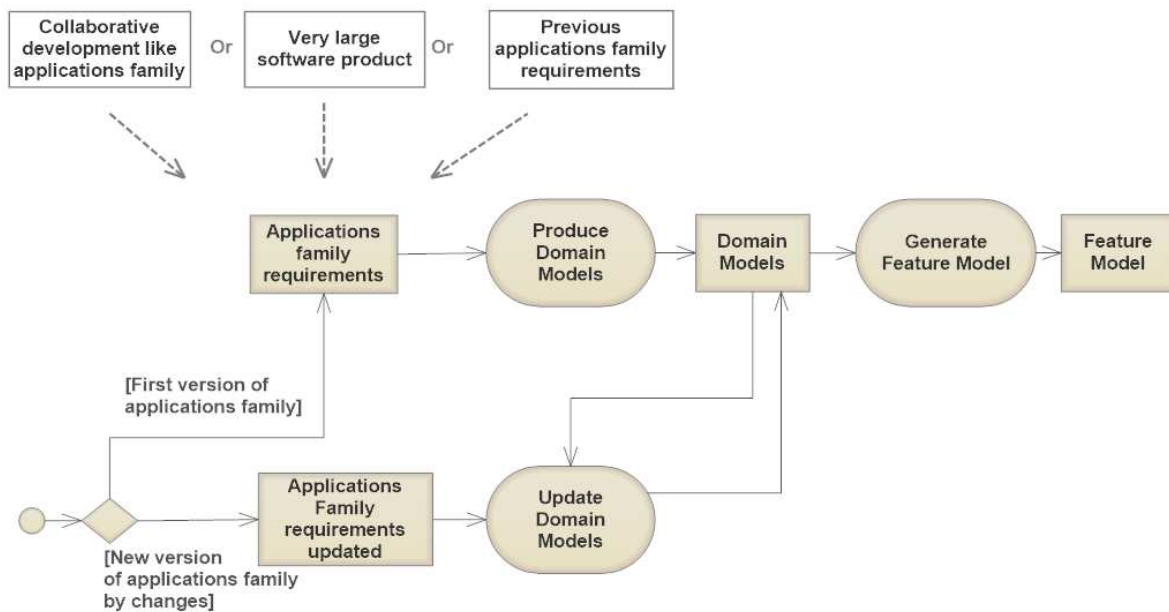


Figura 4.2: Proceso de desarrollo de modelos de dominio y feaures de familias de aplicaciones web.

En la figura 4.2 se muestran las tres actividades principales del proceso (representadas con óvalos), las cuales hacen uso o producen como resultado artefactos (rectángulos que representan nodos de objeto en el diagrama). Estas actividades son las que van guiando el proceso de desarrollo de modelos de dominio de familias de aplicaciones web y generación del modelo de features.

El inicio del proceso presenta dos alternativas: desarrollar la primera versión de la familia de aplicaciones o desarrollar una nueva versión de la familia de aplicaciones.

La primera alternativa, de construir la primer versión de los modelos de dominio de la familia de aplicaciones, es la más compleja y la que será desarrollada en detalle a lo largo de los siguientes capítulos y en resumen en la siguiente sección. Esta primer versión parte de un documento de requisitos de la familia de aplicaciones que puede ser elaborado o bien haciendo ingeniería inversa de algún sistema de software exitoso de gran escala o a partir de algún desarrollo colaborativo considerado como una familia de aplicaciones, o bien a partir de versiones de requisitos de familias de aplicaciones previas. En base al documento de requisitos de la familia de aplicaciones, la primer actividad involucra la producción de modelos de dominio de requisitos y diseño, para luego, en la segunda actividad realizar la generación automática del modelo de features de la familia de aplicaciones a partir de los modelos de dominio producidos.

La segunda alternativa, de construir una nueva versión de los modelos de dominio de la familia de aplicaciones, parte de un documento de requisitos actualizado, toma como entrada los modelos de dominio previamente construidos e incluye la actividad de actualizar los modelos de dominio de la familia de aplicaciones en base a los documentos de requisitos actualizados, que da como resultado modelos de dominio actualizados. Esta actividad es desarrollada en una sección posterior de este capítulo.

4.3.1. Desarrollo de modelos de dominio para familias de aplicaciones

Esta actividad está incluida en la región superior del diagrama de actividades de la figura 4.2 (alternativa de construir primera versión de la familia de aplicaciones) y tiene como objetivo desarrollar los modelos de dominio de la familia de aplicaciones, utilizando para ello las notaciones con variabilidades de diagramas de casos de uso, diagramas de actividad y modelo de interfaz de usuario abstracta.

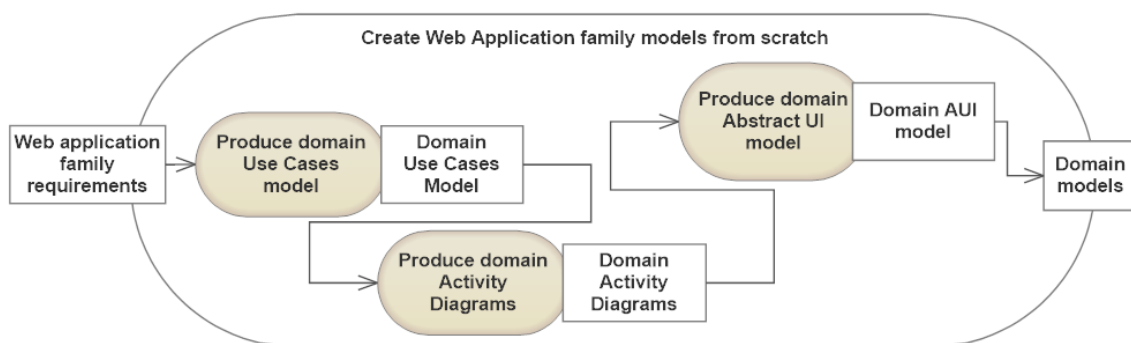


Figura 4.3: Proceso de producción de modelos de dominio de familia de aplicaciones web desde cero.

Como ya explicamos anteriormente, en este trabajo hemos optado por construir los modelos de dominio desde cero. Como se puede apreciar en la figura 4.3 el proceso de producción de modelos de dominio desde cero toma como elementos de entrada los requisitos de la familia de aplicaciones web (funcionales, no funcionales y de interfaz de usuario) e incluye las siguientes actividades secuenciales: *producción de diagramas de casos de uso dominio* (utilizando la propuesta de notación definida en la sección 3.1.3); *producción de diagramas de actividad de dominio* (los requisitos funcionales y los diagramas de casos de uso con variabilidades y utilizando la propuesta de notación definida en la sección 3.2.3); y, finalmente, *producción de modelo de interfaz de usuario abstracta de dominio* (utilizando la propuesta de notación definida en la sección 3.5.3 y teniendo en cuenta las descripciones de casos de uso a través de diagramas de actividades con variabilidades y los requisitos de interfaz de usuario). Como resultado de esta actividad se obtienen los modelos de dominio de la familia de aplicaciones web, que comprende todos los modelos citados anteriormente.

4.3.2. Generación de modelo de features a partir de modelos de dominio

Generación de modelo de features a partir de modelos de dominio de requisitos.

La generación automática del modelo de features de la familia de aplicaciones web se realiza de manera incremental y de acuerdo al orden en el que los modelos son desarrollados. En primer lugar se obtiene el modelo de features para los requisitos de la familia de aplicaciones web y, en segundo lugar, el modelo de features definitivo de la familia de aplicaciones web, teniendo en consideración los modelos de interfaz de usuario abstracta con variabilidades.

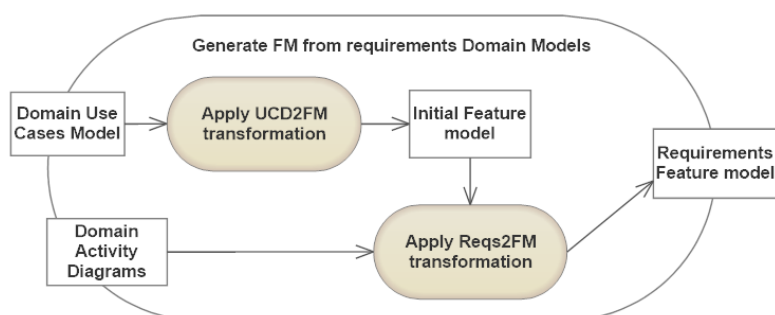


Figura 4.4: Actividad para generar modelo de features a partir de modelos de dominio de requisitos.

Como se puede observar en la figura 4.4, esta actividad se descompone en otras dos actividades:

1. *Aplicar la transformación UCD2FM*: esta actividad (*Apply UCD2FM transformation* en el diagrama) toma como entrada el modelo de casos de uso de dominio y en base a la ejecución (utilizando MDA) de la transformación llamada *UCD2FM*, que incluye una serie de reglas de transformación en lenguaje *ATL* genera como resultado el modelo de features inicial de la familia de aplicaciones web, el cual incluye mapeos a features de elementos provenientes de diagramas de casos de uso con variabilidades.

2. *Aplicar la transformación Reqs2FM*: esta actividad (*Apply Reqs2FM transformation* en el diagrama) recibe como entradas el modelo de features inicial de la familia de aplicaciones web y los diagramas de actividad de dominio que describen a los casos de uso de dominio. Aplicando la transformación MDA llamada *Reqs2FM* y en base a un conjunto de reglas de transformación en lenguaje *ATL* se genera como resultado el modelo de features de requisitos de la familia de aplicaciones web.

Generación de modelo de features a partir de modelos de interfaz de usuario abstracta de dominio.

La última tarea para obtener el modelo de features final de la línea de productos es la de aplicar la transformación llamada *UI2FM*.

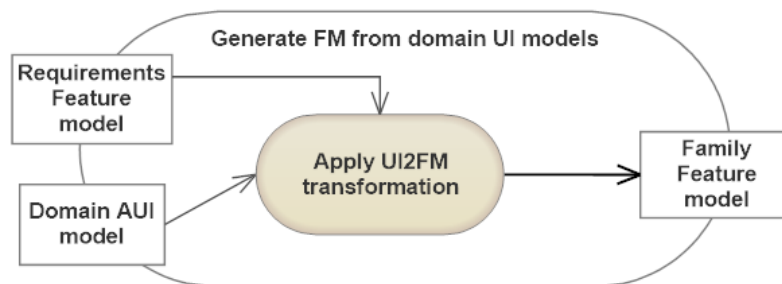


Figura 4.5: Actividad para generar modelo de features a partir de modelos de interfaz de usuario abstracta con variabilidades.

Como se puede apreciar en la figura 4.5, esta transformación toma como entrada el modelo de interfaz de usuario de dominio para cada caso de uso y el modelo de features de requisitos, y en base a un conjunto de reglas de transformación genera nuevas features correspondientes a las variabilidades especificadas en los modelos de interfaz de usuario de dominio y las adjunta (de acuerdo a las features correspondientes a cada caso de uso) en el modelo features de requisitos, dando como resultado un nuevo modelo de features, denominado modelo de features de la familia de aplicaciones web. Este modelo de features será el utilizado para realizar las configuraciones necesarias para especificar los modelos de casos de uso, descripciones de casos de uso y de interfaz de usuario abstracta de las aplicaciones.

4.3.3. Actualización de modelos de dominio

Si se cuenta con alguna versión de los modelos de dominio producida y en consecuencia con su modelo de features correspondiente generado (todo esto conocido como infraestructura reutilizable), existen diversos factores que pueden hacer necesario actualizar los requisitos de la familia de aplicaciones. A continuación, listaremos algunos

- Validación del modelo de features con el cliente. El cliente puede sugerir cambios de acuerdo a sus necesidades, o bien sus necesidades pueden variar a lo largo del tiempo.
- Cambios tecnológicos. La tecnología evoluciona constantemente y la familia de aplicaciones puede necesitar cambios para mantenerse al corriente del avance tecnológico.
- Surgimiento de un producto del dominio de aplicaciones de la familia que considera características adicionales a las que se consideraban en la familia.
- Si los requisitos fueron creados a través de ingeniería inversa de un sistema colaborativo de un software de gran escala, usuarios o desarrolladores pueden realizar actualizaciones al sistema por medio de issues (correcciones) y requests (peticiones).

Estos cambios deben ser reflejados en una nueva versión de los requisitos de la familia de aplicaciones que será elaborada por el analista de la familia. Con un nuevo documento de requisitos de la familia de aplicaciones, los modelos de dominio y el modelo de feature de la familia quedará desactualizado, por lo tanto, es necesario llevar a cabo una actualización de los modelos de dominio de la familia de aplicaciones,

para luego generar el modelo de features nuevamente.

Según se dice en [Ripon, 2013], diseñar, desarrollar y mantener un buen sistema de software es una tarea difícil, aún en este siglo 21. El enfoque de reutilizar soluciones buenas existentes para desarrollar cualquier aplicación nueva es, hoy en día, uno de los focos centrales de los ingenieros de software. Construir sistemas de software desde componentes desarrollados previamente brinda un ahorro de tiempo, trabajo y mejora la mantenibilidad. El paradigma de línea de productos de software se basa en estos principios de reutilización, pero para atender el asunto de mantenibilidad de la línea de productos es necesario tener en cuenta algunas consideraciones. Por ejemplo, actualizar los modelos de dominio colabora fuertemente en mantener la mantenibilidad de la línea de productos de software.

Independientemente de los modelos de dominio a actualizar (casos de uso con variabilidades, diagramas de actividad con variabilidades o modelos de interfaz de usuario abstracta con variabilidades), en este trabajo la manera de definir las variabilidades es la misma, por lo cual las consideraciones de actualización van a ser las mismas. A continuación brindaremos consideraciones generales a tener en cuenta a la hora de actualizar modelos de dominio:

- Tratar elementos de dominio obsoletos. Algunos requisitos pueden eliminar funcionalidades de la familia de aplicaciones, esto involucra:
 - Remover elementos obsoletos o relaciones entre elementos.
 - Remover variabilidades enteras con sus hijos y restricciones de dependencia relacionadas a esas variabilidades.
 - Aplicar remociones en cascada como consecuencia de las remociones anteriores.
- Tratar modificaciones a modelos. Esto involucra:
 - Modificar cardinalidades. Por ejemplo, puede ser necesario hacer elementos mandatorios, opcionales o viceversa; cambiar la cardinalidad de variabilidades.
 - Modificar nombres de elementos o variabilidades. El punto anterior o algún cambio en los requisitos, puede dar lugar a la necesidad de cambiar nombres de elementos o variabilidades.
 - Modificar comportamientos de elementos o relaciones entre elementos. Puede ser necesario cambiar el flujo de comportamiento de algún elemento que incluya comportamiento o, bien, modificar relaciones entre distintos elementos.
- Tratar agregados a modelos. Esto involucra:
 - Agregar nuevos elementos o relaciones entre elementos.
 - Agregar nuevas variabilidades o restricciones de dependencias entre variabilidades.
- Chequear consistencia e integridad de los modelos de dominio, de acuerdo a los cambios realizados en los puntos anteriores. Esto involucra:
 - Cerciorar que las variabilidades incluyan al menos un elemento.
 - Cerciorar que las variabilidades incluyan la cantidad de elementos que denotan sus cotas mínimas y máximas.
 - Para casos de uso, cerciorar que las relaciones extend, asociación e incluye se respetan debidamente.
 - Para diagramas de actividad, cerciorar que no queden elementos inconexos o aislados.
 - Chequear que si se agrega un caso de uso, se agrega también su comportamiento por medio de diagrama de actividad y la interfaz de usuario abstracta con variabilidades del mismo.

Aplicando las consideraciones anteriores y, en base a las versiones actualizadas de los modelos de dominio de la línea de productos, se puede generar nuevamente el modelo de features de la línea de productos aplicando consecutivamente las transformaciones UCD2FM, Reqs2FM y UI2FM.

4.4. Desarrollos Open Source que pueden ser considerados como familias de aplicaciones

En la actualidad existe una gran cantidad de desarrollos open source de aplicaciones de gran escala que pueden ser considerados familia de aplicaciones por su poder de configuración e instanciación a través de funcionalidades, módulos o paquetes seleccionables; que permiten crear aplicaciones bastante diferentes entre sí, de acuerdo a necesidades particulares. Este tipo de desarrollo brinda la posibilidad de realizar ingeniería inversa y a partir del software, realizar una elicitación de requisitos de la familia de aplicaciones. Se puede crear un modelo de dominio para este tipo de aplicaciones, que luego irá siendo modificado de manera incremental y colaborativa, ya que en su gran mayoría, estas familias de aplicaciones de nutren de que distintos desarrolladores puedan aportar funcionalidades y agregados, por iniciativa propia o por requisitoria de los usuarios. El repositorio *gitHub* es la herramienta más popular para este tipo de desarrollo.

A continuación, recopilamos algunos ejemplos de estos tipos de desarrollo que pueden ser considerados como familia de aplicaciones (además del caso de estudio realizado en este trabajo basado en el sistema Koha):

- *Evergreen*: es un software para bibliotecas altamente escalable, algunas de sus funcionalidades posibles son: ayudar a buscar material de bibliotecas, manejar catálogos, circulación de material de biblioteca y otros tipos de tareas complejas que se pueden realizar en una biblioteca. Posee gran similitud con el sistema Koha, en relación a la gran cantidad de opciones de configuración que pueden resultar en aplicaciones muy distintas de acuerdo al establecimiento que la configure.
- *Islandora*: es un framework de software open source diseñado para ayudar a instituciones y organizaciones (y sus audiencias) a realizar un manejo colaborativo de sus elementos digitales como libros, videos, imágenes (entre otros tipos). Posee una gran flexibilidad de configuración por su diseño modular. Distintos paquetes de funcionalidades pueden ser seleccionados o no, y a su vez dentro de los paquetes, distintos módulos.
- *Mahara*: es una aplicación web open source para gestionar portafolios electrónicos y redes sociales. Ofrece a los usuarios herramientas para crear y mantener un portafolio digital sobre su formación. Incluye funcionalidades sociales que permiten la interacción entre los usuarios. Además, otros tipos de funcionalidades que incluye son: creación de repositorios, blog, generador de currículum vitae, repositorios de archivos, entre otras. Brinda muchas posibilidades de configuración de funcionalidades por lo que también puede ser considerado una familia de aplicaciones.
- *Alfresco*: es una plataforma para manejo de documentos de negocios críticos. Automatiza procesos de negocios intensivos en documentos a través de un sofisticado manejo de usuarios y roles. Posibilita colaboración en gran escala a través de distintos sectores o negocios. Permite a las organizaciones capturar, almacenar, buscar y colaborar en documentos de muchos tipos distintos. No solo ofrece a las empresas herramientas de gestión documental, también se ocupa de la gestión de contenido web, records management, trabajo colaborativo, flujos de trabajo. Alfresco Community es la edición gratuita para desarrolladores, entornos de testeo o pequeñas instalaciones.
- *Omeka*: es un software open source, flexible y pensado para la publicación en la web de colecciones digitales de bibliotecas, archivos, museos o cualquier otra institución que desee difundir su patrimonio cultural.
- *CiviCRM*: es un paquete de software open source basado en la web para la gestión de relaciones con clientes, comúnmente denominados CRM. Posee una gran capacidad de configuración de acuerdo a la institución que lo utilice.

Si bien se mencionan solo algunos productos, teniendo en cuenta distintos dominios, existen numerosos productos similares de cada dominio.

Capítulo 5

Desarrollo de modelos de requisitos de dominio

5.1. Desarrollo de diagramas de casos de uso de dominio

En este trabajo utilizamos la notación de casos de uso con variabilidades definida en la sección 3.1.3 para desarrollar diagramas de casos de uso para familia de aplicaciones web.

5.1.1. Guías para construir diagramas de casos de uso de dominio

Partiendo de fuentes de información de entrada, que en general pueden ser documentos de texto con requisitos funcionales de la familia de aplicaciones web provistos por el cliente (especificando en detalle requisitos obligatorios y variables) brindamos algunas guías que pueden ser seguidas:

1. *Identificar actores.* Los actores son los elementos de modelado que representan la interacción de los usuarios con los casos de uso. Estos pueden ser usuarios humanos de la aplicación (con posibilidad de distintos roles), elementos de hardware, sistemas externos (necesarios para algunas funcionalidades de la familia de aplicación) u otro tipo de entidad que interactúe con el sistema.
2. *Identificar subsistemas.* Este paso puede ser obviado si se trata de una familia de aplicaciones web de pequeña escala. Para familias de aplicaciones grandes o medianas, es posible que los requisitos incluyan distintos agrupamientos de funcionalidades (de elevada granularidad) en subsistemas. Si esto no ocurre, no es una tarea compleja identificarlos a través de la lectura de los documentos de requisitos. Para cada subsistema se debe crear un paquete de casos de uso con el nombre representativo del mismo.
3. *Identificar paquetes de casos de uso.* Existen distintos criterios para agrupar funcionalidades en subsistemas o en grupos de funcionalidades, como pueden ser operaciones sobre mismas unidades de información, relación por tipo de funcionalidad u otros tipos de criterios a decisión del desarrollador. Por cada grupo de funcionalidad identificado, que no represente un subsistema, se debe crear un paquete de casos de uso con el nombre correspondiente al grupo de funcionalidades que incluirá. Cada paquete de casos de uso identificado debe ser incluido en el subsistema que corresponde (en caso de existir división del sistema en subsistema). En el caso de familia de aplicaciones muy grandes, esta tarea puede ser recursiva.
4. *Identificar paquetes de casos de uso variantes.* Se deben identificar cuales paquetes de casos de uso son obligatorios y cuales variantes. En el caso de que un paquete de casos de uso sea opcional se debe adjuntar al paquete un elemento PackageVariability con cardinalidad 0..1. En el caso que existan distintos paquetes alternativos de los cuales se deben seleccionar uno o más, se debe adjuntar a cada paquete alternativo un elemento PackageVariability con el mismo nombre de variabilidad y la cardinalidad correspondiente a la selección posible.
5. *Identificar casos de uso y relaciones entre ellos.* La siguiente tarea a realizar, es la identificación de las unidades de funcionalidades, es decir los casos de uso de la familia de aplicaciones. Para

ello, se pueden seguir distintos tipos de criterios, a decisión del analista, para la granularidad que cada caso de uso tenga. Además, estos deben ser asociados a los actores con los cuales interactúan. También se deben identificar las relaciones de tipo «**extend**» e «**include**» entre los casos de uso, para contemplar mecanismos de inclusión y extensión entre casos de uso. Finalmente, el último paso es incluir cada caso de uso en el paquete que corresponda (en caso de existir).

6. *Identificar variabilidades involucrando casos de uso y dependencias entre estas variabilidades.* La última tarea es la de definir cuales casos de uso serán obligatorios y cuales variantes, a éstos últimos se les deben anotar notas de variabilidades. A los casos de uso identificados como opcionales se les deben adjuntar un elemento de variabilidad *AssociationVariability* con cardinalidad 0..1. Para algunas funcionalidades en casos de uso pueden existir varias alternativas de realización de las mismas, estos casos de uso alternativos deben ser ajuntados dependiendo del tipo de alternativa, con un elemento de variabilidad que puede ser del tipo *ExtendVariability* (cuando las alternativas son relaciones de tipo «**extend**»), *IncludeVariability* (cuando las alternativas son relaciones de tipo «**include**») o *AssociationVariability* (cuando las alternativas están relacionadas a actores).

5.1.2. Aplicación al caso de estudio de Sistema de Administración de Biblioteca Online

Identificación de actores. En la figura 5.1 se muestran los actores identificados para el caso de estudio de sistema de administración de bibliotecas online. Se descubrieron siete actores, cinco de los cuales representan roles de usuario humano: actor *Administrator* (se encarga de tareas de administración y configuración del sistema), actor *Staff member* (es un miembro de la biblioteca que se ocupa de mantener los usuarios y funcionalidades que el sistema provee a los usuarios), actor *Patron* (es un cliente o usuario de un sistema de biblioteca online), actor *Anonymous Patron* (es un cliente o usuario anónimo de un sistema de biblioteca online) y actor *Public User* (es un usuario público, que no necesariamente está asociado a la biblioteca); además se identificó un actor de tipo hardware llamado *Timer* (es un temporizador que realiza acciones cada cierto período de tiempo) y un actor que es un componente del sistema llamado actor *Statistical Patron* (se encarga de registrar el uso de los clientes de algunas operaciones que el sistema provee).



Figura 5.1: Actores identificados para el caso de estudio de familia de sistemas de administración de bibliotecas online.

Identificación de subsistemas. Para el caso de estudio se identificaron tres subsistemas (ver Figura 5.2):

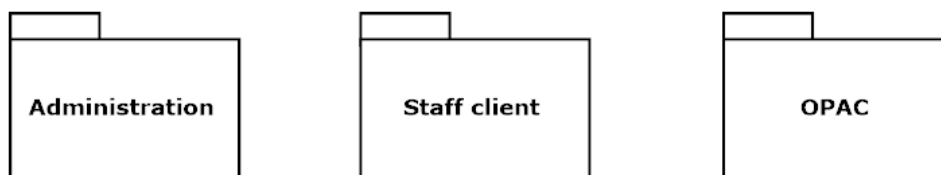


Figura 5.2: Subsistemas identificados para el caso de estudio de familia de sistemas de administración de bibliotecas online.

- **Administration:** se encarga de tareas de administración, tales como fijar preferencias globales del sistema, parámetros básicos, definir reglas de uso y otro tipo de configuraciones necesarias para una aplicación de biblioteca online. Se creó un paquete de casos de uso con el mismo nombre.
- **Staff client:** representado con el paquete staff client, contiene las distintas tareas y operaciones que los miembros de la biblioteca se deben encargar de realizar, tales como administración de usuarios (alta, baja, modificación, consultas y otras operaciones), circulación, administración de reportes y catálogos, administración de carrito, entre otras tareas.
- **OPAC:** representado con el paquete OPAC, contiene las distintas funcionalidades con las que el usuario final puede interactuar con el sistema, tales como búsqueda de ítems, administración de cuenta de usuario, hacer operaciones en el carrito y otras funcionalidades que pueden proveer los sistemas de bibliotecas online.

Identificación de paquetes de casos de uso. En la figura 5.3 se muestran los paquetes identificados para el subsistema Administration del caso de estudio. Los mismos son: *Global System preferences* (para fijar preferencias globales de una biblioteca), *Libraries, Branches and Groups* (para administración de elementos de tipo bibliotecas, grupos de bibliotecas; entre otras funcionalidades), *Item types* (para administración de tipos de ítems de biblioteca), *Authorized values* (administración de valores autorizados para controlar los valores que pueden ser introducidos en los campos MARC), *Patron categories* (administración de tipos de clientes de una biblioteca), *Patron attributes types* (administración de tipos de atributos de clientes de una biblioteca), *Circulation and Fine rules* (incluye varias funcionalidades de administración de políticas de checkout de ítems y de reglas de circulación), *Catalog* (administración de catálogo, contiene administración de reglas de presentación y distintas funcionalidades de mapeos de formatos), *Currencies and Exchange rates* (administración de monedas y tasas de cambio), *Budgets* (administración de presupuestos) y *Funds* (administración de fondos). No se identificó ningún grupo de funcionalidades recursivo en este subsistema, por lo no hay paquetes anidados.

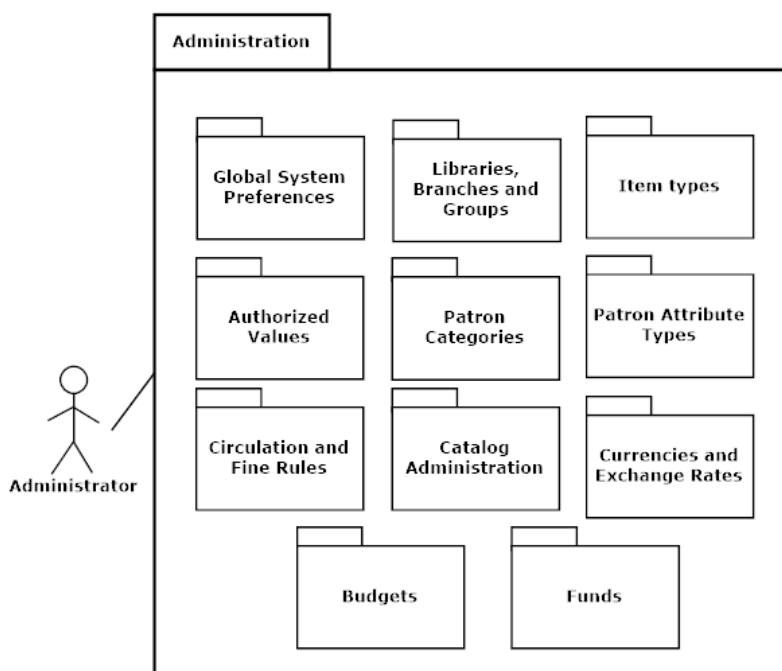


Figura 5.3: Paquetes identificados para el subsistema Administration del caso de estudio de familia de sistemas de administración de bibliotecas online.

En la figura 5.4 se muestran los paquetes identificados para el subsistema Staff client del caso de estudio. Los mismos son: *Fines* (distintas funcionalidades acerca de penalizaciones o multas), *List* (para administración y manejo de listas de ítems), *Patrons* (administración de clientes y de operaciones que los

mismos pueden realizar), *Circulation* (funcionalidades con respecto a ítems), *Acquisitions* (para manejo y administración de adquisiciones realizadas por la biblioteca), *Cataloguing* (administración de ítems), *Serials* (administración de suscripciones y listas de enrutamiento), *Reports* (creación y manejo de distintos tipos de reportes), *Purchase suggestions* (para creación y visualización de sugerencias de compra), *Comments and Holidays* (visualización de calendario y manejo de eventos sobre el mismo). A su vez, los paquetes *Acquisitions*, *Cataloguing*, y *Serials* incluyen otros paquetes de casos de uso dentro.

El paquete *Acquisitions* (ver figura 5.5) incluye los paquetes: *Vendors* (administración, manejo y operaciones con vendedores), *Orders* (administración, manejo y operaciones con órdenes), *Baskets* (administración, manejo y operaciones con cestas) y *Notices* (administración y manejo de notificaciones).

El paquete *Cataloguing* (ver figura 5.6) incluye los paquetes *Records* (administración, manejo y operaciones con registros), *Items* (administración, manejo y operaciones con ítems) y *Authorities* (administración, manejo y operaciones con autoridades).

El paquete *Serials* (ver figura 5.7) incluye al paquete *Routing list* (administración y manejo de listas de enrutamiento).

En la figura 5.8 se muestran los paquetes identificados para el subsistema OPAC. Ellos son: *Public Search* (para realizar búsquedas sobre los ítems de la biblioteca por cualquier usuario web), *Private Search* (para realizar búsquedas sobre los ítems de la biblioteca solamente por clientes de la biblioteca), *My account* (todo tipo de funcionalidades concernientes a una cuenta de usuario perteneciente a un cliente de la biblioteca), *OPAC Fines* (información y operaciones acerca de las multas o penalizaciones de un cliente), *Holds in OPAC* (información y operaciones acerca de las retenciones de un cliente), *Comments in OPAC* (operaciones acerca de los comentarios que puede realizar un cliente), *OPAC renewal* (operaciones acerca de renovaciones de préstamos de un ítem), *Cart* (lo necesario para realizar el proceso de checkout de ítems), *Browse Subject Authorities* (navegar por categorías de asuntos de un ítem).

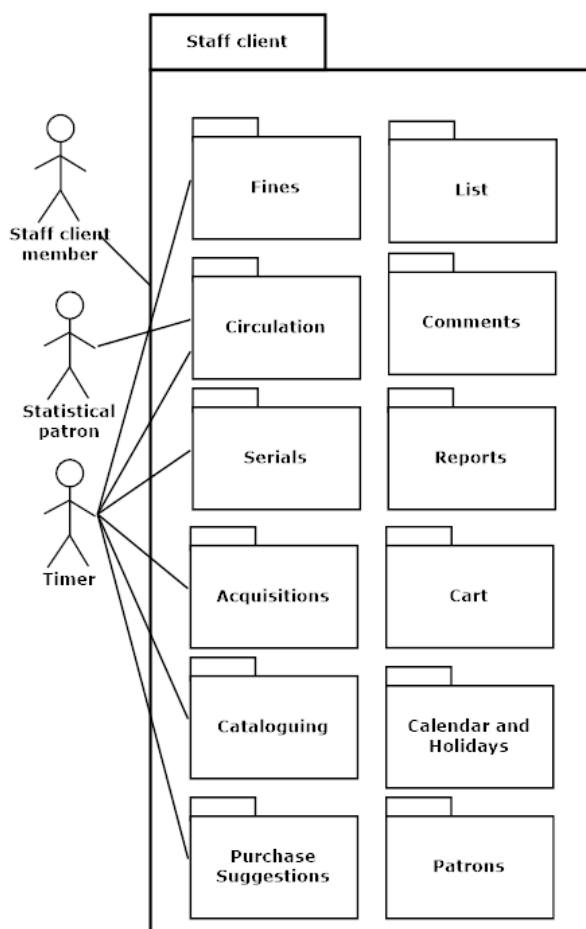


Figura 5.4: Paquetes identificados para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.

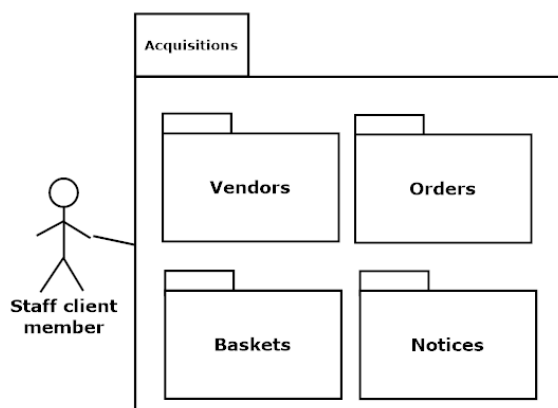


Figura 5.5: Paquetes identificados dentro del paquete de casos de uso Acquisitions para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.

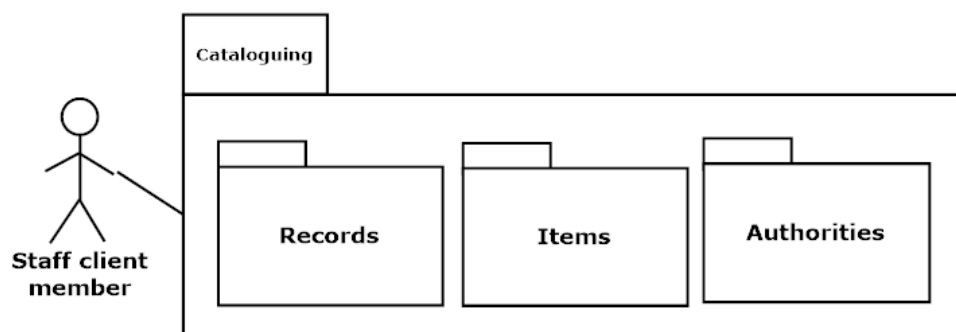


Figura 5.6: Paquetes identificados dentro del paquete de casos de uso Cataloguing para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.

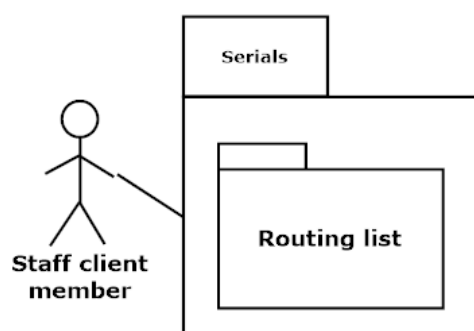


Figura 5.7: Paquete identificado dentro del paquete de casos de uso Serials para el subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.

Algunos paquetes identificados en el subsistema OPAC, a su vez incluyen otras agrupaciones de funcionalidades dentro. En la figura 5.9 se muestran los paquetes de casos de uso identificados para el paquete My account: *Tag* (funcionalidades para el manejo de etiquetas), *Search history* (funcionalidades para visualizar y administrar el historial de búsquedas realizadas por un cliente), *Reading history* (funcionalidades para visualizar y administrar el historial de lecturas realizadas por un cliente), *Purchase suggestions* (funcionalidades para visualizar y configurar sugerencias de compra para el cliente), *Lists* (para administración y manejo de listas de ítems por un cliente) y *OPAC Fines* (para visualizar multas del cliente).

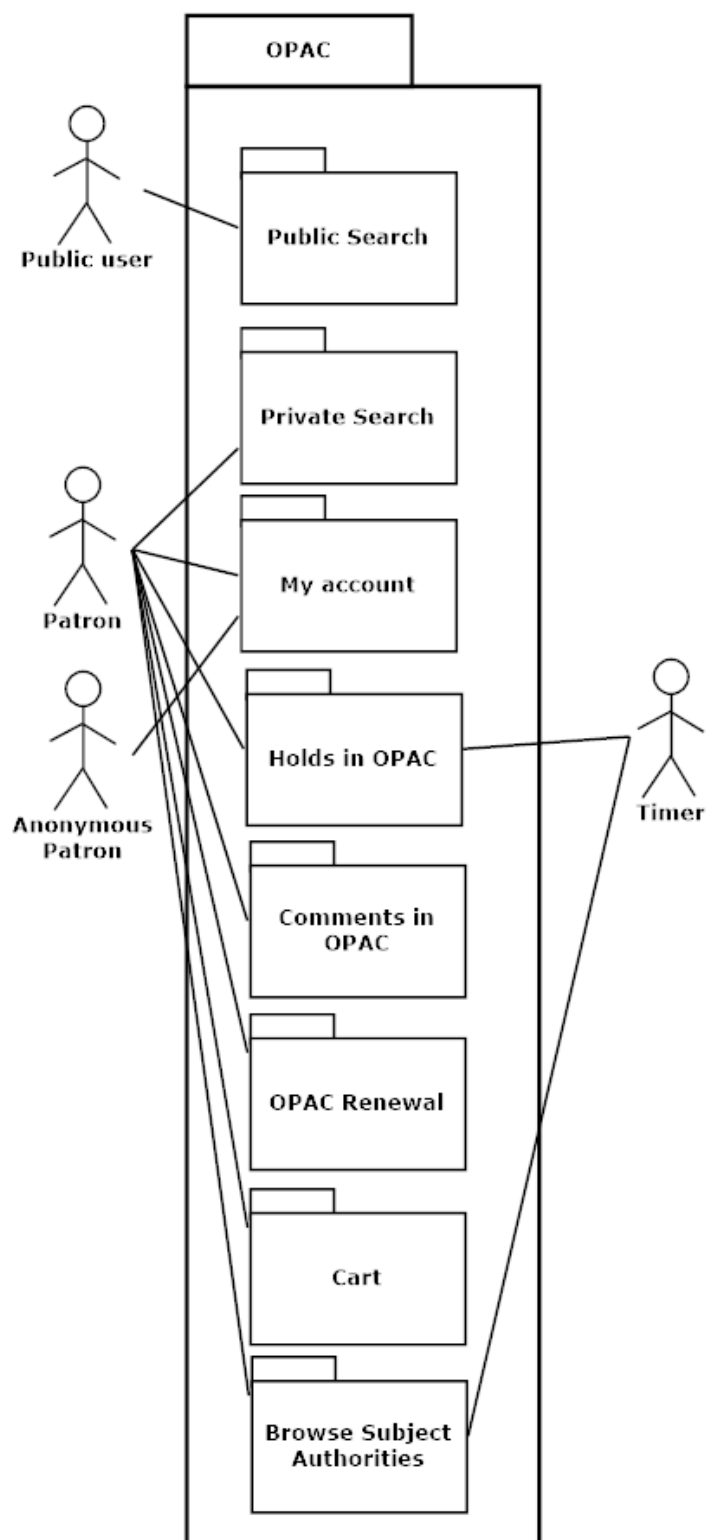


Figura 5.8: Paquetes identificados para el subsistema OPAC del caso de estudio de familia de sistemas de administración de bibliotecas online.

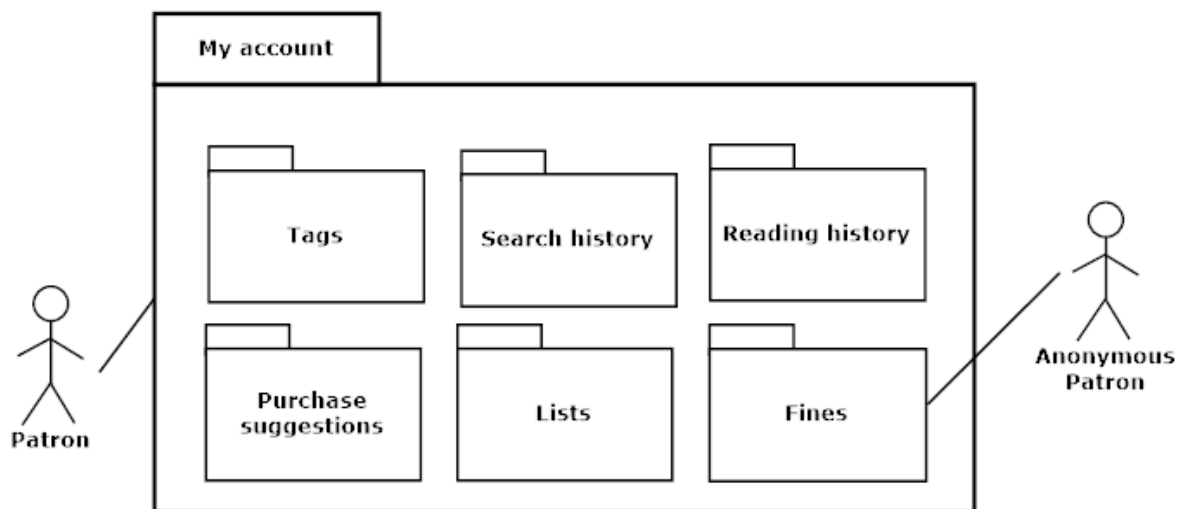


Figura 5.9: Paquetes identificados dentro del paquete de casos de uso My account del subsistema OPAC del caso de estudio de familia de sistemas de administración de bibliotecas online.

Identificación de agrupamientos de funcionalidades variantes. Dentro del subsistema Administration del caso de estudio de familia de sistemas de administración de bibliotecas online se encontró solo un paquete de casos de uso variante, *Patron attribute types* (ver figura 5.10), que es una agrupación opcional de funcionalidades (puede ser seleccionada o no) y que a su paquete de casos de uso se le adjuntó una anotación de variabilidad llamada *ExtendedPatronAttributes* con una cardinalidad 0..1, lo cual denota opcionalidad. El resto de los paquetes del subsistema Administration son obligatorios y deben estar en todos los miembros de la familia de aplicaciones.

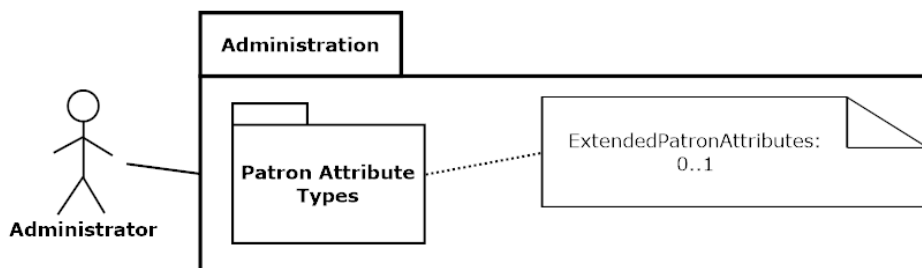


Figura 5.10: Variabilidades identificadas en paquetes del subsistema Administration del caso de estudio de familia de sistemas de administración de bibliotecas online.

Dentro del subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online se hallaron tres paquetes de casos de uso variantes (ver figura 5.11); *Cart*, *Comments* y *Routing Lists*. Los tres representan agrupamientos de funcionalidades opcionales (pueden ser seleccionadas o no dependiendo de las necesidades de la aplicación a desarrollar) y a sus paquetes de casos de uso correspondientes se les adjuntó una anotación de variabilidad con nombres representativos al grupo de funcionalidades y con una cardinalidad 0..1, lo cual refleja que es un paquete opcional. El resto de los paquetes del subsistema Staff client son obligatorios y deben estar en todos los miembros de la familia de aplicaciones.

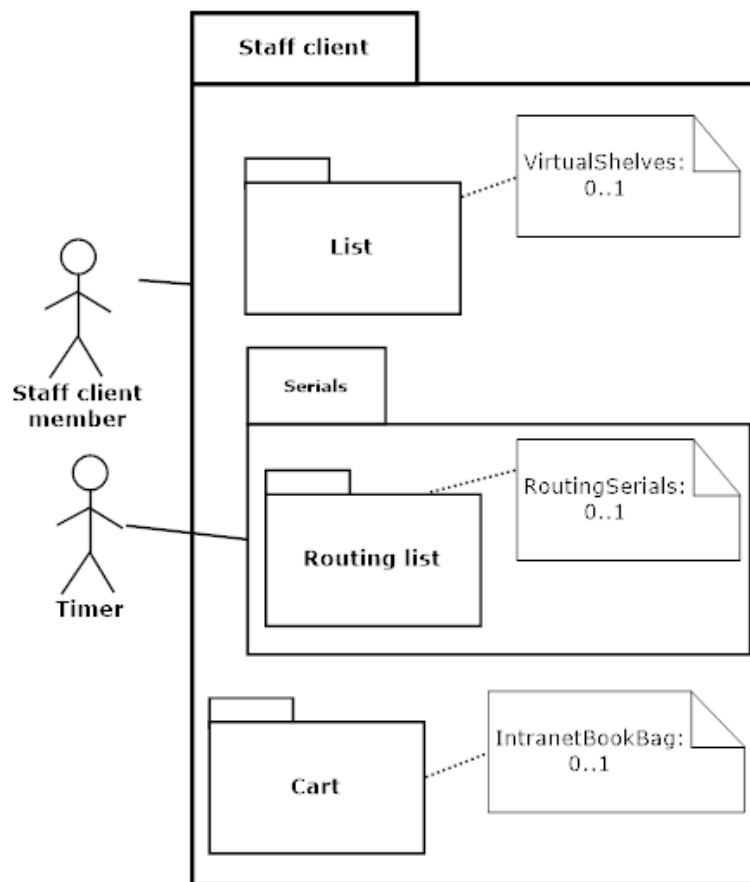


Figura 5.11: Variabilidades identificadas en paquetes del subsistema Staff client del caso de estudio de familia de sistemas de administración de bibliotecas online.

El subsistema OPAC del caso de estudio de familia de sistemas de administración de bibliotecas online contiene el único caso de variabilidad que involucra más de un variante encontrado en variabilidad de agrupamiento de funcionalidades. Se trata de la variabilidad *SearchType*, con cardinalidad 1..1 que involucra a los paquetes de casos de uso *Private Search* y *Public Search*, y de los cuales al momento de la configuración se debe seleccionar uno, que posibilitará a los usuarios de la aplicación tener un tipo de búsqueda de ítems determinado, ya sea público (sin necesidad de registro) o privado (los usuarios deberán ser miembros de la biblioteca online para poder realizar búsquedas de ítems), dependiendo de las necesidades de una aplicación. Además, se hallaron varios paquetes de casos de uso opcionales (ver figura 5.12); es decir con variabilidad con cardinalidad 0..1. Ellos son: *Tags*, *Search History*, *Reading History*, *Search History*, *Purchase Suggestions*, *Lists*, *OPAC Fines*, *Holds in OPAC*, *Comments in OPAC* y *OPAC Renewal*. El resto de los paquetes del subsistema OPAC son obligatorios y deben estar en todos los miembros de la familia de aplicaciones.

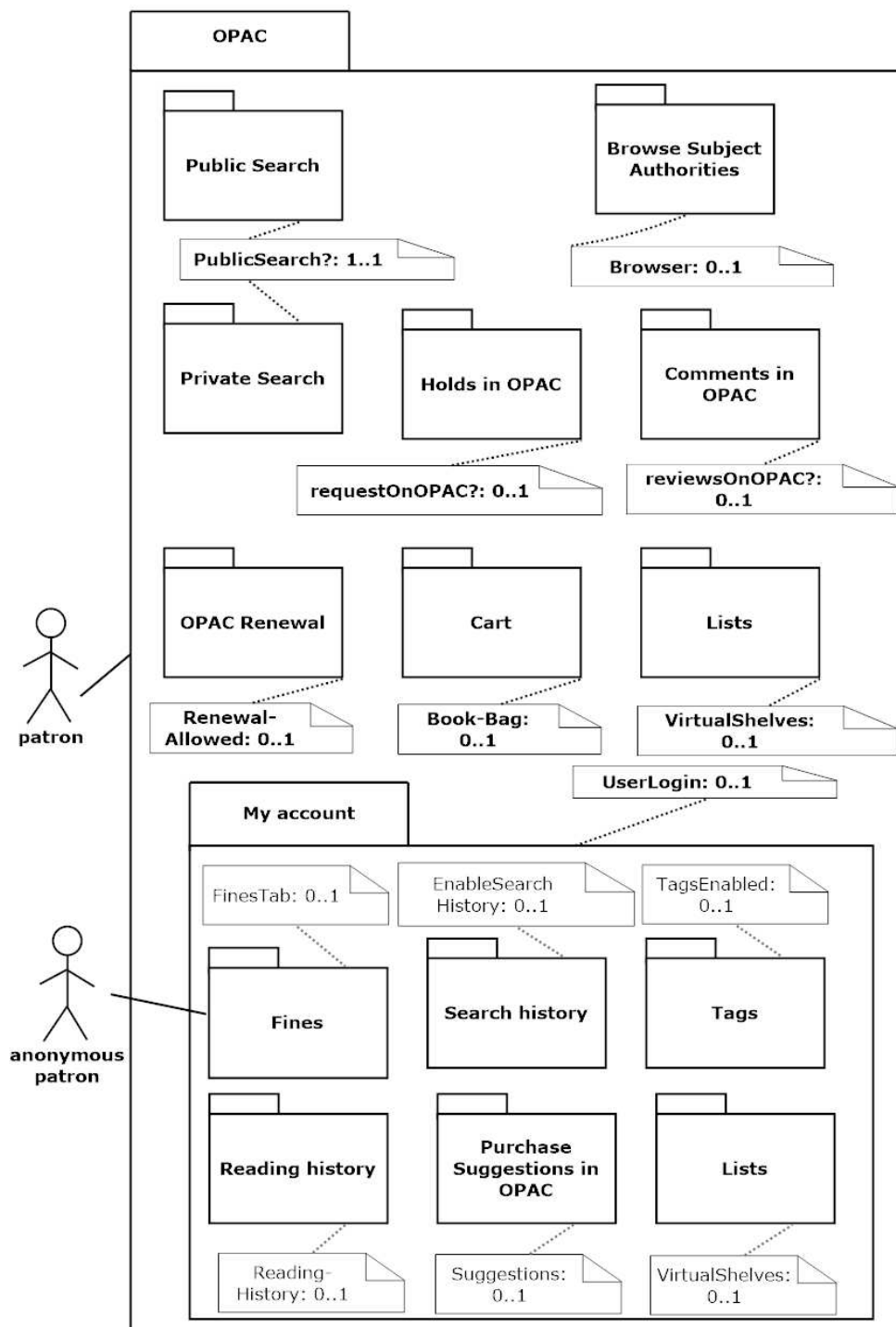


Figura 5.12: Variabilidades identificadas en paquetes del subsistema OPAC del caso de estudio de familia de sistemas de administración de bibliotecas online.

Identificación de casos de uso, relaciones entre casos de uso, variabilidades involucrando casos de uso y dependencias entre variabilidades. Se ejemplifican los casos de uso que se identificaron en el caso de estudio a través del paquete OPAC, ya que es el paquete que contiene mayor diversidad de situaciones de variabilidad de casos de uso y mayor cantidad de casos de uso.

En la Figura 5.13 se muestra el paquete de casos de uso OPAC del sistema de manejo de bibliotecas online con los paquetes y los casos de uso que el mismo incluye. Dentro del paquete OPAC se identificaron dos casos de uso: *see most checked out items* y *authority records search*, los cuales se ven involucrados en

variabilidades del tipo AssociationVariability (*TopIssue* y *Authorities*, respectivamente) con cardinalidad 0..1, es decir que son casos de uso opcionales.

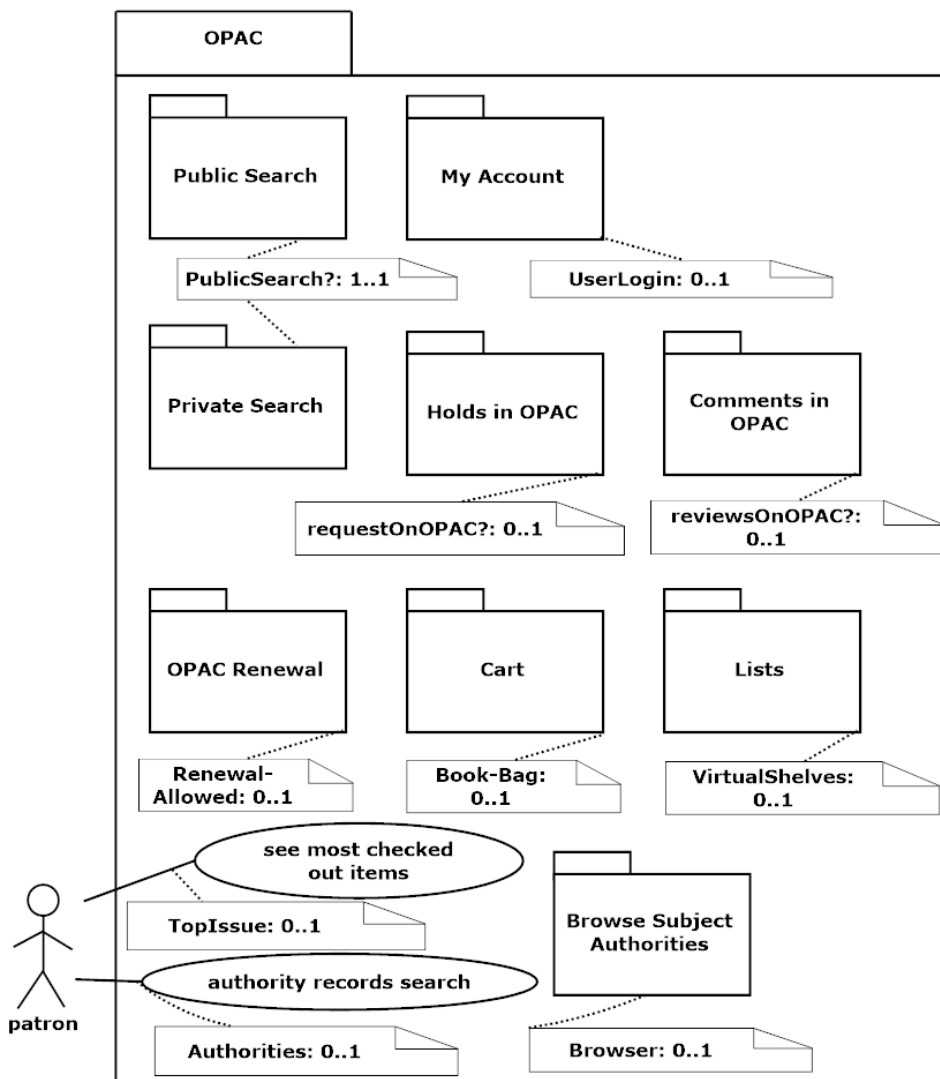


Figura 5.13: Paquete de casos de uso OPAC.

con distintos tipos de relaciones, variabilidades entre casos de uso y relaciones de dependencia entre variabilidades. Se incluyen dos relaciones de tipo extend que no están involucradas en variabilidades, el caso de uso *cancel hold* extiende el comportamiento del caso de uso *show holds* y el caso de uso *show my fines* extiende el comportamiento del caso de uso *show fines summary*. Además, se incluye una variabilidad de tipo ExtendVariability con cardinalidad 0..1 llamada *MySummary*, que significa que el caso de uso *show links* extiende opcionalmente al caso de uso *see checked out items*. El caso de uso *show account summary* incluye tres casos de uso (a través de relación include): *see checked out items*, *see overdue items* y *show holds*; e incluye opcionalmente (a través de la variabilidad *Fines Allowed* de tipo IncludeVariability con cardinalidad 0..1) al caso de uso *show fines summary*, que se encuentra dentro del paquete de casos de uso *Fines*. Esta variabilidad tiene una relación de dependencia de tipo requires con la variabilidad OPAC*Fines*, que significa que para seleccionar esta variabilidad se requiere haber seleccionado el paquete de casos de uso *Fines*, que también es opcional. El resto de casos de uso presentes en el paquete solo se encuentran relacionado al actor *Patron* a través de asociaciones, cuatro de ellos son opcionales, es decir que sus asociaciones con el actor *Patron* están dentro de un elemento AssociationVariability con cardinalidad 0..1. La variabilidad *PatronDetails* de tipo AssociationVariability tiene una cardinalidad 1..1 e involucra dos relaciones de asociación de los casos de uso *show my details* y *change my details* con el actor *Patron*, lo cual significa que se debe seleccionar uno de los dos casos de uso. La variabilidad opcional *Privacy*, de tipo AssociationVariability, que involucra a la relación entre el actor *Patron* y el caso de uso *show my privacy settings* se encuentra en una relación de dependencia de tipo requires entre ésta variabilidad y la variabilidad opcional *Reading history*, que significa que el paquete *Reading history* debe ser seleccionado para poder seleccionar el caso de uso *show my privacy settings*.

En la figura 5.15 se encuentran los casos de uso, las relaciones entre los mismos y la variabilidades de casos de uso identificadas en los paquetes de casos de uso (pertenecientes al subsistema OPAC) *Public Search* y *Private Search*. Ambos paquetes tienen la misma estructura de casos de uso, con la diferencia del actor que los puede utilizar. Contienen dos variabilidades de tipo ExtendVariability con cardinalidad 0..1 que son *Shelf Brower* entre el caso de uso extendido *search catalog from OPAC* y el caso de uso extensión *view near items on the shelf*; y *SearchForTitleIn* entre el caso de uso extendido *show record from OPAC* y el caso de uso extensión *search for title in*. El resto de relaciones no incluyen variabilidades y son las comunes de tipo extend e include.

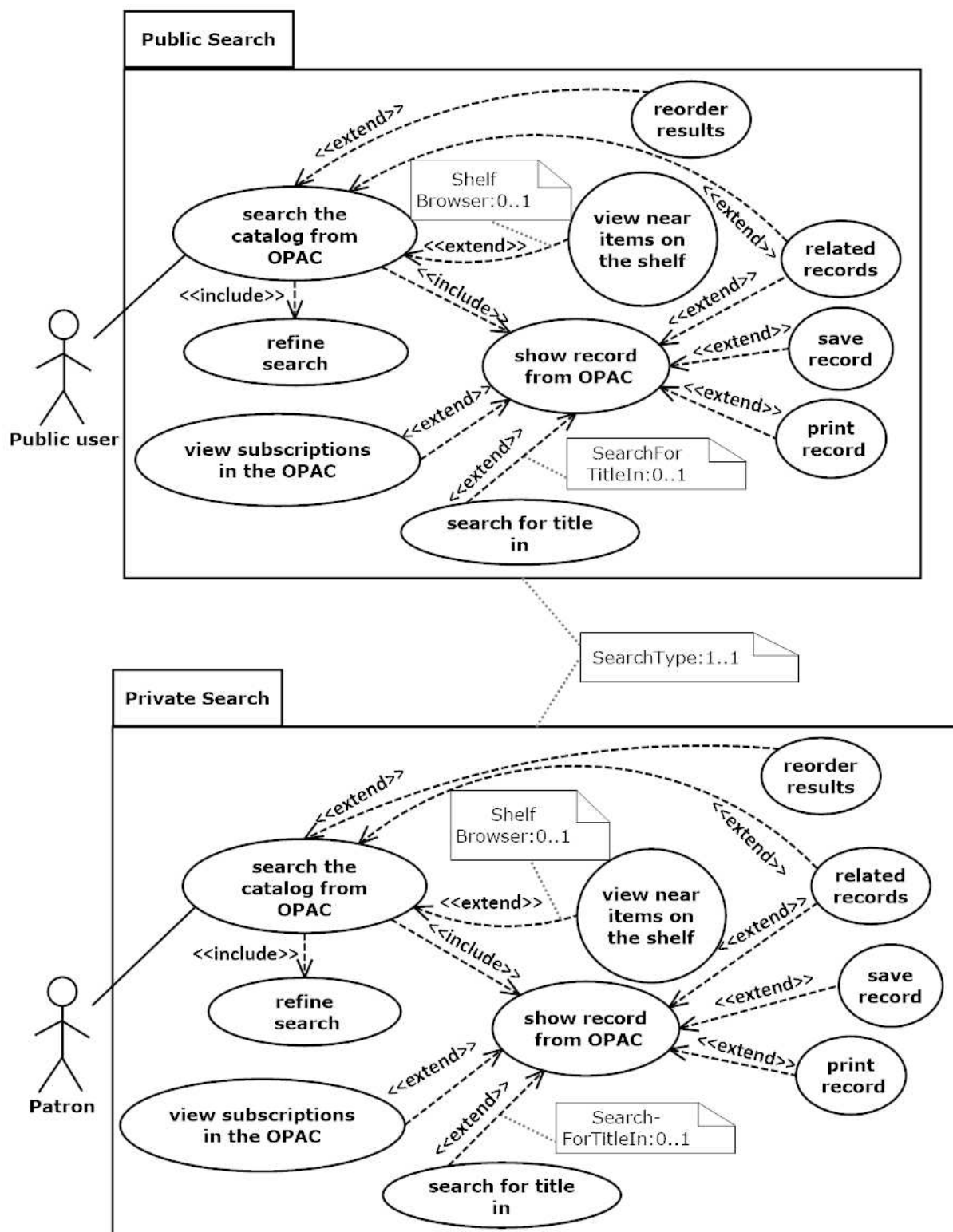


Figura 5.15: Casos de uso, variabilidades y relaciones entre casos de uso identificados en los paquetes, del subsistema OPAC, Public Search y Private Search.

La figura 5.16 muestra la composición de casos de uso de los paquetes, que se encuentran dentro del paquete My account del subsistema OPAC: *Tags*, *Search history*, *Reading history* y *Purchase suggestions*. Solo hay tres variabilidades de casos de uso y el resto son relaciones de tipo extend que no se ven

afectadas por variabilidades (algunas de estas extensiones afectan a casos de uso de otros paquetes). Las tres variabilidades existentes son opcionales (es decir con cardinalidad 0..1), dos de ellas de tipo ExtendVariability (*tagsInputOnDetail* y *tagsInputOnList*) y la restante de tipo AssociationVariability (*AnonSuggestions*).

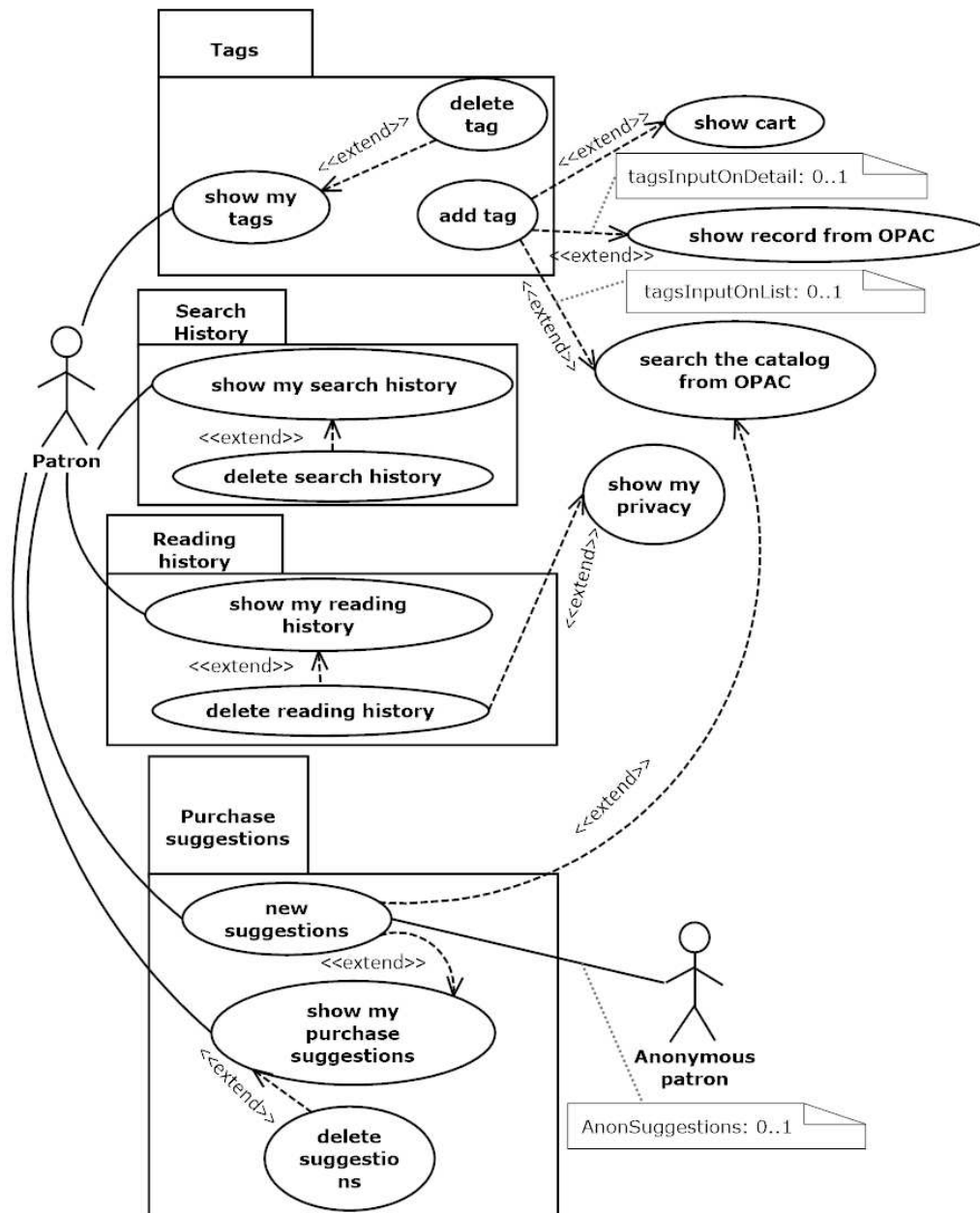


Figura 5.16: Casos de uso, variabilidades y relaciones entre casos de uso identificados en los paquetes (que están dentro del subsistema OPAC y del paquete My account) Tags, Search history, Reading history y Purchase suggestions.

La figura 5.17 muestra la composición de casos de uso del paquete *List*, que se encuentra dentro del paquete My account del subsistema OPAC. Dicho paquete no posee variabilidades entre casos de uso, pero si una rica estructuración a través de relaciones de tipo extend.

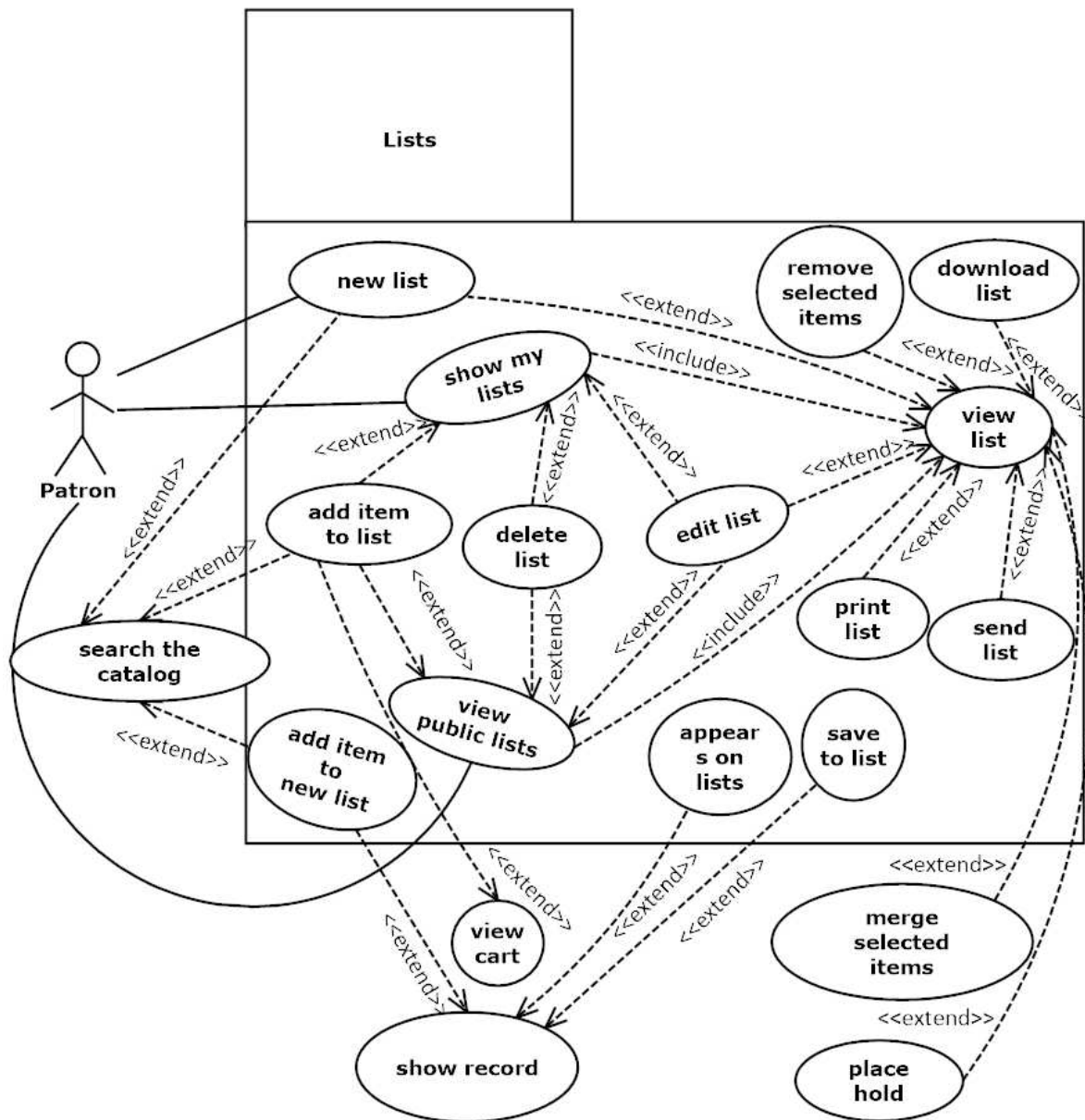


Figura 5.17: Casos de uso y relaciones entre casos de uso identificados en el paquete (que está dentro del subsistema OPAC y del paquete My account) Lists.

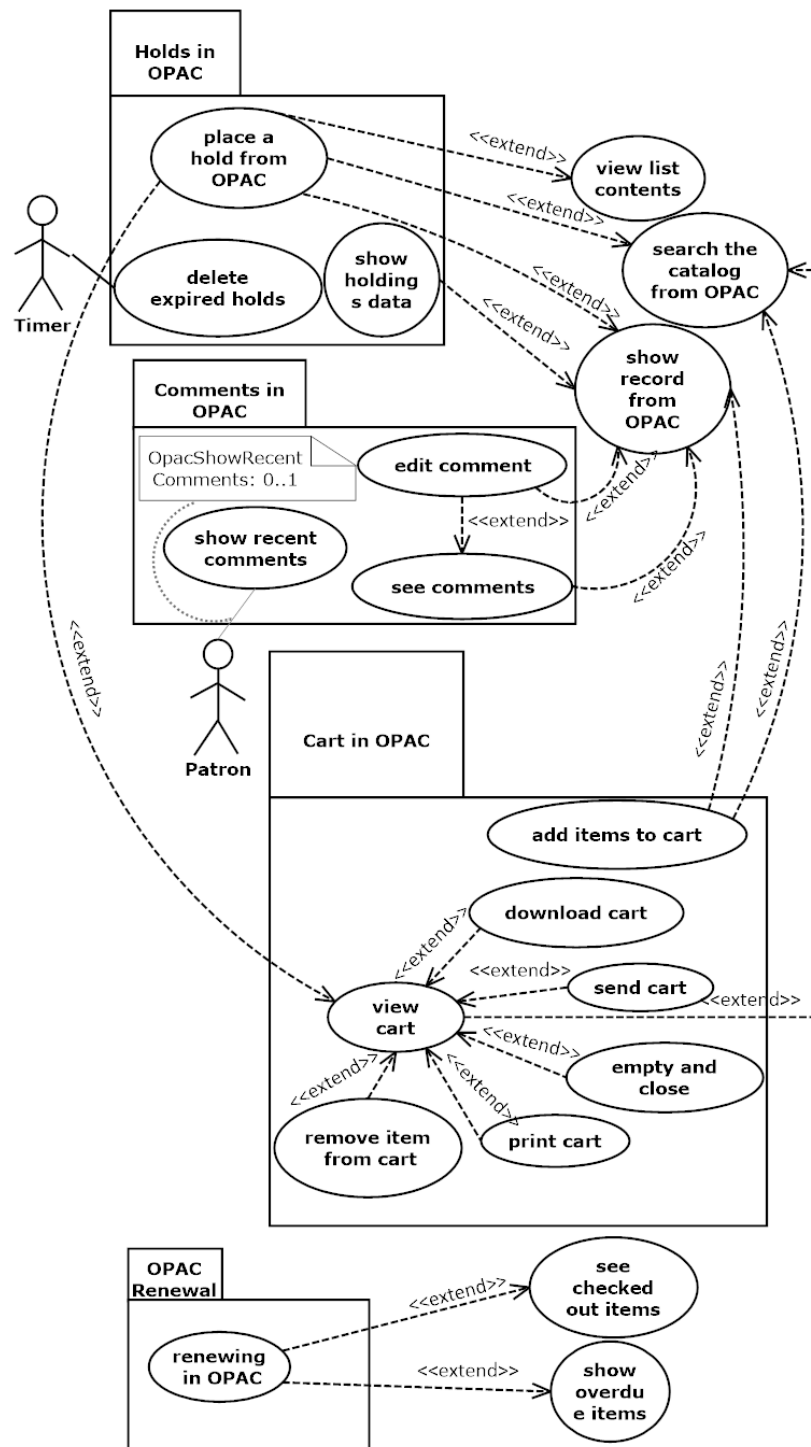


Figura 5.18: Casos de uso, variabilidades y relaciones entre casos de uso identificados en los paquetes (que están dentro del subsistema OPAC y del paquete My account) Holds in OPAC, Comments in OPAC, Cart in OPAC y OPAC Renewal.

La figura 5.18 muestra la composición de casos de uso de los paquetes que se encuentran dentro del paquete My account del subsistema OPAC, *Holds in OPAC*, *Comments in OPAC*, *Cart in OPAC* y *OPAC Renewal*. Solo hay una variabilidad de casos de uso y el resto son relaciones de tipo extend que no se ven afectadas por variabilidades (algunas de estas extensiones afectan a casos de uso de otros paquetes). La variabilidad existente es opcional (es decir con cardinalidad 0..1), se encuentra dentro del

paquete de casos de uso Comments in OPAC con el nombre *OpacShowRecentComments* y es del tipo AssociationVariability e involucra a la asociación entre el actor Patron y el caso de uso *show recent comments*. El resto de las relaciones que se encuentran en la figura son de tipo extend y no involucran variabilidades.

La figura 5.19 muestra la composición de casos de uso del paquete *Browse Subject Authorities*, que se encuentra dentro del subsistema OPAC. Dicho paquete no contiene variabilidades de casos de uso.

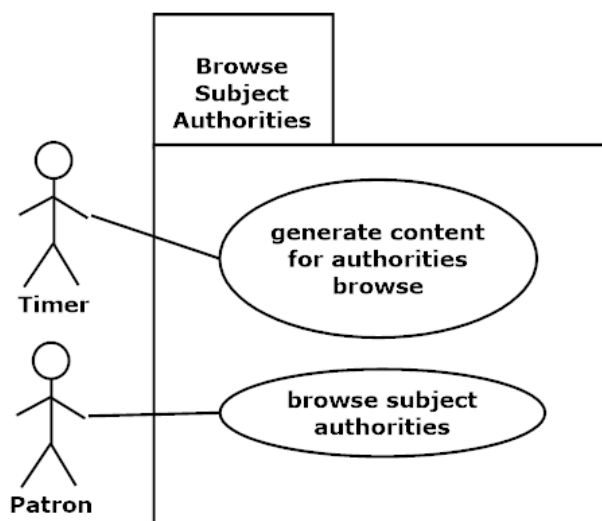


Figura 5.19: Casos de uso identificados en el paquete (que está dentro del subsistema OPAC y del paquete My account) Browse Subject Authorities.

5.2. Desarrollo de diagramas de actividad de dominio

Para desarrollar diagramas de actividad para familias de aplicaciones web, en este trabajo utilizamos la notación de diagramas de actividad con variabilidades definida en la sección 3.2.3. Además, se utiliza la taxonomía para describir acciones de diagramas de actividad de aplicaciones web presentada en 2.1.1.

Una vez que se cuenta con los casos de uso con variabilidades de la familia de aplicaciones web que se está desarrollando, el siguiente paso es la descripción detallada de cada caso de uso con variabilidades por medio de diagramas de actividad con variabilidades. Estas descripciones detalladas pueden dar lugar a variabilidades a un nivel de granularidad más fina, que deben ser identificadas y documentadas haciendo uso de la notación definida en la sección 3.2.3, la cual también permite tener en cuenta distintas consideraciones de reutilización. Por medio del perfil UML para diagramas de actividad de aplicaciones web de 2.1.1 cada acción de diagramas de actividad es estereotipada de manera que resulte más claro su fin y su posterior refinamiento en etapas siguientes del proceso de desarrollo.

5.2.1. Guías para construir diagramas de actividad de dominio

Si bien no se pretende presentar un método estricto de desarrollo, algunas guías pueden ser seguidas y tenidas en cuenta para facilitar el desarrollo de esta etapa:

1. *Construir diagramas de actividad con variabilidades.*

- *Identificar la clasificación de cada acción.* De acuerdo a la taxonomía presentada en 2.1.1, cada acción identificada en el diagrama de actividad con variabilidades debe ser estereotipada según su función en el caso de uso, ya sea si es realizada por el sistema o por el usuario. Esto es importante para identificar la interacción del usuario con el sistema y ayuda a separar tareas en las etapas siguientes.

- *Identificar el resto de elementos y elementos variantes.* Distintos elementos participantes de un diagrama de actividad deben ser identificados (como nodos de objeto, IBD, CBA, nodos de decisión y demás elementos). Además, se deben identificar las variabilidades presentes y de acuerdo a ellas, modelar el diagrama de actividad siguiendo la notación presentada en 3.2.3 (allí se expresa en profundidad cómo y en qué circunstancias se debe utilizar cada elemento de variabilidad).
 - *Conectar los elementos identificados.* Finalmente, para que se constituya correctamente un diagrama de actividades con variabilidades, las acciones y distintos elementos identificados (ya sean variantes o no) deben ser conectados a través de aristas de actividad (de flujo de datos o de flujo de control) para así determinar el flujo a seguir en cada actividad.
2. *Identificar subdiagramas para reutilización en los diagramas de actividad con variabilidades.* Puede haber distintas situaciones de subdiagramas dentro de un diagrama de actividad que se repiten (involucrando variabilidades o no, que son elementos variantes o no), ya sea dentro del mismo diagrama de actividad o en distintos diagramas de actividad. Existen dos tipos de éstas situaciones: la primera es descrita con el Problema 6 (subdiagramas reutilizables que no requieren alteraciones ni variaciones dependiendo de donde sean utilizados) y la segunda se describe en el Problema 7 (subdiagramas reutilizables que contienen alteraciones y variaciones de acuerdo al contexto en donde son utilizados). Es necesario distinguir entre estos dos tipos de situaciones para tener en claro cómo generalizar y reutilizar el subdiagrama. Además, para que una situación sea identificada como subdiagrama de reutilización se debe dar: *a)* que la situación contenga caminos o elementos que son utilizados más de una vez, ya sea dentro del mismo diagrama de actividad o en otros diagramas de actividad; *b)* que la situación contenga elementos o caminos accedidos a través de una guarda de contexto que hace referencia a más de un contexto.
 3. *Generalizar y abstraer los subdiagramas para reutilización.* Una vez identificados los subdiagramas para reutilización dentro de los diagramas de actividad con variabilidades, para que estos puedan ser reutilizados es necesario que sean abstraídos a un cierto nivel que permita que los mismos contemplen todas las situaciones donde los subdiagramas sin generalizar son utilizados. Si el subdiagrama reutilizable es del tipo del Problema 6 se deben abstraer nombres de elementos que son específicos de una situación a nombres más generales que pueden ser interpretados para distintas situaciones donde el subdiagrama puede ser reutilizado. Si el subdiagrama reutilizable es del tipo del Problema 7, además puede ser necesario definir distintas guardas con contextos dentro de los subdiagramas reutilizables para los distintos contextos posibles.
 4. *Modificar los diagramas de actividad con variabilidades utilizando los subdiagramas para reutilización.* Finalmente se deben reemplazar los subdiagramas identificados siguiendo la guía 2 por los subdiagramas generalizados y abstraídos siguiendo la guía 3. Para esto, puede ser necesario definir guardas con contextos para cada situación de reutilización.
 5. *Reutilizar diagramas ya generalizados (en caso de existir).* Si en la actividad que se está desarrollando se detecta algún subdiagrama de reutilización que ya ha sido generalizado para alguna actividad anterior, este subdiagrama de reutilización puede ser incluido (ya sea con algunas pequeñas adaptaciones a su generalización o no) en la actividad.

5.2.2. Aplicación al caso de estudio de Sistema de Administración de Biblioteca Online

Ya que el caso de estudio posee una elevada cantidad de casos de uso, aquí se incluyen algunos diagramas que ejemplifican la aplicación de las guías para construir diagramas de actividad con variabilidades que, además, son diagramas de actividad con variabilidades de los casos de uso correspondientes al subsistema OPAC, debido a que este subsistema incluye casos de uso ricos a nivel de situaciones de variabilidades y restricciones de dependencias halladas.

Debido a que los diagramas de actividad ocupan un gran volumen de espacio, en general se presentan las explicaciones a los mismos y luego los diagramas.

Construcción de diagramas de actividad con variabilidades. La figura 5.20 presenta la primer parte de diagrama de actividad para el caso de uso *Search catalog from the OPAC* (por razones de espacio

ha sido dividido en dos partes), que ha sido producido como resultado de aplicar la primer guía para construcción de diagramas de actividad con variabilidades. El caso de uso comienza con la selección del usuario del tipo de búsqueda a realizar y en base a esa elección se realizan las acciones correspondientes a alguno de los tres tipos de búsquedas posibles, los cuales pueden ser *basic search* (búsqueda básica), *advanced search* (búsqueda avanzada, con más opciones) y *command language search* (búsqueda basada en lenguaje de comandos). Los dos primeros tipos de búsquedas presentan una variabilidad con dos IBD, de los cuales solo uno debe ser seleccionado en el momento de la configuración. Luego de que el usuario ingrese los datos para el tipo de búsqueda seleccionado, se incluye una acción opcional (variabilidad llamada *HideLostItems* con cardinalidad 0..1) para que el sistema busque también incluyendo ítems extraviados. Además se incluye una variabilidad opcional *HoldOnOpac*, la cual está afectada por una relación de dependencia del tipo *Requires* hacia el paquete *Hold in OPAC*.

La continuación del caso de uso *Search catalog from the OPAC* está dado en la figura 5.21, la cual incluye la presentación de los resultados de la búsqueda al usuario con la posibilidad de realizar alguna operación sobre los ítems de resultado presentados. Los resultados pueden ser mostrados al usuario de dos maneras posibles: *normal view*, que representa una vista normal y *XSLT view*, donde los resultados son mostrados utilizando plantillas XSLT. Estas dos alternativas son modeladas con dos IBD y una variabilidad con cardinalidad 1..1 llamada *OPACXSLTResultsDisplay*. Finalmente, el sistema obtiene el estado de cada ítem resultado de la búsqueda, y en base a ellos y a la selección del usuario de algún ítem, el sistema sugiere una lista de acciones a realizar con el ítem.

La figura 5.22 presenta la primer parte de diagrama de actividad para el caso de uso *Show record in the the OPAC* (por razones de espacio ha sido dividido en dos partes), el cual incluye varias situaciones de variabilidad interesantes. En el comienzo de la actividad se incluyen dos variabilidades anidadas, donde se debe decidir, entre 3 variantes, el tipo de vista de ítem a visualizar (a través de la variabilidad con cardinalidad 1..1 *BiblioDefaultView*). Si el variante elegido es el intermedio, a su vez se debe elegir entre dos variantes mediante la variabilidad *marcFlavour*, con cardinalidad 1..1. Esta variabilidad se repite más adelante en el diagrama. Además hay que decidir, de manera muy similar con el caso de uso *Search catalog from the OPAC*, el tipo de vista de un ítem, entre *normal record view* y *XSLT record view*.

La continuación del caso de uso *Show record in the the OPAC* está dada en la figura 5.23, donde de manera similar al diagrama de actividad del caso de uso *Search catalog from the OPAC*, el sistema muestra el estado de ítem y brinda al usuario la posibilidad de incluir el ítem en el carrito y de realizar operaciones sobre el ítem de acuerdo a su estado.

Identificación de subdiagramas para reutilización en diagramas de actividad con variabilidades. Observando los dos diagramas de actividad con variabilidades desarrollados para los casos de uso *Search catalog from the OPAC* y *Show record in the the OPAC*, podemos observar que existen dos comportamientos repetidos que puede ser generalizados y reutilizados en los dos diagramas de actividad. En primer lugar, ambos diagramas presentan alternativas para mostrar los ítems de la biblioteca de manera similar, ya sea en vista normal o utilizando plantillas XSLT. Si bien la manera de presentar alternativas vistas varía de acuerdo a un diagrama y otro, utilizando guardas de contexto se puede generalizar adecuadamente. Además, en ambos diagramas se brinda la posibilidad de sugerir acciones para los registros, comportamiento que puede ser generalizado. Al igual que para la otra generalización, utilizando guardas de contexto este comportamiento también puede ser generalizado adecuadamente.

Además, en el diagrama de actividad para el caso de uso *Show record in the the OPAC* (figura 5.22), como ya se dijo anteriormente, el comportamiento que incluye la variabilidad *marcFlavour* se repite en dos lugares del diagrama, con lo cual puede ser generalizado y reutilizado en ambos lugares.

Otros comportamientos generalizables fueron identificados para el diagrama de actividad del caso de uso *Search catalog from the OPAC*, los cuales se repiten en otros diagramas de actividad no incluidos aquí. Se identificaron como comportamientos reutilizables: el comportamiento correspondiente a la selección de la opción *advanced search* (además se utiliza en el diagrama de actividad del caso de uso *Search catalog from the Staff Client*), el comportamiento correspondiente a la opción *command language search* (además se utiliza en el diagrama de actividad del caso de uso *Search catalog from the Staff Client*) y el comportamiento donde se indica en que lugar se deben agregar los registros (además se utiliza en el diagrama de actividad del caso de uso *Search catalog from the Staff Client*). Al no tener variación en los diagramas donde se utilizan estos comportamientos, no es necesario definir guardas de contexto para los mismos.

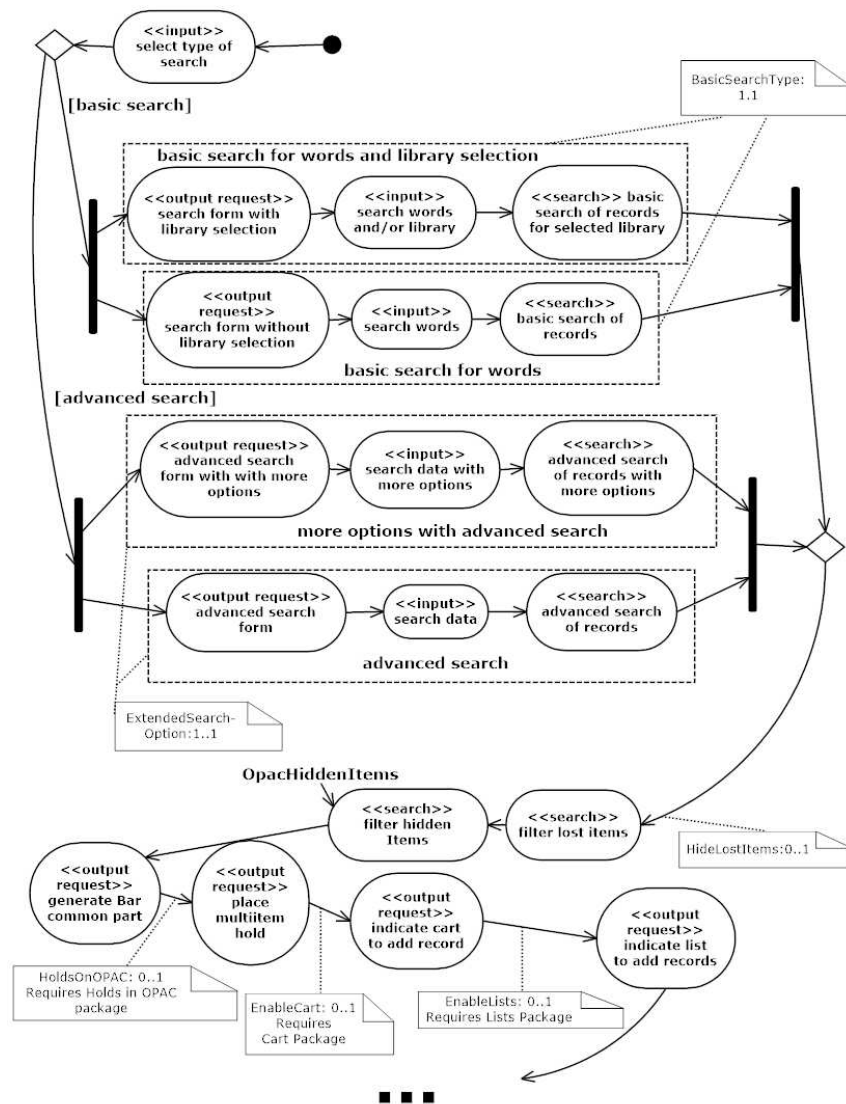


Figura 5.20: Primera parte de diagrama de actividad para el caso de uso Search catalog from the OPAC, incluido en los paquetes Public Search y Private Search del subsistema OPAC.

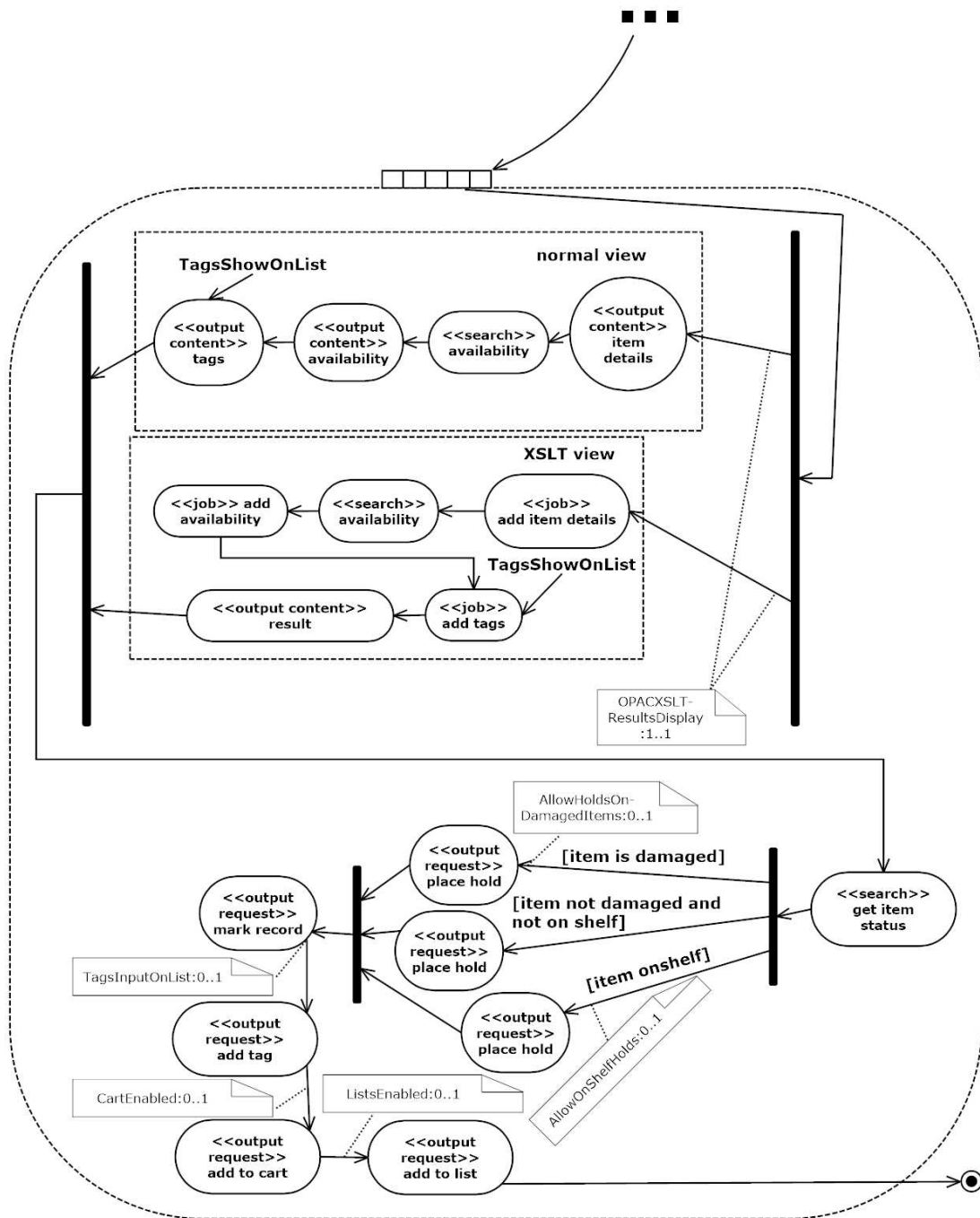


Figura 5.21: Segunda parte de diagrama de actividad para el caso de uso Search catalog from the OPAC, incluido en los paquetes Public Search y Private Search del subsistema OPAC.

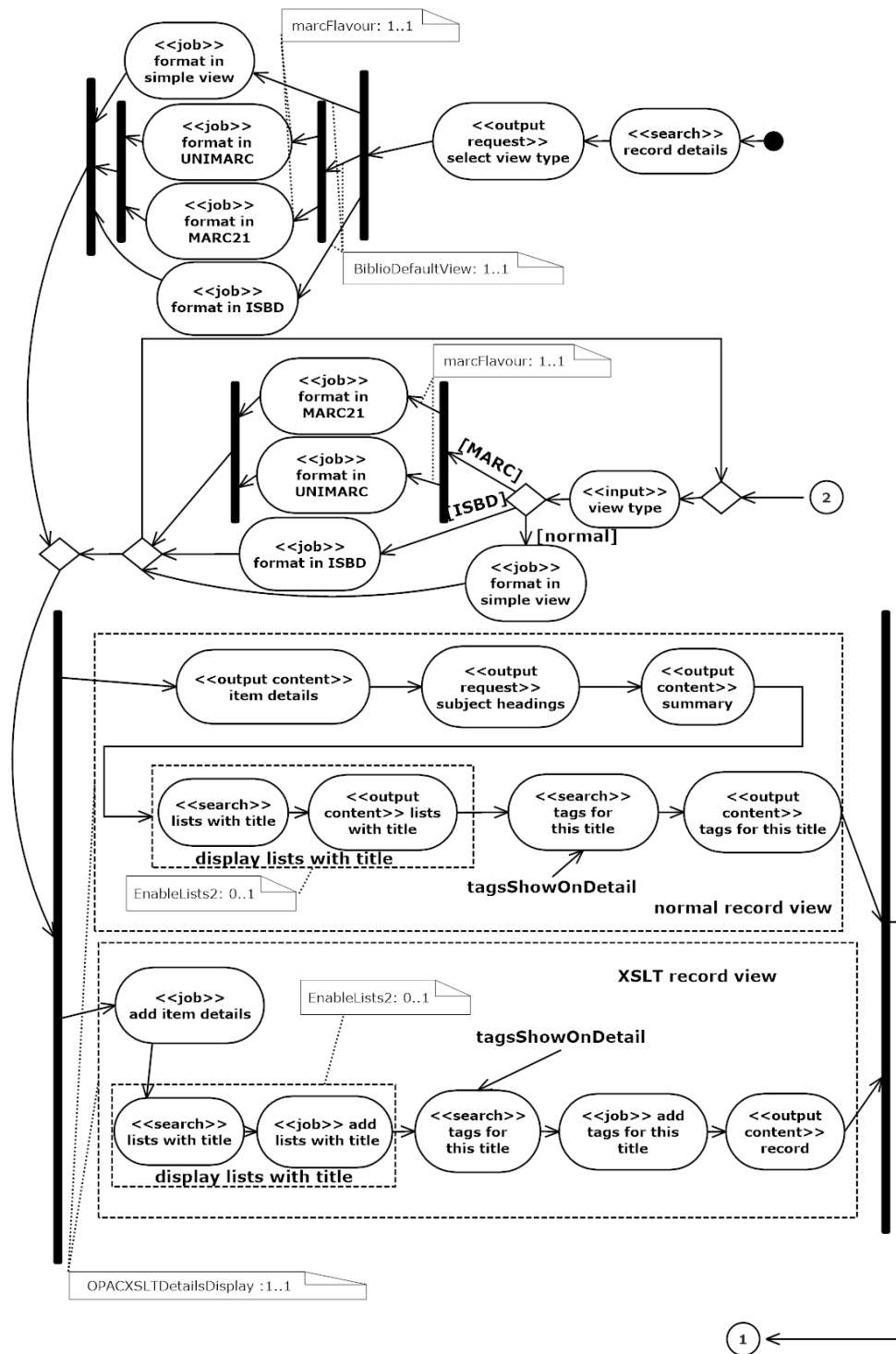


Figura 5.22: Primera parte de diagrama de actividad para el caso de uso Show record in the the OPAC del subsistema OPAC.

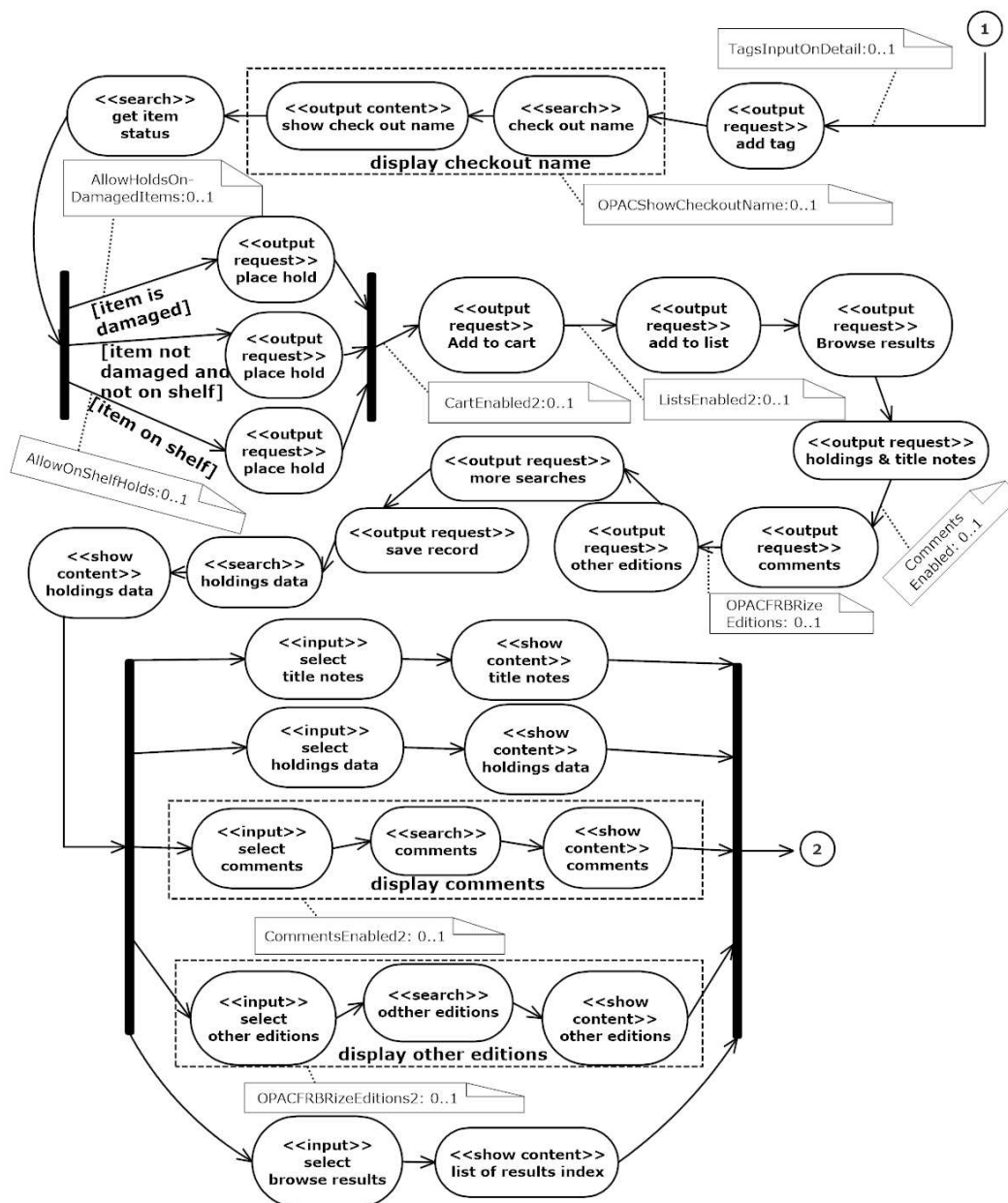


Figura 5.23: Segunda parte de diagrama de actividad para el caso de uso Show record in the the OPAC del subsistema OPAC.

Generalización y abstracción de subdiagramas para reutilización. En la figura se presentan los subdiagramas generalizados que fueron identificados en el diagrama de actividad con variabilidades para el caso de uso *Search catalog from the OPAC*. Ya que estos diagramas no presentan variaciones en su uso en los distintos diagramas en los que son reutilizados, no fue necesario definir guardas de contexto. Los subdiagramas reutilizables *Advanced Search*, *Command Language Search* e *Indicate where add records* además del diagrama de actividad para el caso de uso *Search catalog from the OPAC* son utilizados de manera idéntica en el diagrama de actividad con variabilidades del caso de uso *Search catalog from the Staff Client*, el cual es muy similar a *Search catalog from the OPAC* y por ese motivo no es incluido aquí.

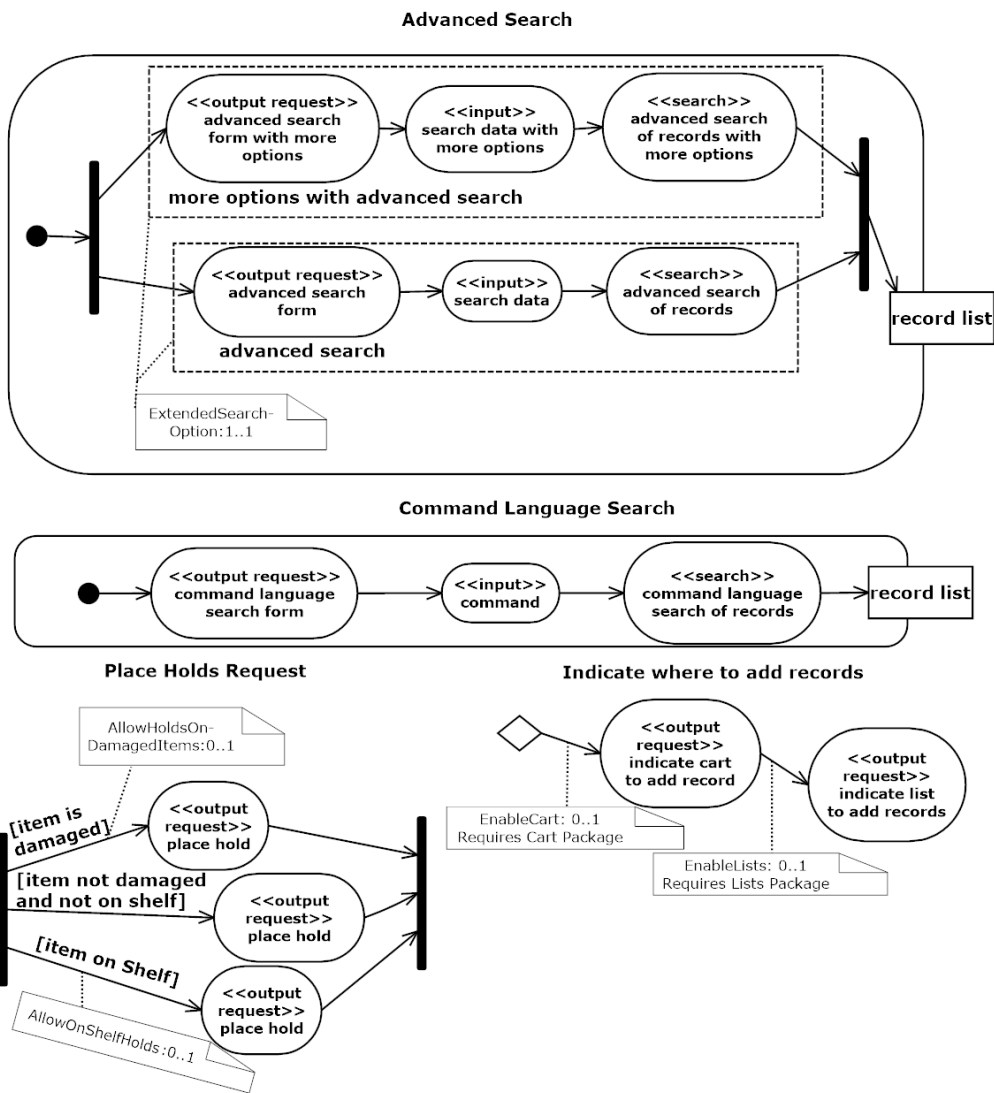


Figura 5.24: Comportamientos reutilizables generalizados, utilizados en el diagrama de actividad del caso de uso Search catalog from the OPAC y en otros diagramas de actividad (Search catalog from the Staff Client y Show record in the the OPAC).

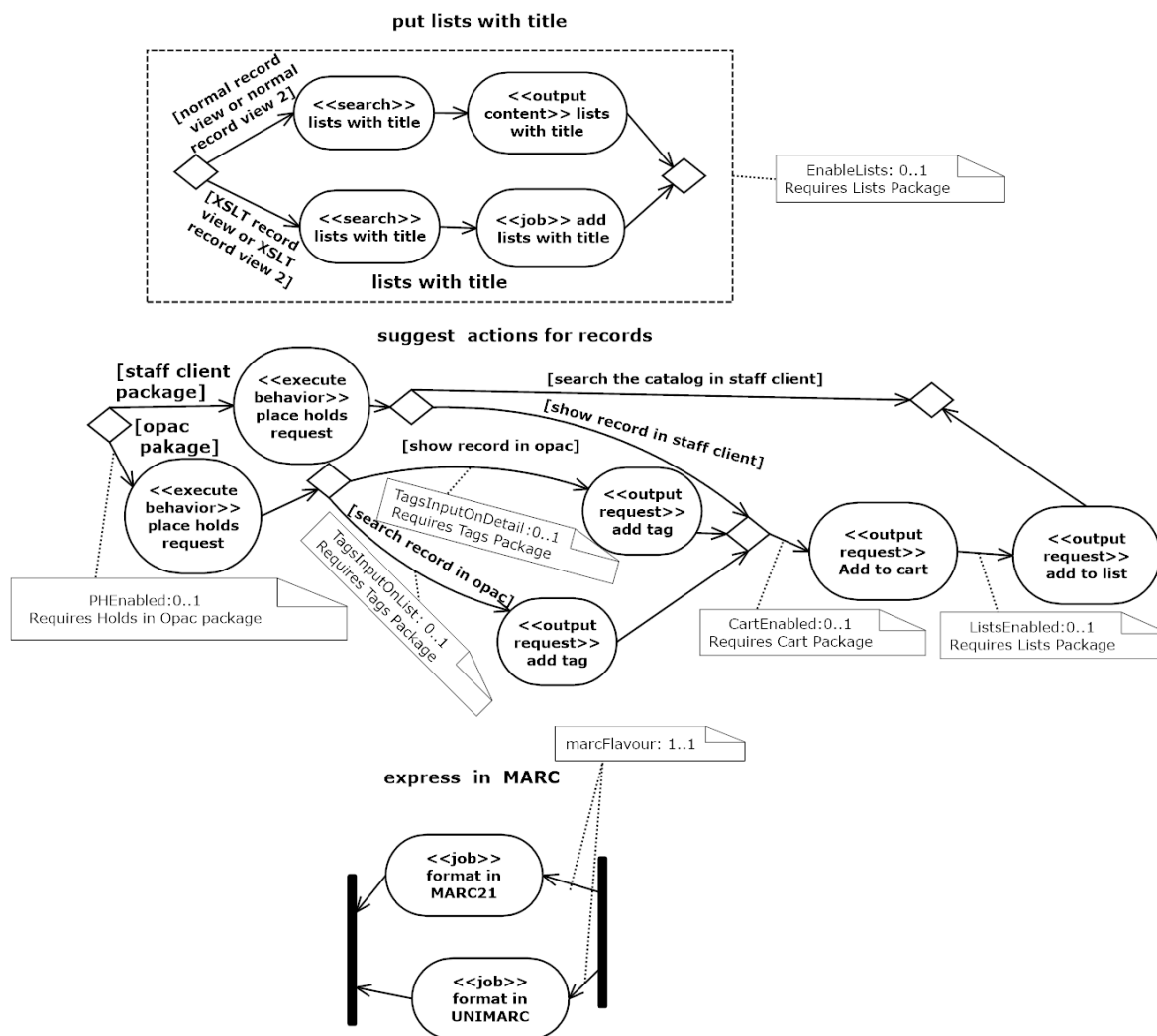


Figura 5.25: Comportamientos reutilizables generalizados, utilizados en el diagrama de actividad del caso de uso Show record in the the OPAC y en otros diagramas de actividad del caso de estudio.

En la figura 5.25 se presentan los subdiagramas generalizados que fueron identificados en el diagrama de actividad con variabilidades para el caso de uso *Show record in the the OPAC*. El subdiagrama reutilizable *express in Marc* es utilizado en dos situaciones dentro del mismo diagrama de actividad para el caso de uso *Show record in the the OPAC*, y, de manera similar es reutilizado en el diagrama de actividad para el caso de uso *Show record in the the staff client*; al no tener variaciones en sus situaciones de uso, no es necesario que incluya guardas de contexto. El subdiagrama reutilizables *put lists with title* es utilizado en situaciones donde, dependiendo del tipo de visualización del ítem (normal de registro, XSLT de registro, normal y XSLT) se deben llevar a cabo acciones distintas, dependiendo de estos contextos. El subdiagrama reutilizable *suggest actions for records* es el más rico en cuestiones de reutilización en distintos lugares e inclusión de guardas de contexto. Este subdiagrama reutilizable es utilizado en dos paquetes de casos de uso distintos (*staff client* y *OPAC*), los cuales son incluidos en la primer selección de guardas de contexto del diagrama, en base a su elección acciones distintas deben ser llevadas a cabo de acuerdo al diagrama de actividad del caso de uso donde se utilice. Hay cuatro guardas de contexto seleccionables para los casos de uso *Search the catalog in the staff client*, *Show record in the staff client* (estos dos si se seleccionó la guarda para el paquete de casos de uso *Staff client*), *Search the catalog in the OPAC*, *Show record in the OPAC* (éstos dos últimos si se seleccionó la guarda para el paquete *OPAC*).

Modificación de diagramas de actividad con variabilidades utilizando subdiagramas para reutilización. En la figura 5.26 se puede observar la versión final y acabada del diagrama de actividad

con variabilidades para el caso de uso *Search catalog from the OPAC*. Podemos observar que, a diferencia del diagrama de actividad de las figuras 5.20 y 5.21, éste diagrama de actividad se encuentra más compacto en cuestiones de espacio ya que los comportamientos modelados como subdiagramas reutilizables han sido incluidos como un único nodo del tipo Execute Behavior. Los subdiagramas reutilizables incluidos como nodos Execute Behavior son: *Advanced Search*, *Command Language Search*, *Indicate where to add records*, *put tags for this title* (en dos oportunidades) y *Suggest actions for records*.

En la figura 5.27 se puede observar la versión final y acabada del diagrama de actividad con variabilidades para el caso de uso *Show record in the OPAC*. De la misma manera que para el diagrama de actividad con variabilidades de *Search catalog from the OPAC* podemos observar que, a diferencia del diagrama de actividad de las figuras 5.22 y 5.23, éste diagrama de actividad se encuentra más compacto en cuestiones de espacio ya que los comportamientos modelados como subdiagramas reutilizables han sido incluidos con un único nodo del tipo Execute Behavior. Los subdiagramas reutilizables incluidos como nodos Execute Behavior son: *format in MARC*, *put lists with title* (en dos oportunidades), *put tags for this title* (en dos oportunidades) y *Suggest actions for records*.

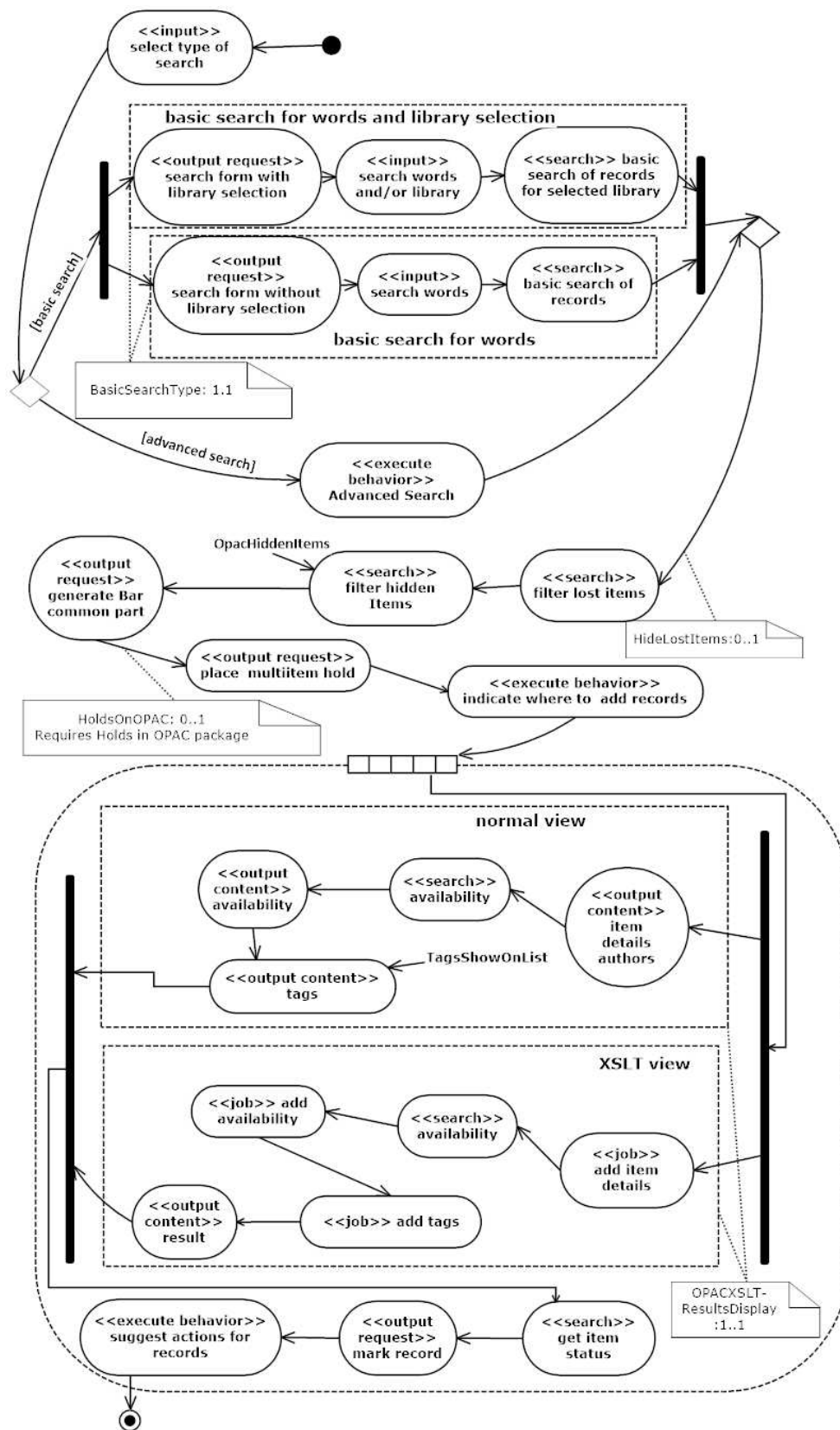


Figura 5.26: Diagrama de actividad final para el caso de uso Search catalog from the OPAC.

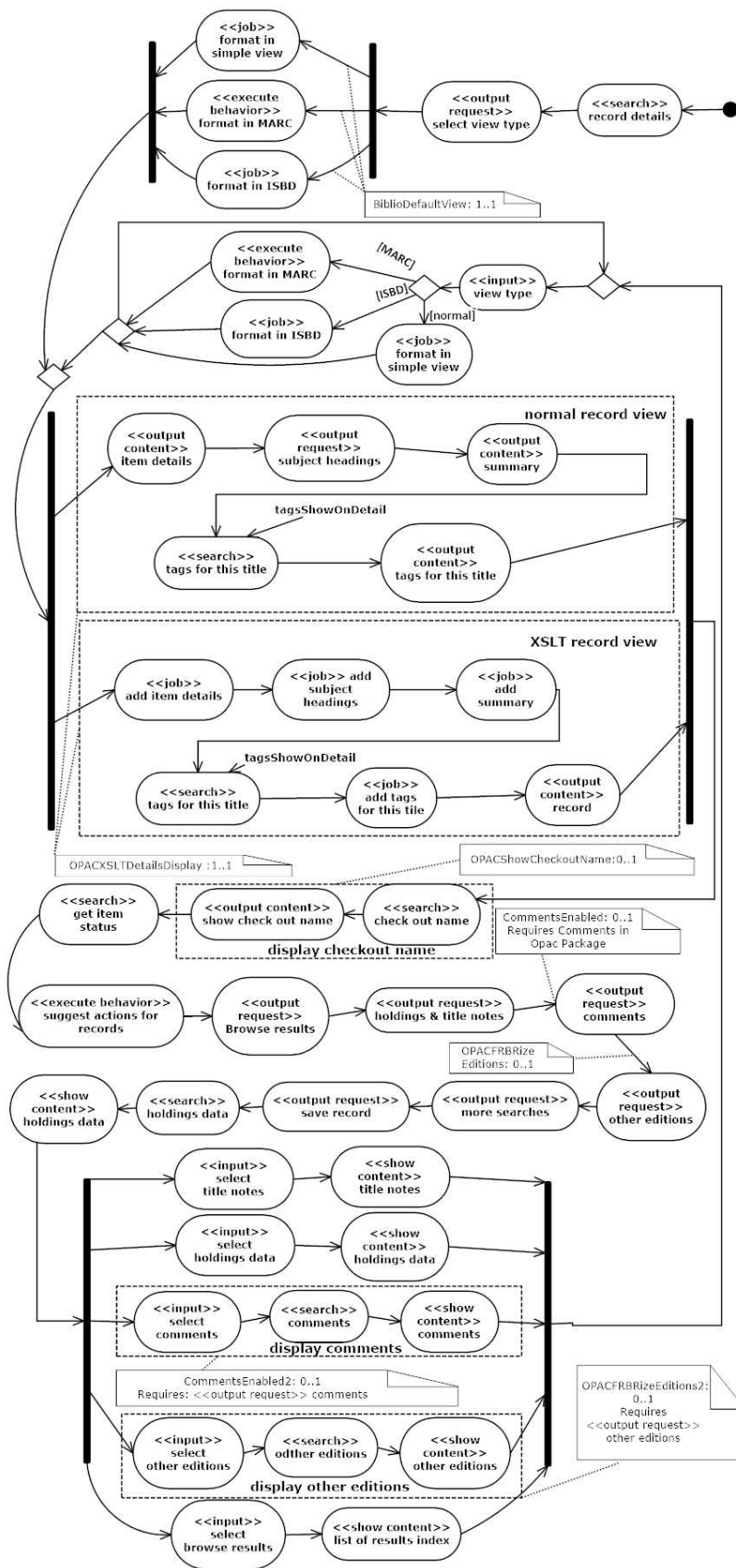


Figura 5.27: Diagrama de actividad final para el caso de uso Show record in the the OPAC.

A continuación incluiremos una serie de diagramas de actividad con variabilidades completos (a los que ya se le han aplicado las reglas de construcción de diagramas de actividad con variabilidades) pertenecientes al caso de estudio de Sistemas de administración de Bibliotecas Online.

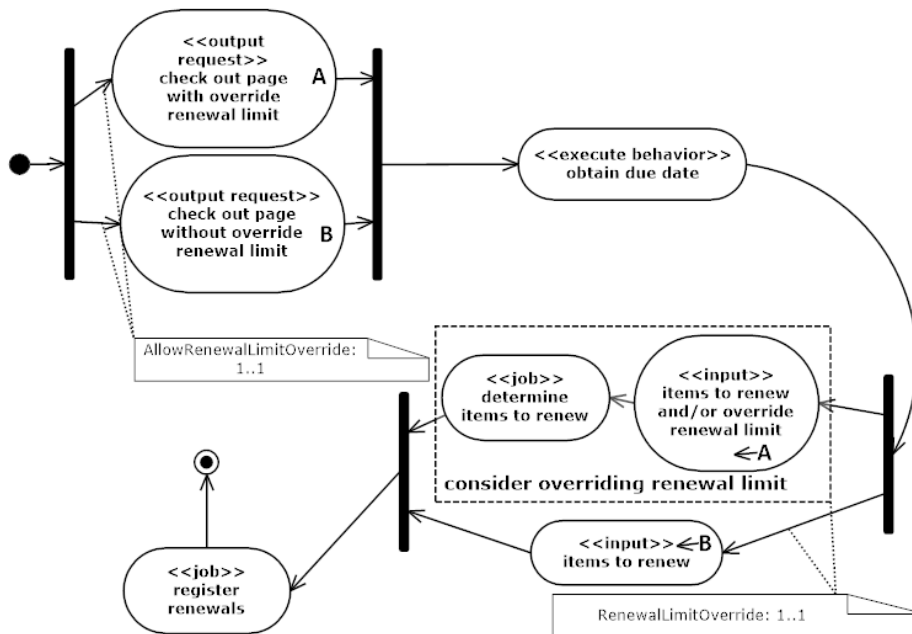


Figura 5.28: Diagrama de actividad final para el caso de uso Renewing item (usado en los subsistemas Staff client y OPAC).

El diagrama de actividad con variabilidades de la figura 5.28, correspondiente al caso de uso *Renewing item* incluye un elemento de tipo Execute Behavior llamado *obtain due date* que representa un diagrama de reutilización utilizado en distintos casos de uso del caso de estudio. Además, el diagrama contiene una situación interesante de dependencias entre variabilidades que involucran elementos dentro del mismo diagrama de actividad. La variabilidad llamada *RenewalLimitOverride* incluye elementos que dependen de elementos de la variabilidad anterior en el diagrama llamada *AllowRenewalLimitOverride*. Es decir, que es necesario haber seleccionado el elemento del que se depende para poder seleccionar el elemento que depende.

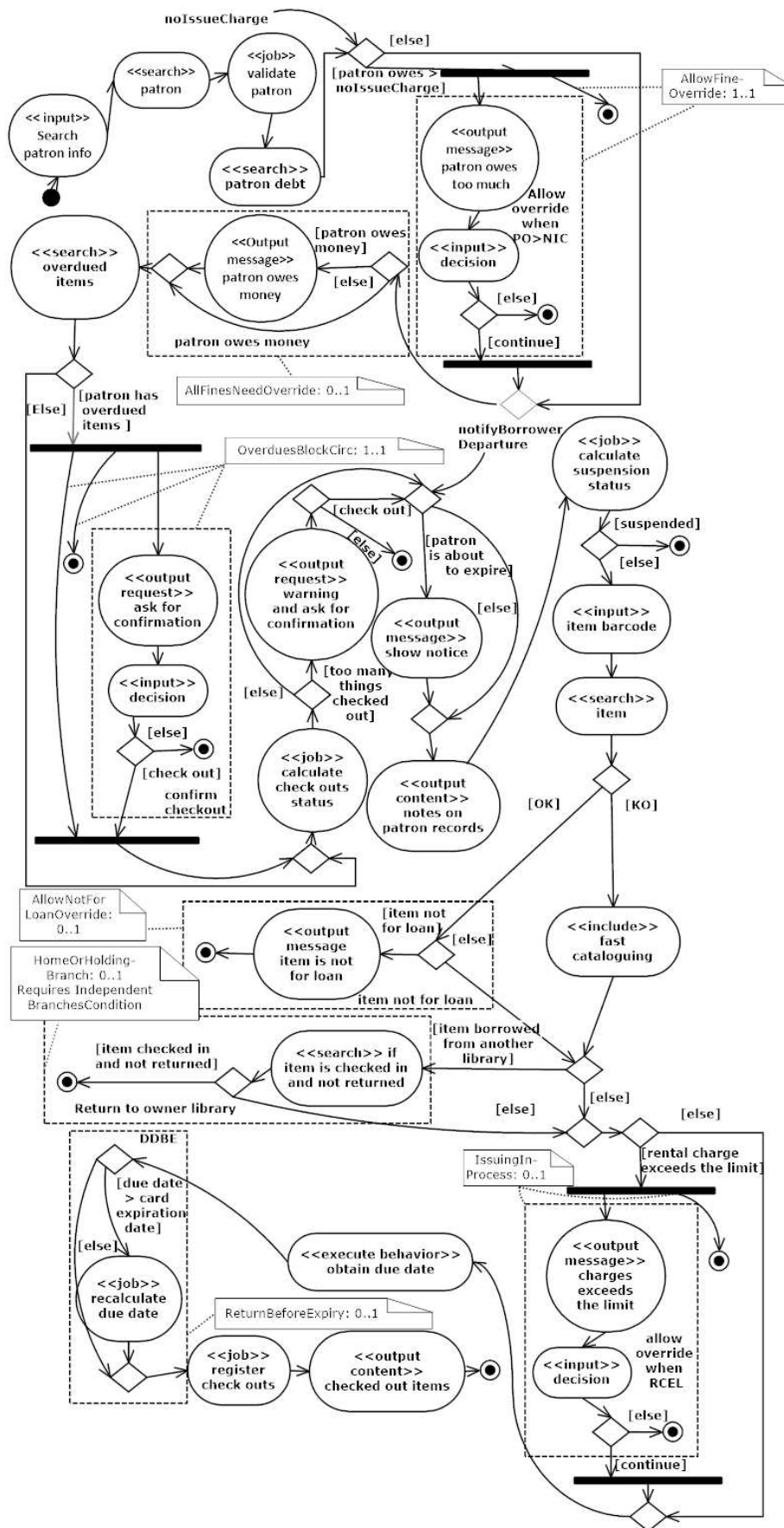


Figura 5.29: Diagrama de actividad final para el caso de uso Check Out (usado en el subsistema Staff client).

La figura 5.29 muestra el diagrama de actividad con variabilidades correspondiente al caso de uso *Check out*, del subsistema *Staff client*. En el diagrama se va especificando el proceso de check out de ítems de la biblioteca; se comienza buscando el usuario de biblioteca involucrado en el proceso, esto incluye varios chequeos de si el usuario está habilitado ya sea no posee deudas u otro tipo de sanciones. Estos chequeos incluyen variabilidades seleccionables de acuerdo al estado del usuario. Luego, se deben seleccionar los ítems involucrados en el proceso de check out, que de acuerdo a su estado distintas variabilidades deben ser seguidas. Este diagrama incluye un elemento de tipo Execute Behavior llamado *obtain due date*, que es utilizado por varios casos de uso del sistema.

La figura 5.30 muestra el diagrama de actividad con variabilidades correspondiente al caso de uso *Check in*, del subsistema *Staff client*. En el diagrama se va especificando el proceso de check in de ítems de la biblioteca, comenzando por el ingreso de los ítems, luego, de acuerdo al estado del ítem y a configuraciones de la biblioteca (modeladas con variabilidades) se van realizando las operaciones necesarias. Lo interesante de este diagrama es que involucra dos relaciones de dependencia entre elementos de diagramas de actividades y elementos de casos de uso. La variabilidad llamada *ReturnToShelvingCart* con cardinalidad 0..1 tiene asociada una relación de dependencia del tipo Requires entre la arista de actividad que apunta a la acción de tipo «job» *move items to location CART* y la asociación de casos de uso entre el actor *timer* y el caso de uso *change from CART to permanent location*. La variabilidad llamada *InProcessingToShelvingCart* con cardinalidad 0..1 tiene asociada una relación de dependencia del tipo Requires entre la arista de actividad que apunta a la acción de tipo «job» *move items with locations PROC to location CART* y la asociación de casos de uso entre el actor *timer* y el caso de uso *change from CART to permanent location*.

La figura 5.31 muestra el diagrama de actividad con variabilidades correspondiente al caso de uso *Place hold in staff client*, del subsistema *Staff client*. A diferencia que en los otros casos de uso, en éste caso de uso no se incluyen elementos del tipo Execute Behavior. El diagrama consta un conjunto de pasos de ingreso de datos y procesamiento de los mismos, y como se puede observar en la figura, lo particular de esta descripción de caso de uso es que presenta una variabilidad llamada *CanReserveFromOthersBranches* con cardinalidad 0..1, que incluye una restricción de dependencia del tipo Requires de la arista de actividad que apunta a la acción *filter items belonging to other branches* hacia el elemento de variabilidad de condición *IndependentBranchesCondition*.

La figura 5.32 muestra el diagrama de actividad con variabilidades correspondiente al caso de uso *Place hold from the OPAC*, del subsistema *OPAC*. A diferencia que en los otros casos de uso, en éste caso de uso no se incluye elementos del tipo Execute Behavior. Consta un conjunto de pasos de ingreso de datos y procesamiento de los mismos, y como se puede observar en la figura, lo particular de este diagrama es que presenta una variabilidad llamada *CanReserveFromOthersBranches* con cardinalidad 0..1, que incluye una restricción de dependencia del tipo Requires de la arista de actividad que apunta a la acción *filter items belonging to other branches* hacia el elemento de variabilidad de condición *IndependentBranchesCondition*.

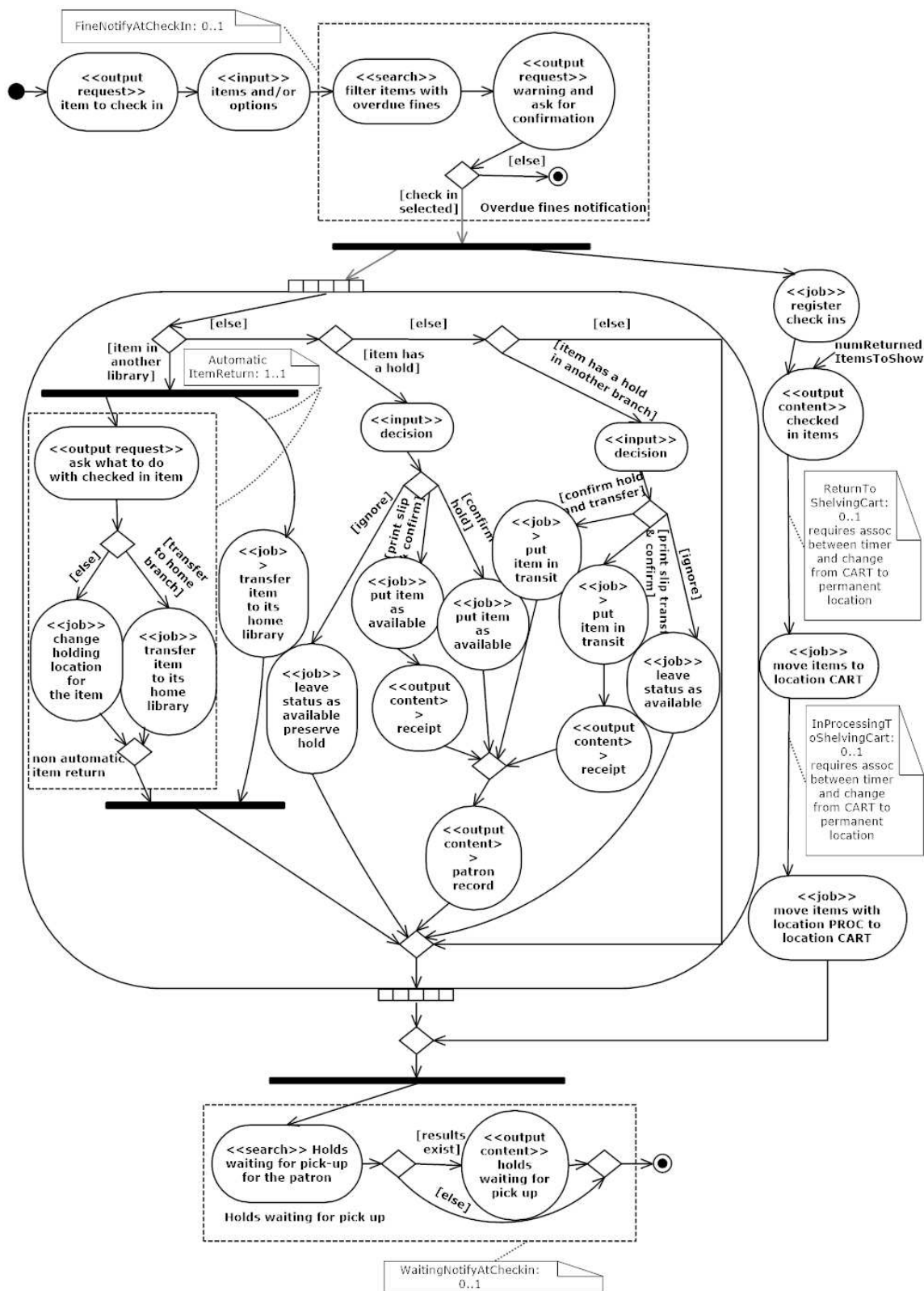


Figura 5.30: Diagrama de actividad final para el caso de uso Check In (usado en el subsistema Staff client).

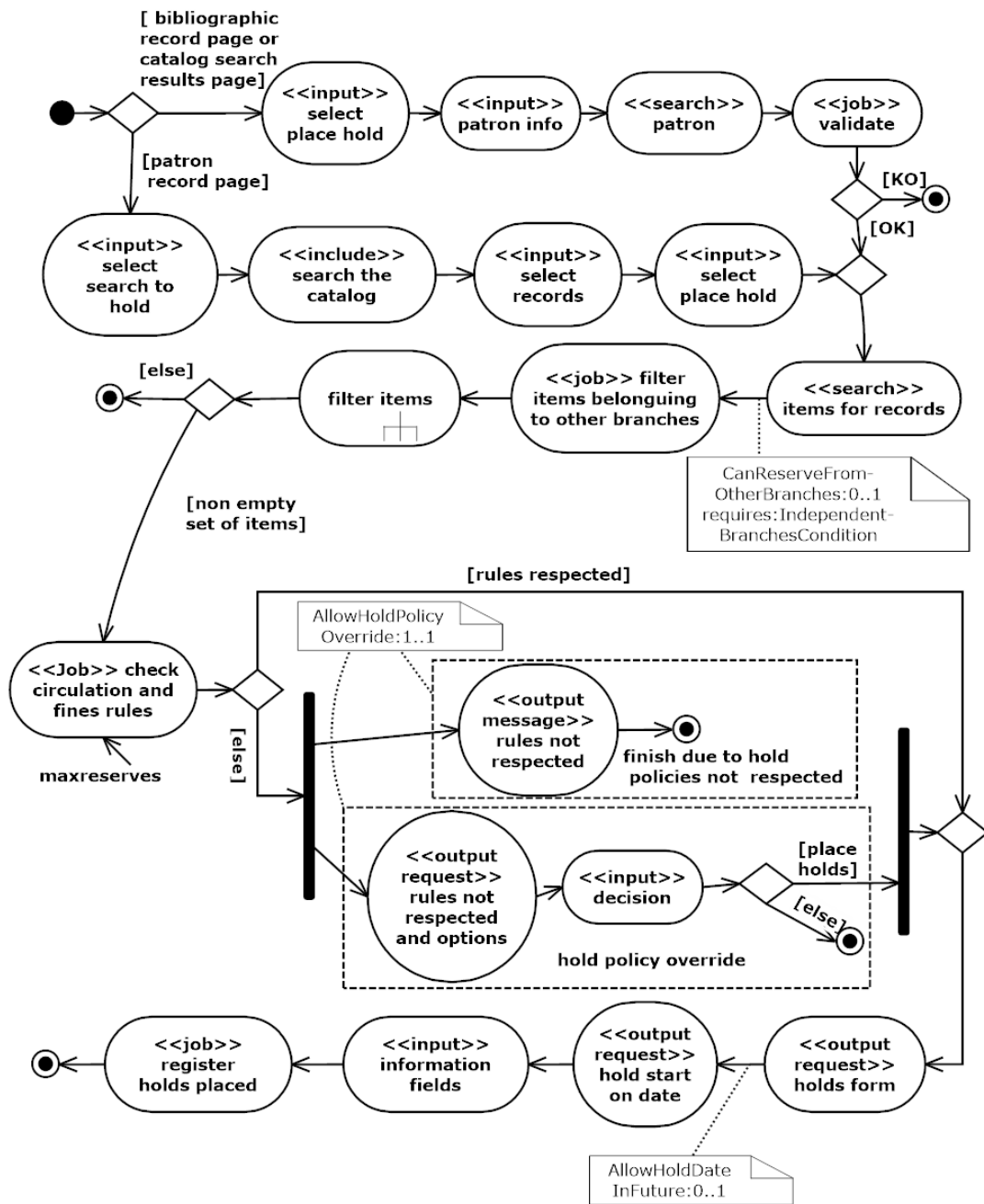


Figura 5.31: Diagrama de actividad final para el caso de uso Place Hold in staff client.

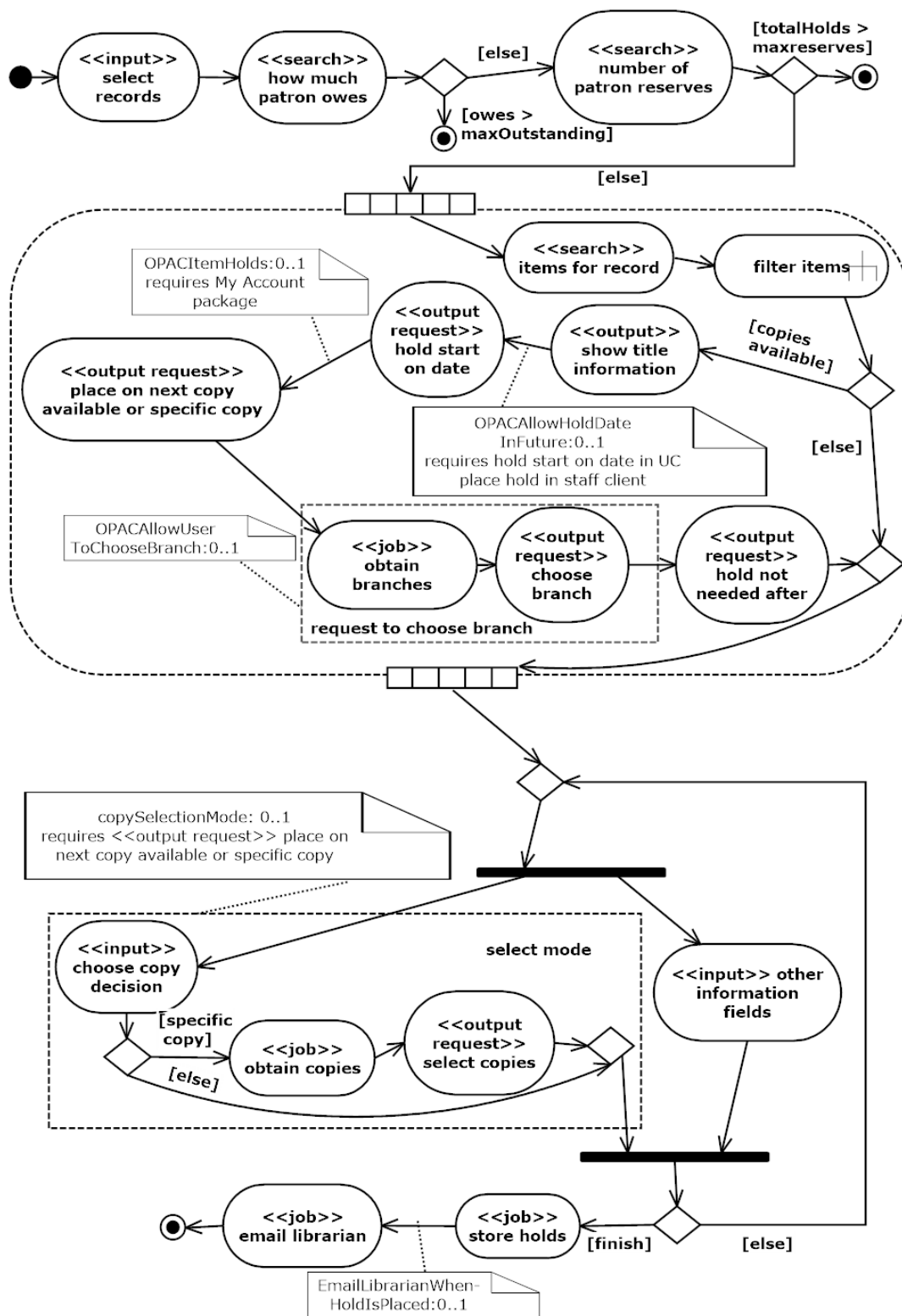


Figura 5.32: Diagrama de actividad final para el caso de uso Place Hold from the OPAC.

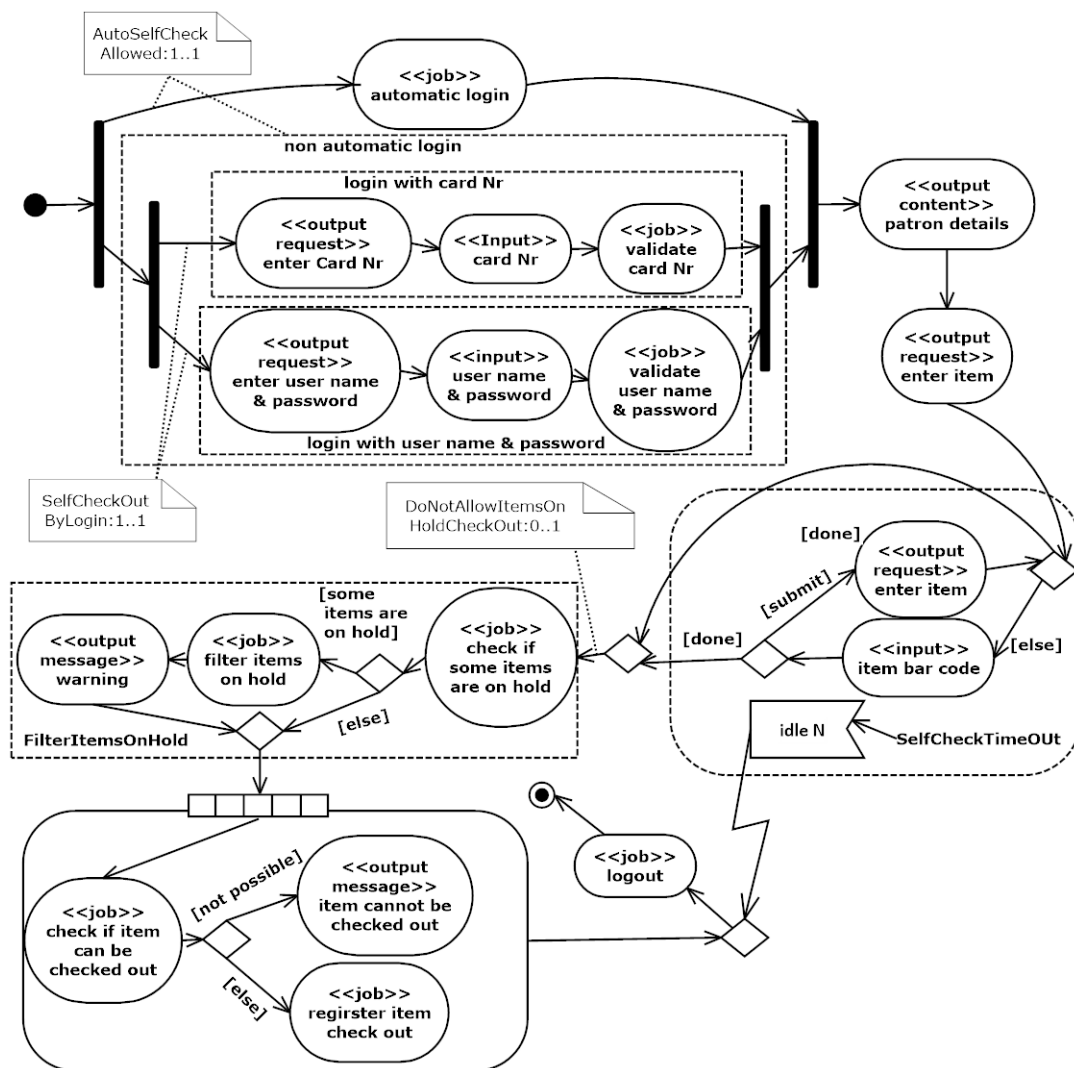


Figura 5.33: Diagrama de actividad final para el caso de uso Self-Checkout, del subsistema Staff client.

La figura 5.33 muestra el diagrama de actividad con variabilidades correspondiente al caso de uso *Self-Checkout*, del subsistema *Staff client*. Al iniciar la descripción del caso de uso, hay dos tipos de posibilidades para ingresar en el sistema, login automático o login no automático, que son modeladas a través de la variabilidad *AutoSelfCheck* con cardinalidad 1..1; una de estas dos posibilidades (para login no automático) a su vez presenta dos alternativas que son utilizando tarjeta Nr o ingresando nombre de usuario y contraseña, estas dos alternativas son modeladas con la variabilidad *SelfCheckoutByLogin* con cardinalidad 1..1. Luego, el sistema muestra los datos del patrón y solicita ingresar los datos de ítem, el sistema tiene un tiempo límite de espera para ingresar los datos del ítem, transcurrido este tiempo se finaliza la ejecución del caso de uso, esto es modelado con el evento *idle N*. Para finalizar el caso de uso se realizan dos operaciones, la primera es opcional (a través de la variabilidad con cardinalidad 0..1 *DoNotAllowItemsOn HoldCheckOut*) y se trata de permitir colocar ítems en espera. Finalmente, el sistema chequea si el ítem es apto para check out, y en caso afirmativo realiza la operación.

Capítulo 6

Generación automática de modelo de features a partir de modelos de requisitos de dominio

En esta sección se presentan las dos transformaciones utilizadas para generar el modelo de features de la familia de aplicaciones web a partir de los modelos de dominio de requisitos en *UML*. La primera transformación llamada *UCD2FM* transforma los diagramas de casos de uso con variabilidades a un modelo de features inicial de la familia de aplicaciones web. Con el modelo de features inicial construido, se aplica la segunda transformación, denominada *Reqs2FM*; la cual adjunta features provenientes de diagramas de actividad con variabilidades al modelo de features inicial.

6.1. Mapeando diagramas de casos de uso de dominio a modelo de features inicial

6.1.1. Background

Como hemos mencionado en la sección 1.3 del capítulo de Introducción, durante la ingeniería de dominio tanto los modelos de features como los modelos de requisitos (en particular diagramas de casos de uso de UML) son necesarios, ya que ambos modelos son utilizados para distintas finalidades. Sin embargo, el desarrollo en paralelo de ambos tipos de modelos pueden causar numerosas inconsistencias, debido a que, si los modelos de casos de uso de UML incluyen variabilidades, éstas deben tener una correspondencia total con las variabilidades especificadas en el modelo de features.

Para lidiar con el problema de inconsistencias entre modelo de features y modelo de casos de uso de dominio, en [Bragança, 2007] se automatiza esta tarea generando modelo de features a partir de modelo de casos de uso de UML de dominio. Pero, como hemos mencionado en la sección 3.1.1, la notación de modelos de casos de dominio de [Bragança, 2007] presenta, entre otros, algunos problemas importantes: se introducen casos de uso “extra” no interesantes y no se consideran variabilidades cuyos variantes representan subsistemas o sub-subsistemas enteros involucrando casos de uso. Como consecuencia de estos problemas, para que los modelos de features generados de manera automática tengan una buena calidad en cuestiones de modularidad y comprensibilidad, es necesario que los casos de uso de dominio sean definidos con algunas restricciones, introduciendo casos de uso “extra” y sin la utilización de paquetes de casos de uso. Esto puede generar errores y distintas interpretaciones en la construcción de modelos de casos de dominio, ya que no debería ser exigible que los encargados de definir estos modelos tengan en cuenta tales restricciones.

En este trabajo hemos atacado el problema de producir modelo de casos de uso de dominio con restricciones teniendo en cuenta una notación de casos de dominio (ver sección 3.1.3) que suple las carencias identificadas en la notación de [Bragança, 2007], y en general en todas las notaciones halladas para casos de uso de dominio. En base a la notación definida para casos de uso de dominio, proponemos

definir una transformación de modelos que genere un modelo de features con buena calidad en cuestiones de modularidad y comprensibilidad.

Esta sección presenta una transformación de modelos de casos de uso de dominio a modelo de features realizada en ATL que tiene en cuenta paquetes de casos de uso para generar modelo de features.

6.1.2. Trabajo relacionado

Para la generación de modelos de features a partir de diagramas de casos de uso se consideran como trabajos relacionados a los enfoques que consideran automatización completa para generar modelo de features a partir diagramas de casos de uso de dominio. Teniendo en cuenta esta consideración, solo hemos considerado como relacionado el trabajo de [Bragança, 2007]. La transformación de modelos realizada en [Bragança, 2007], a diferencia de este trabajo, fue realizada en QVT. Como principal distinción de la transformación realizada aquí con la transformación de [Bragança, 2007], se puede decir que nuestro trabajo tiene en cuenta paquetes de casos de uso, lo cual agrega complejidad y sofisticación debido a que es necesario tener en cuenta paquetes anidados, variabilidad entre paquetes, casos de uso y relaciones dentro de paquetes y relaciones de dependencia que involucran paquetes.

6.1.3. Reglas de transformación

La tabla de la figura 6.1 define la propuesta de transformación para mapeo de diagramas de casos de uso con variabilidades a modelo de features inicial, llamada *UCD2FM*. Se toman como entradas elementos del modelo de casos de uso con variabilidades y dan como resultado elementos de modelo de features.

A partir de ahora, se utilizará la siguiente nomenclatura:

UC significa Use Case (caso de uso),

$N_1, \dots, N_k : M$ significa instancias de meta-clases *M* con nombres N_1, \dots, N_k ,

la relación *n..m* hace referencia a una relación con `minCardinality:n` y `maxCardinality:m`,

V: Variability: *n..m* expresa un elemento Variability con nombre *V*, min *n* y max *m*,

\notin U:Feature significa que el modelo de feature que se está construyendo no contiene U:Feature.

Rule	maps	onto
0	D name of use case diagram	D: Feature
1	Non variant association between A: Actor and U: UC, U is not inside a package.	SingleRelation:1..1 between D:Feature and U:Feature
2	Non variant P: Package that is not inside another package.	SingleRelation:1..1 between D:Feature and P:Feature
3	U: UC. U is inside a P: Package. U base UC.	SingleRelation:1..1 between P:Feature and U:Feature
4	Non variant P': package inside P: package	SingleRelation:1..1 between P:Feature and P':Feature
5	V: AssocVariability: $n..m$ Variants: associations from A: Actor to U_1, \dots, U_k : UC.	If U_1, \dots, U_k inside P package. Then Relation: $n..m$ between P:Feature and U_1, \dots, U_k : Feature Else Relation: $n..m$ between D:Feature and U_1, \dots, U_k : Feature
6	Extend from U': UC to U: UC that is not a variant.	SingleRelation:1..1 between U:Feature and if $\exists U'$: Feature then U': Feature else Reference to U': Feature
	Include from U: UC to U': UC that is not a variant.	SingleRelation:1..1 between U:Feature and if $\exists U'$: Feature then U':Feature else Reference to U': Feature
7	V: ExtendVariability: $n..m$ Variants are Extend from U_1 : UC to U: UC Extend from U_k : UC to U: UC	Relation: $n..m$ between U:Feature and if $\exists U_1$:Feature then U_1 :Feature else Reference to U_1 :Feature ... if $\exists U_k$: Feature then U_k : Feature else Reference to U_k : Feature
	V: IncludeVariability: $n..m$ Variants are Include from U: UC to U_1 : UC Include from U: UC to U_k : UC	Relation: $n..m$ between U:Feature and if $\exists U_1$:Feature then U_1 :Feature else Reference to U_1 :Feature ... if $\exists U_k$: Feature then U_k : Feature else Reference to U_k : Feature
8	V: PackageVariability: $n..m$ Variants: P_1, \dots, P_k : Package.	If P_1, \dots, P_k inside P package. Then Relation: $n..m$ between P:Feature and P_1, \dots, P_k : Feature Else Relation: $n..m$ between D:Feature and P_1, \dots, P_k : Feature
9	V: ExtendVariability2: $n..m$ Variants are Extend from: U UC to U_1 : UC Extend from U: UC to U_k : UC	SingleRelation: 1..1 between U:Feature and V:Feature, and Relation: 1..1 between V: Feature and if $\exists U_1$:Feature then U_1 :Feature else Reference to U_1 :Feature ... if $\exists U_k$: Feature then U_k : Feature else Reference to U_k : Feature
10	DC of type T between variants N and N'	DC of type T between <i>Feature for N</i> and <i>Feature for N'</i>

Figura 6.1: Reglas de transformación de modelo de casos de uso a modelo de features.

Para la regla 10 de la figura 6.1 T puede ser del tipo *Requires* o *Excludes* y *Feature for* (Feature para) se define de la siguiente manera:

- Feature para asociación entre actor A y caso de uso $U = U$
- Feature para «include» entre casos de uso U y $U' = U'$
- Feature para «extend» entre casos de uso U' y $U = U'$
- Feature para paquete de casos de uso $P = P$

Ejemplo. La figura 3.30 de la sección 3.4.2 (*Notación propuesta para modelo de features*) muestra el modelo de features resultado de la transformación llamada *UCD2FM* aplicada a los diagramas de casos de uso de las figuras 5.13 y 3.5 (paquetes de casos de uso para subsistema *OPAC* y paquetes de casos de uso *My account*). Usando la regla 0 se crea una feature raíz para el sistema de manejo de bibliotecas online, llamada *Integrated Library Management System*. Usando la regla 2, el paquete de casos de uso *OPAC* es mapeado a una relación simple con cardinalidad 1..1 entre la feature raíz *Integrated Library Management System* y la feature correspondiente al paquete de casos de uso *OPAC* (con el mismo nombre). Usando la regla 8, la variabilidad de paquete *PublicSearch?: 1..1* es mapeada a una relación con cardinalidad 1..1 entre la feature *OPAC* y las features correspondientes a los casos de uso *Public Search* y *Private Search*. Usando la regla 5 la variabilidad de asociación *TopIssue: 0..1* es mapeada a una relación simple con cardinalidad 0..1 entre la feature *OPAC* y la feature correspondiente al caso de uso *See most checked out items*. Usando la regla 6 (segunda fila) la dependencia del tipo «include» entre el caso de uso *show account summary* y el caso de uso *see checked out items* es mapeada a una relación simple con cardinalidad 1..1 entre la feature *show account summary* y la feature correspondiente al caso de uso *see checked out items*. Usando la regla 7 (primera fila), la variabilidad de tipo extend *MySummary: 0..1* es mapeada a una relación simple con cardinalidad 0..1 entre la feature *see checked out items* y la feature correspondiente al caso de uso *show links*. Usando la regla 3, el caso de uso *show fines summary* (dentro del paquete *Fines*) es mapeada a una relación simple con cardinalidad 1..1 entre la feature correspondiente al paquete de casos de uso *Fines* y la feature asociada al caso de uso *show fines summary*. Usando la regla 7 (segunda fila) la variabilidad de tipo include *Fines Allowed: 0..1* es mapeada a una relación simple con cardinalidad 0..1 entre la feature *show account summary* y la referencia a la feature *show fines summary*. Usando la regla 10, la restricción de dependencia entre la relación «include» (entre los casos de uso *show account summary* y *show fines summary*) y el paquete de casos de uso *Fines* es mapeada a una restricción de dependencia del mismo tipo entre las features *show fines summary* y *Fines*. Usando la regla 9, la variabilidad *createItemAccess: 1..1* de la figura 3.6 es mapeada a un elemento *singleRelation: 1..1* entre las features *Add item* y *createItemAccess* y a un elemento *Relation: 1..1* entre la feature *createItemAccess* y las features *add record*, *receive order* y *close basket*.

6.2. Mapeando diagramas de actividad de dominio a modelo de features de requisitos

6.2.1. Background

Para desarrollar una familia de aplicaciones web con suficiente nivel de detalle y granularidad en cuestiones de variabilidad, es necesario que los casos de uso sean descritos con más detalle en alguna notación de dominio. Para dicho fin, en este trabajo hemos definido una notación de diagramas de actividad de UML de dominio (ver sección 3.2.3), la cual durante su desarrollo puede introducir nuevas consideraciones de variabilidad, a un nivel más concreto. Ya que consideramos que todas las variabilidades de la familia de aplicaciones deberían ser incluidas en el modelo de features de la familia de aplicaciones web, para que puedan ser validadas por el cliente y para facilitar el proceso de configuración de modelos de dominio, es necesario trasladar las variabilidades definidas en los diagramas de actividad de dominio al modelo de features de la familia de aplicaciones.

No hemos hallado ningún enfoque en la bibliografía que genere de manera totalmente automática features a partir de diagramas de actividad de dominio. Solo hemos hallado un enfoque (ver [Varela, 2011]), que a partir de requisitos descritos en una notación textual usando escenarios genera modelo de features, pero no de manera completamente automática.

En esta sección nos proponemos, en base a la notación definida para diagramas de actividad de dominio, definir una transformación de modelos que genere features que representen a las variabilidades más importantes de los diagramas de actividad de dominio que describen los casos de uso de dominio.

Esta sección presenta una transformación de modelos de diagramas de actividad de dominio a features realizada en ATL, donde las features son organizadas jerárquicamente a partir de la feature correspondiente al caso de uso de dominio que el diagrama de actividad describe.

6.2.2. Trabajo relacionado

Hemos considerado como relacionados a los enfoques que mapean requisitos de una familia de aplicaciones expresados en alguna notación usando escenarios (que pueden ser usados para describir casos de uso) a modelo de features. Solo hemos hallado el trabajo de [Varela, 2011] que cumple este requisito.

En [Varela, 2011], en base a la construcción de plantillas de intereses (escenarios de requisitos funcionales) se aplican un conjunto de heurísticas de procesamiento de lenguaje natural para derivar modelos de features teniendo en cuenta la identificación de intereses transversales (como aspectos) en los niveles de ingeniería de dominio y aplicación. Además este enfoque incluye una herramienta para identificar y generar de manera sistemática y automática features comunes y variables y utiliza procesos que utilizan un método multi-criterio para identificar y resolver conflictos.

A diferencia del trabajo realizado para esta tesis, [Varela, 2011] no genera el modelo de features de manera completamente automática y no tiene en cuenta algunos asuntos propios de la notación de diagramas de actividad de dominio definida aquí, tales como: identificación de variantes que representan diferentes tipos de objetos de flujo de datos, representación en features de conjuntos parámetros, representación en features de elementos que consideran reutilización de comportamientos.

6.2.3. Reglas de transformación

La tabla de la figura 6.2 define la propuesta de transformación *Reqs2FM*, utilizada para mapear diagramas de actividad con variabilidades (usados para describir casos de uso) a modelo de features de requisitos.

A partir de ahora, se utilizará la siguiente nomenclatura, CBA: para acciones de tipo *callBehaviourAction*, EB: para nodos execute behavior. Sean C_1, \dots, C_n elementos dentro de un diagrama de actividad o un diagrama de comportamiento que necesitan ser mapeados a features en el modelo de features, se define su padre (parent) en el modelo de features de la siguiente manera:

Parent(C_1, \dots, C_n) =*def* Si C_1, \dots, C_n están en la actividad de C: CBA entonces C:Feature

si no y si C_1, \dots, C_n están en la actividad de B: EB entonces B: feature

si no y si C_1, \dots, C_n en G: IBD entonces G: Feature

si no U:Feature donde C_1, \dots, C_n están en la actividad de U, caso de uso

Es necesario saber cuáles acciones obligatorias (que no son variantes) de un diagrama de actividad deben ser mapeadas a features con su mismo nombre. Mapear todas las acciones obligatorias puede ser demasiado (es decir, se puede producir una explosión de features). Para algunos dominios una posibilidad es mapear a features del modelo de features todas las acciones de un conjunto de estereotipos. Para este propósito, se incluye la regla 1.a.

En este trabajo se considera el caso para el dominio de aplicaciones web, para el cual se definió un perfil de UML para describir casos de uso utilizando diagramas de actividad (ver sección 2.1.1). Teniendo en cuenta el perfil definido para describir casos de uso de aplicaciones web se consideraron que los estereotipos importantes de acciones a ser mapeadas son: «search» y «job», debido a que estos tipos de acciones representan actividades llevadas a cabo por el sistema y que pueden representar acciones importantes.

Rule	maps	onto	
1	a	E:ExecutableNode that is not a variant, and can be of stereotype: «job», «search»	SingleRelation:1..1 between Parent(E) and if $\exists E:Feature$ then E:Feature else Reference to E:Feature
	b	E:ExecutableNode that is not a variant, and is either a CBA or an EB	SingleRelation:1..1 between Parent(E) and if $\exists E:Feature$ then E:Feature else Reference to E:Feature
2	a	V: Control flow variability: 0..1. Variant: an ActivityEdge to E:ExecutableNode.	SingleRelation: 0..1 between Parent(E) and E:Feature
	b	V: Control flow variability: 0..1. Variant: G: IBD.	SingleRelation: 0..1 between Parent(G) and G:Feature
3	V: Control flow variability: n..m Variants: activity edges to executable nodes E_1, \dots, E_k : ActivityNode and G_1, \dots, G_j : IBD.	GroupRelation: n..m between Parent($E_1, \dots, E_k, G_1, \dots, G_j$) and $E_1, \dots, E_k, G_1, \dots, G_j$: Features	
4	V: Data flow variability: 0..1 Variant: O: ObjectNode.	SingleRelation: 1..1 between Parent(O) and V:Feature. SingleRelation: 0..1 between V:Feature and O:Feature	
5	V: Data flow variability: n..m, Variants: O_1, \dots, O_k : ObjectNode.	SingleRelation: if $n > 0$ then 1..1 else 0..1 between Parent (O_1, \dots, O_k) and V:Feature. GroupRelation: n..m between V: Feature and O_1, \dots, O_k : Feature	
6	P: Parameter set U: activity of use case containing P	SingleRelation: 1..1 between U: Feature and and P: Feature	
7	A DC of type T between variants named N and N'	DC of type T between Feature for N and Feature for N'.	

Figura 6.2: Reglas de transformación de diagramas de actividad con variabilidades a modelo de features.

Para la regla 8 de la figura 6.2, T puede ser del tipo *Requires* o *Excludes* y *Feature for* (Feature para) se define de la siguiente manera:

- Feature para asociación entre actor A y caso de uso U = U
- Feature para «include» entre casos de uso U y U' = U'
- Feature para «extend» entre casos de uso U' y U = U'
- Feature para paquete de casos de uso P = P
- Feature para object node O = O
- Feature para arista de actividad E apuntando a nodo ejecutable N = N.
- Feature para IBD I = I
- Feature para condición C = C.

Ejemplos. La figura 6.3 muestra un modelo de features para la descripción del caso de uso *renewing* (ver figura 3.14). Usando la regla 1 se mapea la acción de tipo «job» *register renewals* a una relación simple con cardinalidad 1..1 entre las features *Renewing* (correspondiente al caso de uso) y *obtain due date*. Usando la regla 3, la variabilidad de flujo de control *RenewalLimitOverride*: 1..1 es mapeada a un elemento GroupRelation: 1..1 con el mismo nombre, entre la feature *Renewing* y las features para la acción *for input items to renew* y el IBD *consider overriding renewal limit*. Usando la regla 1 se mapea a la acción de tipo «job» *determine items to renew* que se encuentra adentro del IBD *consider overriding renewal*

limit a un elemento singleRelation: 1..1 entre las features determine *items to renew* y *consider overriding renewal limit*. Usando la regla 3 se mapea la variabilidad de flujo de control *specifyDueDate*: 1..1, dentro del «execute behavior» *obtain due date*, a un elemento GroupRelation:1..1 entre la feature *obtain due date* y las features para el IBD *enter due date* y el IBD *due date calculation*. Usando nuevamente la regla 3, se mapean las variabilidades de flujo de control con nombre *renewalPeriodBase* y *useDaysMode* en relaciones de grupo con el mismo nombre.

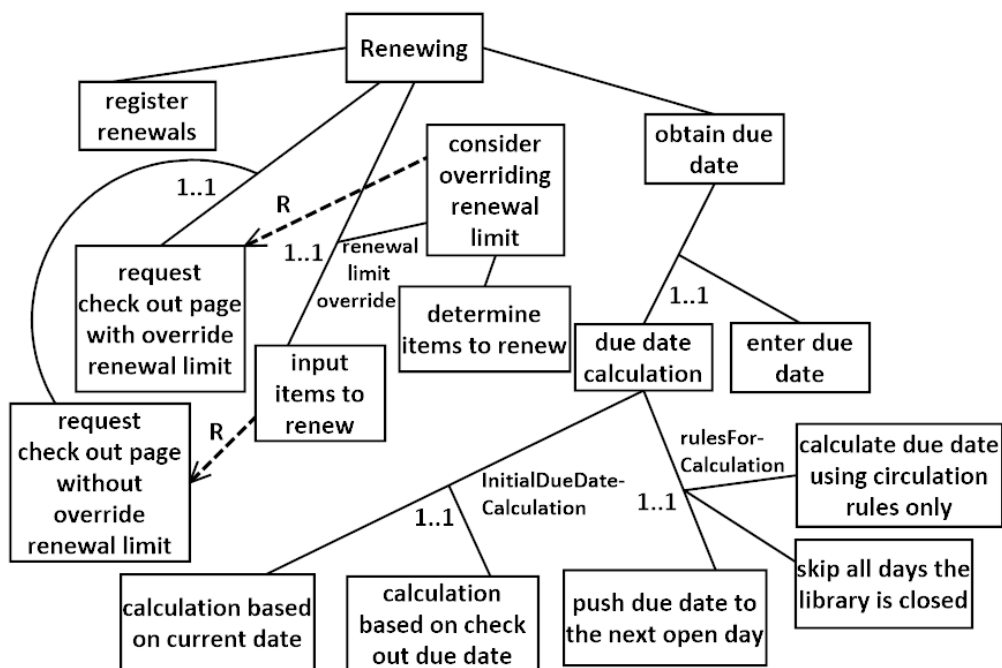


Figura 6.3: Modelo de features para descripción del caso de uso *Renewing*.

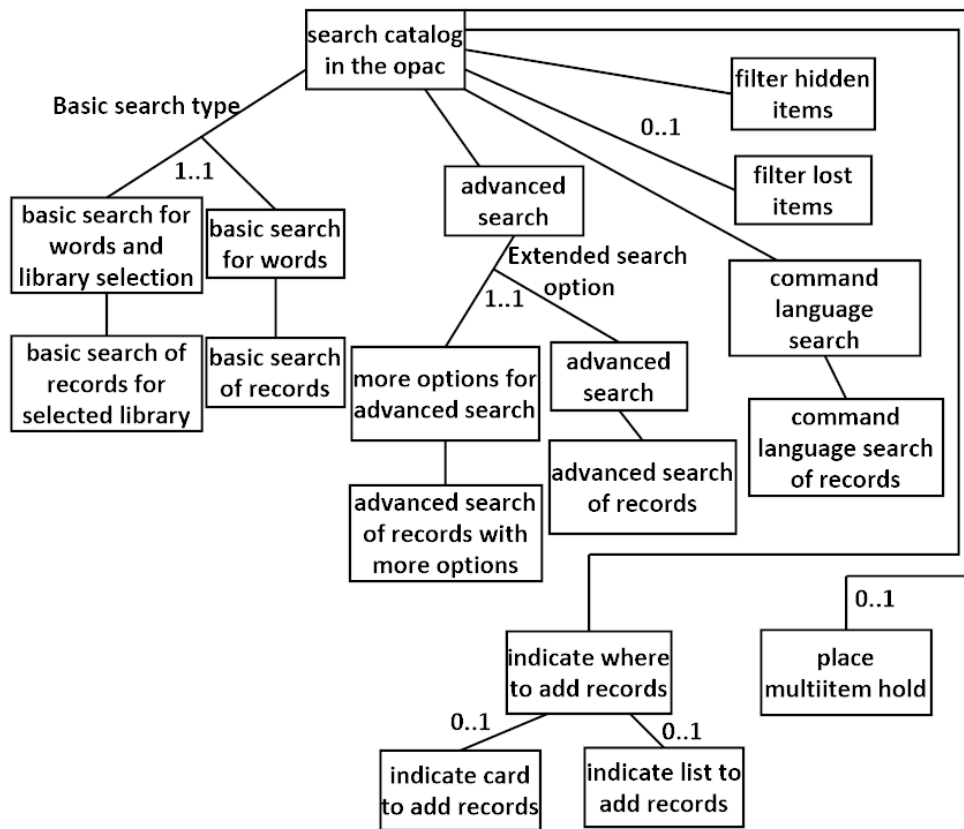


Figura 6.4: Primera parte del modelo de features para descripción del caso de uso *Search catalog in the OPAC*.

La figura 6.4 muestra la primera parte del modelo de features para el caso de uso *Search catalog in opac* (ver figura 5.26). Usando la regla 1b se mapea cada uno de los elementos de tipo «execute behavior» *Advanced search*, *Command language search* e *Indicate where to add records* a un elemento singleRelation: 1..1 entre la feature *search catalog in the opac* y la feature correspondiente al elemento de tipo «execute behavior» (con el mismo nombre del mismo). El resto de los elementos del modelo de features son generados utilizando la regla 3, en dos oportunidades (elementos GroupRelation *Basic search type* y *Extended search option*), la regla 1a y la regla 2a.

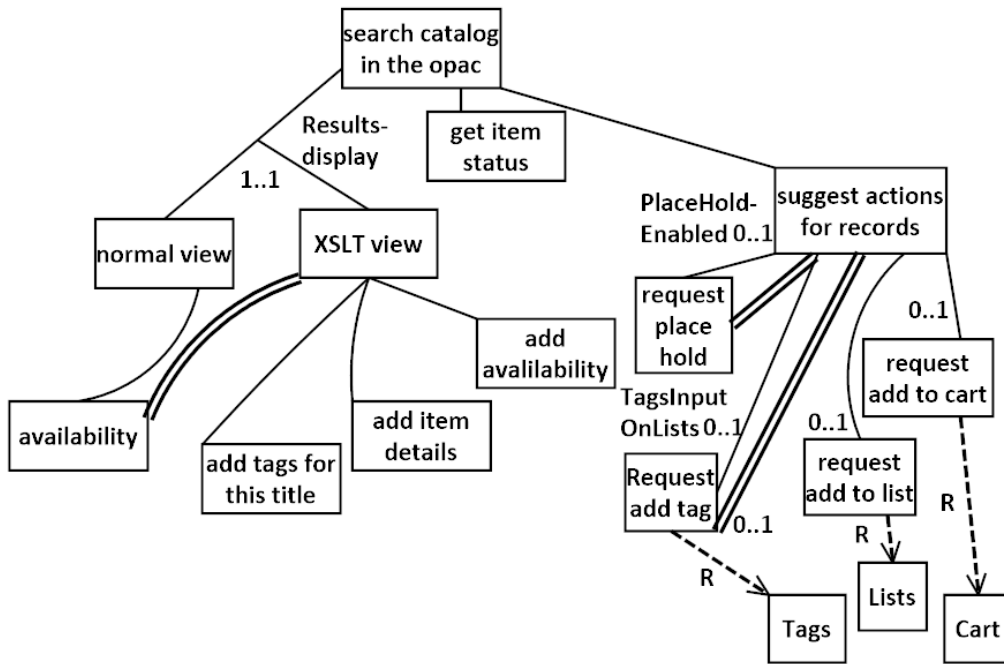


Figura 6.5: Segunda parte del modelo de features para descripción del caso de uso *Search catalog in the OPAC*.

La figura 6.5 muestra la segunda parte del modelo de features para el caso de uso *Search catalog in opac* (ver figura 5.26). Usando la regla 1b se mapea al elemento «execute behavior» *Suggest actions for records* a un elemento singleRelation: 1..1 entre la feature *search catalog in the opac* y la feature *suggest actions for records*. Usando la regla 2.a, se mapea la variabilidad de flujo de control *PHEEnabled*: 0..1 relacionada a la arista de actividad que apunta a la acción *request place hold* que está dentro del elemento «execute behavior» *suggest actions for records* a un elemento singleRelation: 0..1 entre la feature *suggest actions for records* y la feature *request place hold*. Usando la regla 7, se mapea la restricción de dependencia de tipo *requires* entre la arista de actividad que apunta a la acción *request add to cart* y el paquete de casos de uso *Cart* a un restricción de dependencia del mismo tipo entre las features *add to cart* y *Cart*.

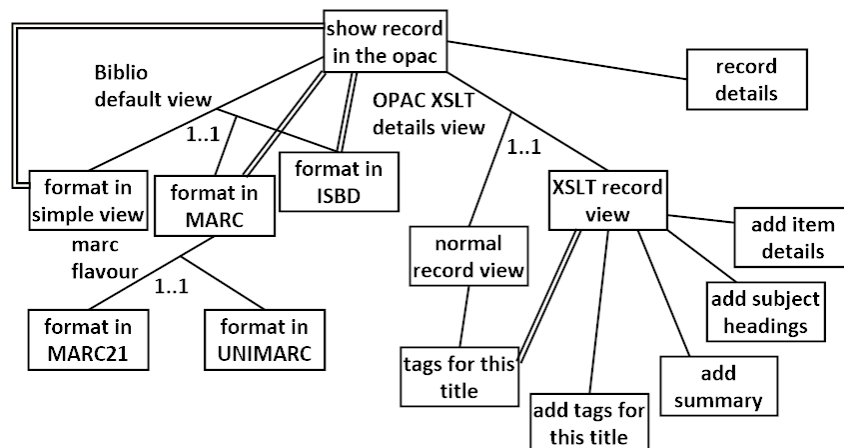


Figura 6.6: Primera parte del modelo de features para descripción del caso de uso *Show record in the OPAC*.

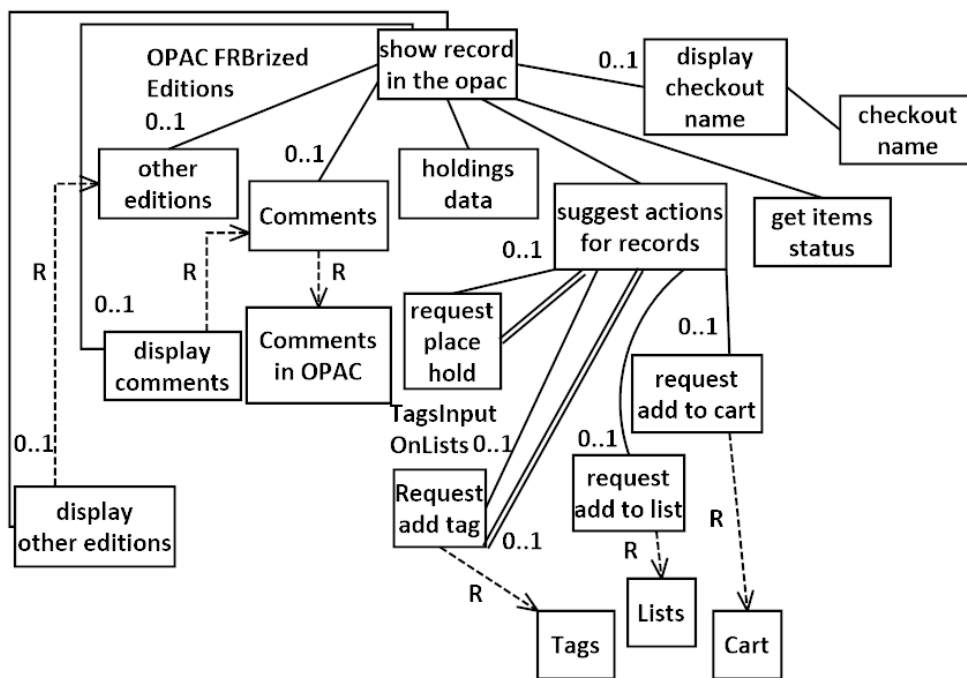


Figura 6.7: Segunda parte del modelo de features para descripción del caso de uso *Show record in the OPAC*.

6.3. Evaluación de reglas de transformación

Para analizar el resultado de la aplicación de las reglas de transformación de esta sección a nuestro caso de estudio se han considerado dos enfoques:

1. $\langle \text{unidad de análisis, pregunta, resultado basado en métrica} \rangle$, donde unidad de análisis puede ser UCD2FM o Reqs2FM.
2. Un enfoque para el análisis de features críticas (es decir, features variantes que deben ser elegidas por el cliente o usuario final) obtenidas a través de la aplicación de la transformación Reqs2FM (aplicada a casos de uso complejos) que no pueden o no deben ser obtenidas por medio de mapeo de variantes relación incluye de diagramas de casos de uso usando la transformación UCD2FM (estas features hacen notar la necesidad de mapear también diagramas de actividad a modelo de features):
 $\langle \text{criterio, pregunta, resultado basado en un métrica} \rangle$; donde criterio es la descripción de un subconjunto de features críticas.

En primer lugar se considera la unidad de análisis UCD2FM:

¿existe una elevada cantidad de paquetes de casos de uso anidados dentro de otros paquetes de casos de uso?

Se halló que el 93% de los paquetes de casos de uso satisfacen este criterio y para estos paquetes de casos de uso se debe usar una regla para anidar el paquete.

A continuación, se considera la unidad de análisis Reqs2FM.

¿es común que un caso de uso complejo incluya una restricción de dependencia entre un variante de diagramas de actividad y un variante de diagramas de casos de uso?

El 45% de los casos de uso complejos involucra restricciones de dependencia de este tipo. Se debe usar para esto una regla para restricciones de dependencia.

¿es común que un caso de uso complejo incluya un nodo para reutilización de comportamiento?

Se halló que el 72% de los casos de uso complejos necesitar la aplicación de la regla para nodos de reutilización de comportamiento (es decir call behavior action o nodo «execute behavior»).

¿es común que un caso de uso complejo incluya anidamientos de algún tipo?

Se encontró que el 54 % de casos de uso complejos incluyen algún anidamiento (2 o más, para ser precisos), y el 36 % de casos de uso complejos incluyen 3 o más ocurrencias de anidamientos.

Se encontraron un total de 83 features críticas en diagramas de actividad para describir casos de uso complejos que representan comportamientos.

Ahora, se evaluará la transformación Reqs2FM utilizando el segundo enfoque.

Features críticas en diagramas de actividad para casos de uso complejos que representan comportamientos que no pueden ser obtenidos mapeando variantes include: ¿es elevado el número de éstas features críticas?

Se encontró que un 9 % de las features críticas son de este tipo (es decir, IBDs con nodos final de actividad).

Features críticas obtenidas a partir de diagramas de actividad que describen casos de uso complejos que son demasiado simples para ser considerados como el mapeo de un variante include: ¿es elevado el número de éstas features críticas?

Se halló que el 68 % de las features críticas son de este tipo.

Capítulo 7

Desarrollo de modelos de dominio de diseño y generación automática de features a partir de ellos

7.1. Desarrollo de modelos de interfaz de usuario abstracta de dominio

Para desarrollar los modelos de interfaz de usuario abstracta correspondientes a los casos de de uso de dominio se utiliza la notación definida aquí, en la sección 3.5.3, de interfaz de usuario abstracta para familias de aplicaciones web.

Una vez que se han desarrollado los modelos de casos de uso de dominio y de diagramas de actividad de dominio, la siguiente y última etapa de modelado de dominio es la de interfaz de usuario abstracta.

Para desarrollar los modelos de dominio de interfaz de usuario abstracta se deben identificar, usando distintos criterios, qué variabilidades van a ser incluidas en la interfaz de usuario de cada caso de uso de la familia de aplicaciones.

7.1.1. Guías para desarrollo de modelos de interfaz de usuario de dominio

Identificación de variabilidades propias de interfaz de usuario

En base a las notaciones aquí definidas y dependiendo de los documentos de requisitos con los que se conste, se pueden identificar variabilidades en interfaz de usuario provenientes de los siguientes orígenes:

- *Variabilidades explicitadas en documentos de requisitos de interfaz de usuario provistos por el cliente.* En muchas ocasiones, es posible que el cliente brinde documentos (ya sea en forma textual, en forma de prototipo u otro tipo de formato) de requisitos que la interfaz de usuario debe cumplir. Para el caso de familia de aplicaciones, es posible que se incluyan situaciones donde fragmentos o pantallas de una interfaz de usuario deba ser modelada de distinta manera de acuerdo a distintas aplicaciones de destino o que algunos fragmentos o elementos de interfaz de usuario deban ser incluidos o no de acuerdo a la aplicación generada de destino. Para estas situaciones, el analista debe crear los modelos de interfaz de usuario abstracta adjuntando las variabilidades que sean necesarias a los elementos de interfaz de usuario abstracta.
- *Variabilidades identificadas por el analista de la familia de aplicaciones.* Al observar las acciones estereotipadas con «output», «output content» y «output message» (éste tipo de acciones hacen referencias a fragmentos o porciones de interfaz de usuario que el sistema presenta al usuario) de los diagramas de actividades con variabilidades utilizados para describir casos de uso, el analista puede creer conveniente presentar distintas alternativas de interfaz de usuario para distintas aplicaciones de destino, esto se debe modelar construyendo fragmentos o porciones de interfaz de usuario con variabilidades para las distintas alternativas presentadas. Para esta etapa puede ser útil

observar la categorización de situaciones de variabilidad en interfaz de usuario abstracta definida en el capítulo 3.5.3.

Identificar elementos de interfaz de usuario abstracta a partir de acciones de diagramas de actividad

En esta parte se brindan algunas guías útiles para definir la interfaz de usuario abstracta de un caso de uso (ya sea con variabilidades o no) a partir de la descripción detallada de un caso de uso a través de su diagrama de actividad (ya sea con variabilidades o no). Por cada estereotipo definido en el perfil de UML llamado *taxonomía para describir acciones de diagramas de actividades de aplicaciones web de la sección 2.1.1* se brindan distintas posibilidades de traducción a elementos de interfaz de usuario abstracta, facilitando así la tarea del analista. A continuación se listan los estereotipos de acciones de diagramas de actividad que deben ser tenidos en cuenta, y de qué manera, para la construcción de la interfaz de usuario abstracta:

- «input»: se refina a un elemento *Event* representando o bien la ocurrencia de un patrón de eventos (cuyos eventos atómicos son sobre elementos de interfaz de usuario que son refinados a partir de acciones con estereotipos comenzados con “output”) o bien la ocurrencia de un evento atómico sobre algún elemento de interfaz (refinado a partir de acciones con estereotipos comenzados con “output”). Se debe incluir la descripción del patrón de eventos o evento atómico en una cadena de texto respetando la gramática BNF presentada en la sección 2.2.2.
- «input and suggest»: se refina a un elemento *Event* incluyendo la descripción del evento en una cadena de texto respetando la gramática BNF presentada en la sección 2.2.2. En general el elemento de interfaz de usuario al que hará referencia la expresión del evento es un elemento *Atomic* con el atributo *editableType*= “input”.
- «output request»: esta acción se utiliza para solicitar entradas, por lo tanto hace referencia a elementos de entradas al sistema, por ello se debe a refinar o bien a un elemento *uiInputStructure* (por ejemplo, un formulario) o bien a un elemento *Atomic* con el atributo *editableType*= “input”.
- «output message»: ya que se utiliza para presentar un resultado al usuario se refina al elemento de interfaz de usuario *Message*, dependiendo del tipo de mensaje puede ser un elemento *Dialog* o un elemento *Notification*, con el atributo *type* especificado.
- «output media»: es usado para presentar archivos de medios y se refina a un elemento *MediaObject*.
- «output content»: se utiliza para presentar contenido, por lo tanto dependiendo del tipo de contenido a presentar se puede refinar a distintos tipos de elementos *ContentStructure*, a distintos tipos de elementos *NotOnlyLinksBased* o a un elemento *Atomic* con *editableType*= “no-editable” o *editableType*= “editable”. Algunos otros casos de refinamiento a otros tipos de elementos podrían surgir, pero suelen ser menos frecuentes.

Pasos para construir la interfaz de usuario abstracta de un caso de uso de dominio

A continuación, se brindan algunas guías útiles para construir la interfaz de usuario abstracta de un caso de uso para familia de aplicaciones. Por cada caso de uso que contenga interfaz de usuario abstracta (ya sea obligatorio, opcional o alternativo) se debe:

1. *Crear un elemento contenedor (de tipo GroupingElement) con el nombre del caso de uso.* Este contenedor será el que incluya todos los elementos de interfaz de usuario abstracta presentes en el caso de uso, con las variabilidades de interfaz de usuario adjuntadas a estos elementos, en caso de haberlas.
2. *Incluir los elementos de interfaz de usuario abstracta involucrados en variabilidades propias de interfaz de usuario.* En base a las variabilidades propias de interfaz de usuario identificadas se debe decidir qué elementos de interfaz de usuario abstracta involucrados en esas variabilidades son incluidos dentro del contenedor asociado al caso de uso y asociar las anotaciones de variabilidades a los elementos según corresponda.

3. *Definir e incluir elementos de interfaz de usuario abstracta de partes obligatorias de descripción del caso de uso.* Para esto se debe observar, en la descripción del caso de uso a través de diagrama de actividades, a las acciones estereotipadas con «output content», «output request» y «output message» que no están involucradas directa o indirectamente (a través de un elemento que los contenga) en ninguna variabilidad, y decidir con qué elementos de interfaz de usuario abstracta se van a modelar esas acciones. Si hay alguna variabilidad propia de interfaz de usuario que involucre a estos elementos, en ese caso la variabilidad debe ser definida e incluida. Estos elementos son incluidos en el contenedor siguiendo la guía 1.
4. *Definir e incluir elementos de interfaz de usuario abstracta de partes alternativas de descripción del caso de uso.* Para esto se debe observar, en la descripción del caso de uso a través de diagrama de actividades, a los elementos variantes que incluyen acciones estereotipadas con «output content», «output request» y «output message». Para estos elementos, se debe decidir con qué elementos de interfaz de usuario abstracta se los va a representar. Además, al elemento de interfaz de usuario abstracta de más alto nivel que represente la parte de interfaz de usuario variante se lo debe relacionar con la feature que corresponda en el modelo de features y se debe marcar con una letra “**F**” en alguna parte de este elemento. Esto agilizará y facilitará, al resolver las variabilidades del modelo de features, la tarea de crear la interfaz de usuario abstracta de una aplicación, descartando las partes de interfaz de usuario marcadas con “**F**” para las cuales fueron descartadas sus features relacionadas en el modelo de features. Los elementos de este tipo son incluidos en el contenedor definido siguiendo la guía 1.

7.1.2. Aplicación al caso de estudio de Sistema de Administración de Bibliotecas Online

En la figura 7.1 se muestra la interfaz de usuario abstracta con variabilidades del caso de uso *Search catalog in the OPAC*. De la etapa de identificación de variabilidades propias de interfaz de usuario se obtuvieron dos variabilidades. La variabilidad *noItemTypeImages* con cardinalidad 0..1, que está asociado al elemento de tipo icon *item type* y que significa que este elemento será opcional y podrá ser incluido o no de acuerdo a la aplicación; y la variabilidad *AuthorisedValueImages* también con cardinalidad 0..1 asociada al elemento de tipo image *authorised value* que significa que este elemento será opcional y podrá ser incluido o no de acuerdo a la aplicación.

Luego, aplicando las guías para la construcción de interfaz de usuario abstracta de un caso de uso para familia de aplicaciones, se generó la interfaz de usuario abstracta del caso de uso. En primer lugar se incluyó un elemento de tipo GroupingElement con el nombre del caso de uso, el cual incluye otros tres elementos del mismo tipo para la parte de realizar las búsquedas de ítems (contenedor *Search*), la parte de brindar información sobre la cantidad de resultados de búsquedas (contenedor *search results information*) y la parte de los resultados de búsquedas (contenedor *results list*). El elemento de tipo GroupingElement *Search* incluye dos elementos del mismo tipo para búsqueda básica marcados con “**F**”, lo cual significa que serán incluidos o no en la interfaz abstracta del caso de uso de una aplicación de acuerdo a la resolución de la variabilidad asociada a los nombres de los contenedores en la etapa de descripciones de casos de uso con variabilidades. Además se incluye un enlace que al ser presionado mostrará una porción de interfaz de usuario para búsqueda avanzada que por razones de legibilidad es presentada en otra figura a continuación. El elemento de tipo GroupingElement *search results information* incluye un elemento de tipo Text *results information* de sólo lectura con la información de la cantidad de ítems encontrados en la búsqueda. El elemento de tipo GroupingElement *results list* incluye dos elementos principales, un GroupingElement llamado *common bar* que representa una barra con posibles acciones a realizar sobre selecciones de la lista de ítems encontrados en la búsqueda; los elementos que representan esas posibles acciones se encuentran marcados con “**F**” ya que se corresponden con variabilidades de la etapa de descripciones de casos de uso; y un elemento del tipo Record de Record, que representa la lista de ítems encontrados. El record *item* incluye algunos elementos que representan posibles acciones sobre el ítem, algunos de los cuales se encuentran marcados con “**F**” ya que se corresponden con variabilidades de la etapa de descripciones de casos de uso.

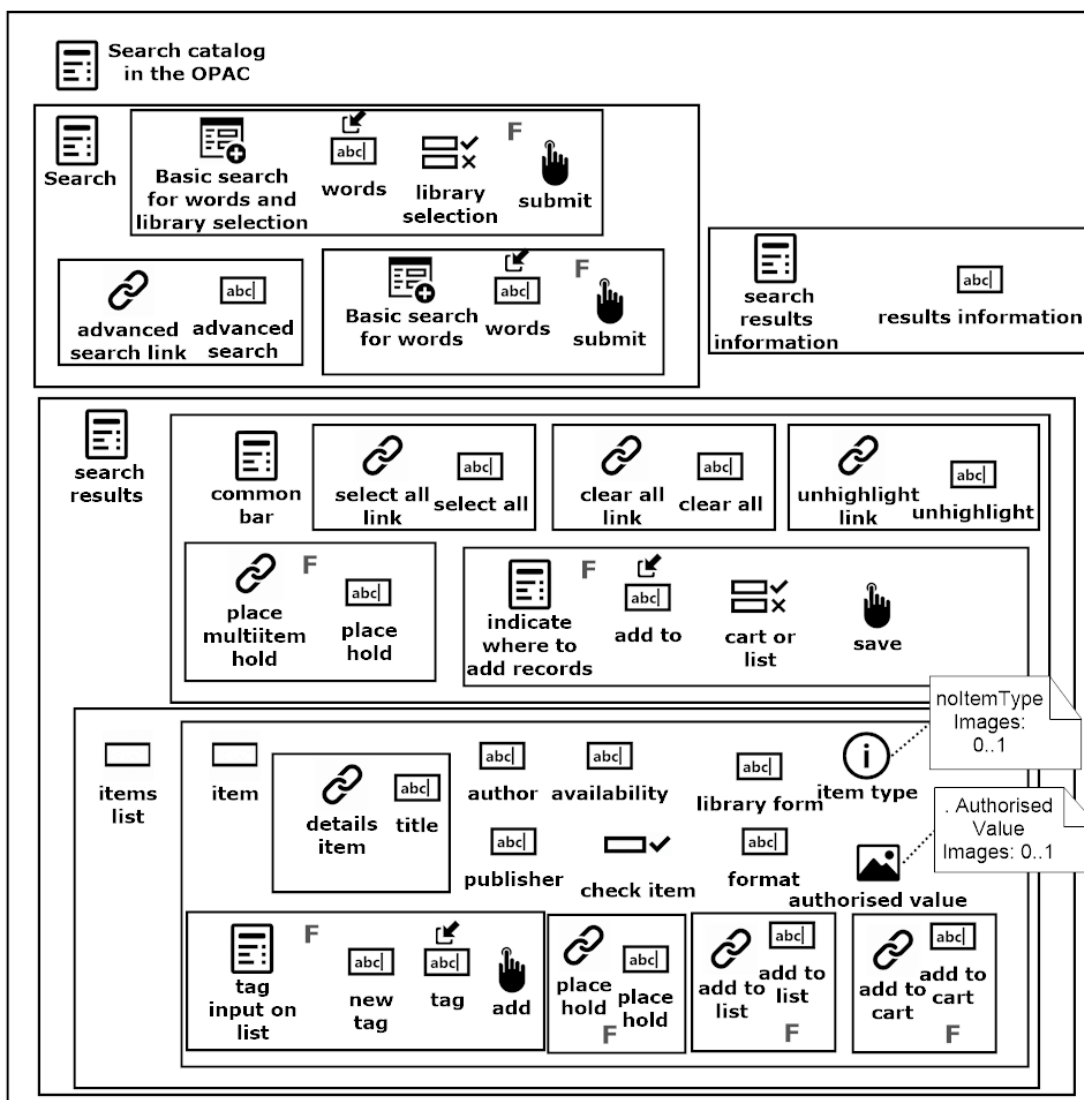


Figura 7.1: Interfaz de usuario abstracta con variabilidades del caso de uso *Search catalog in the OPAC*.

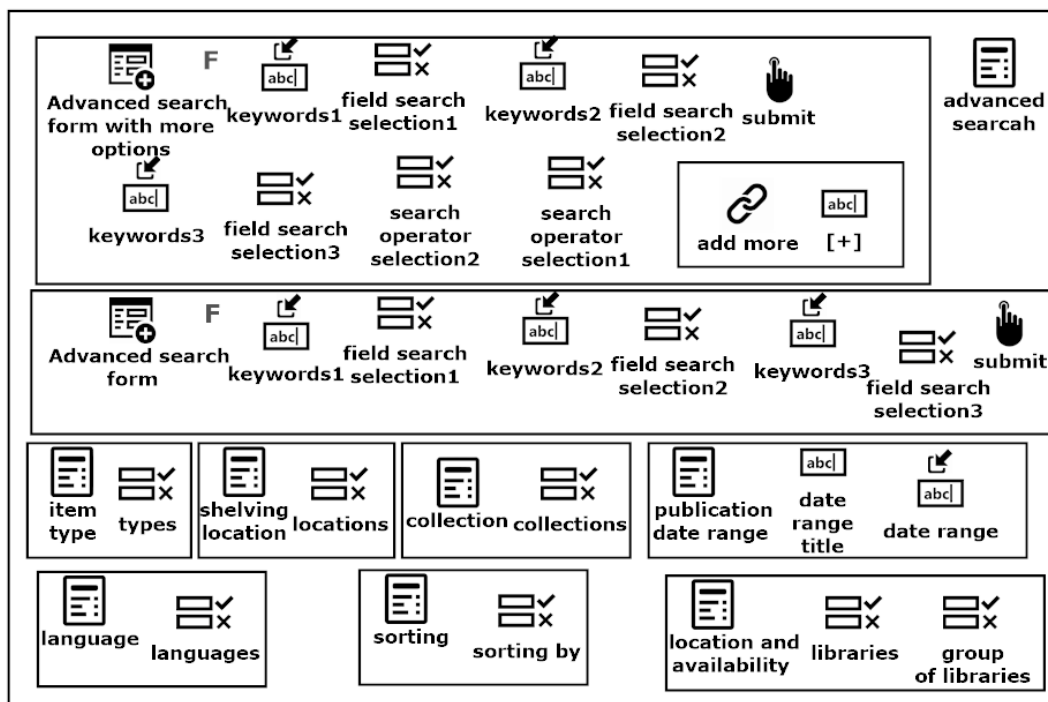


Figura 7.2: Interfaz de usuario abstracta con variabilidades para la funcionalidad *Advanced search*.

En la figura 7.2 se muestra la interfaz de usuario abstracta con variabilidades para una parte del caso de uso *Search catalog in the OPAC* que representa a la funcionalidad *advanced search*. Esta parte incluye un elemento de tipo `GroupingElement` *advanced search* que será visible cuando un usuario presione el enlace incluido en el caso de uso *Search catalog in the OPAC*. Este contenedor incluye dos elementos de tipo `Form` marcados con “F” que representan dos tipos de búsquedas avanzadas, de las cuales solo una será seleccionada para una aplicación en la etapa de descripciones de casos de uso. El primer elemento de tipo `Form` se llama *Advanced search form with more options* y representa un búsqueda avanzada con tres elementos `Text` de tipo `Input`, dos elementos de tipo `Multiple Choice` para selección de operadores de búsqueda y tres elementos de tipo `Multiple Choice` para selección el campo del ítem para el cual se va a aplicar la búsqueda. A esos tres elementos `Text` de tipo `Input` que representan términos de búsqueda, se les pueden agregar más términos de búsqueda como los que se crea necesarios a través de presionar un enlace para agregar términos de búsqueda (representado con elemento de tipo `Anchor` *add more*). El otro elemento de tipo `Form` se denomina *Advanced search form* y representa la búsqueda avanzada con solo tres términos de búsqueda (no se pueden agregar más opciones o términos de búsqueda), los cuales son representados con tres elementos `Text` de tipo `Input` y tres elementos de tipo `Multiple Choice` para selección del campo del ítem al cual se va a aplicar la búsqueda. En ambos elementos de tipo `Form` se incluye un elemento de tipo `Button` representando el envío de los datos del formulario.

Además en el contenedor principal de esta funcionalidad, se incluyen varios contenedores que representan filtros de búsqueda a través de selección de elementos de tipo `Multiple Choice` y un elemento de tipo `GroupingElement` que representa filtro de búsqueda a través de inclusión de rango de fechas.

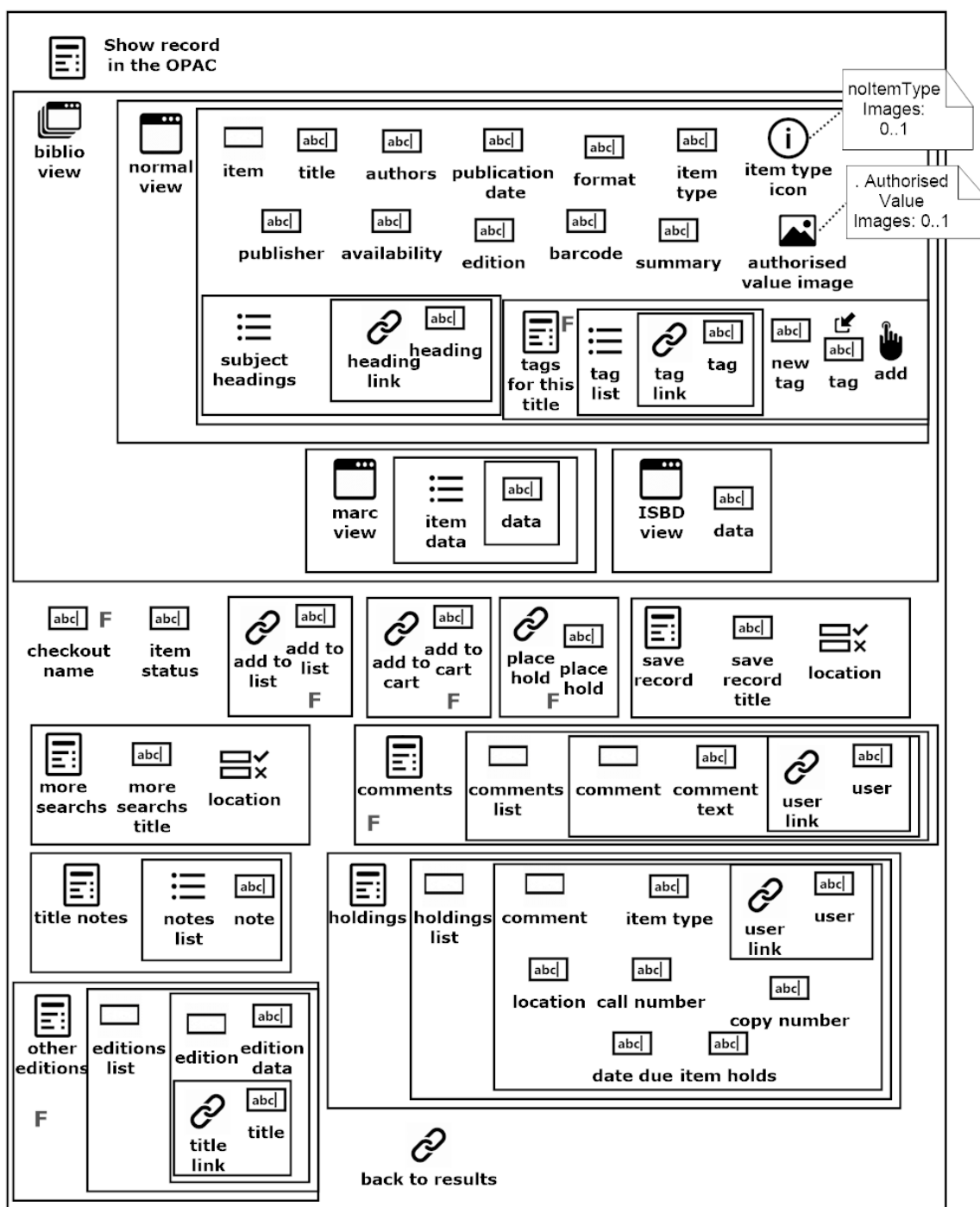


Figura 7.3: Variabilidades identificadas en la interfaz de usuario abstracta del caso de uso *Show record in the OPAC*.

En la figura 7.3 se muestra la interfaz de usuario abstracta con variabilidades del caso de uso *Show record in the OPAC*. El mismo consta de un contenedor principal con el nombre del caso de uso que, a su vez, incluye otros elementos para representar la interfaz de usuario abstracta con variabilidades del caso de uso. Solo dos elementos fueron identificados como variabilidades de interfaz de usuario, estos son *noItemTypeImages* y *AuthorisedValueImages* que tienen el mismo significado que el de los elementos equivalentes del caso de uso *Search catalog in the OPAC*. Dentro del contenedor principal se incluye un elemento del tipo *NavAltBlock* llamado *biblio view*, el cual incluye tres elementos de tipo *Block* que representan las tres posibles formas de visualizar datos de un ítem de la biblioteca. El primer elemento de

tipo Block se denomina *normal view* e incluye un elemento de tipo Record *item* con los datos del ítem y posibles acciones sobre etiquetas. El siguiente elemento de tipo Block se llama *marc view* y representa la visualización del ítem en formato marc a través de una lista de datos de Text de solo lectura. El tercer elemento de tipo Block se llama *ISBD view* e incluye un elemento de tipo Text de solo lectura con los datos del ítem.

Se incluyen dos elementos que representan informaciones del ítem incluidos en las tres vistas: *checkout name* (marcado con “F”) e *item status*. Se incluyen cinco elementos que representan acciones posibles a realizar sobre el ítem: *add to list* (marcado con “F”), *add to card* (marcado con “F”), *place hold* (marcado con “F”), *save record* y *more searchs*. Finalmente se incluyen cuatro elementos con informaciones asociadas al ítem a través de listas de ítems: *comments* (marcado con “F”), *other editions* (marcado con “F”), *title notes* y *holdings*.

7.2. Mapeando modelos de interfaz de usuario abstracta de dominio a features

7.2.1. Background

Sabiendo que “los modelos de features han sido muy populares en las líneas de productos de software y son ampliamente aceptados y utilizados por las comunidades académicas e industriales, en detrimento de otras notaciones” (ver [Schobbens, 2007]). Y que, además, los modelos de features son útiles para validación con el cliente y facilitan la generación de configuraciones válidas; se hace necesario que las variabilidades que surgen en los distintos niveles de abstracción del desarrollo de aplicaciones web, estén contempladas en el modelo de features de la familia de aplicaciones web.

Además, como ya hemos mencionado en la sección de estado de arte en el capítulo 1.3 de Introducción, durante la ingeniería de dominio tanto los modelos de dominio de requisitos como los modelos de features son necesarios, ya que los modelos de features son orientados al especialista en reutilización (son útiles para organizar y validar conceptos de variabilidad este rol) y los modelos de requisitos son orientados al usuario (el usuario interactúa en estos tipo de modelos). Este argumento puede ser generalizado para modelos de dominio de interfaz de usuario, ya que su especificación se basa en requisitos de interfaz de usuario.

Algunos autores en la bibliografía (ver [Pleuss, 2010], [Gabillon, 2013], [Gabillon, 2015], [Sottet, 2015] y [Fadhilillah, 2018]) se han ocupado del asunto de modelar variabilidades con modelos de features por medio de identificación de variabilidades en interfaz de usuario de aplicaciones web, representación de esas variabilidades en modelos de features y posterior modelado de features en notaciones específicas del dominio de interfaces de usuario web. En [Casalánguida, 2012] se menciona un argumento para modelos de UML pero que puede ser generalizado para notaciones específicas de un dominio (como notaciones para describir interfaces de usuario web), que dice “el problema de construir primero modelo de features y a partir de estos construir modelos de UML para requisitos es bastante complejo, ya que no es esperable que los especialistas en UML construyan modelos de features de buena calidad (por ejemplo, a partir de documentos de requisitos en lenguaje natural) porque es una tarea compleja y los modelos de features no forman parte de UML”. Además, algunos autores recomiendan, de ser posible, minimizar la participación humana en la confección de modelos de features. Sobre esta línea de pensamiento, como hemos citado anteriormente, en [Wang, 2009], se dice que “la calidad de construir un modelo features depende fuertemente de los conocimientos y experiencia de la persona que ocupa el rol de analista en el dominio, por lo tanto, será valioso si existe un enfoque que pueda minimizar la participación de los analistas en la construcción del FM al automatizar actividades clave en la construcción de un FM”.

Como una solución a las dificultades planteadas, se propone generar de manera automática features y variabilidades a partir de variabilidades representadas en modelo de dominio de interfaz de usuario abstracta. Esta solución contribuye a alcanzar la meta número 4 de la sección 1.4 del capítulo de Introducción.

En este trabajo se ha realizado una transformación de modelos en ATL que toma como entrada un modelo de interfaz de usuario abstracta con variabilidades y arroja como resultado un conjunto de features y variabilidades que son incorporadas al modelo de features generado para requisitos funcionales de la familia de aplicaciones web.

7.2.2. Trabajo relacionado

No hemos hallado en la bibliografía algún enfoque que permita generar features y variabilidades para ser adjuntadas a un modelo de features ya existente a partir de modelo de dominio de interfaz de usuario para aplicaciones web.

En general, los enfoques hallados (ver [Pleuss, 2010], [Gabillon, 2013], [Gabillon, 2015], [Sottet, 2015] y [Fadhilillah, 2018]) proponen incorporar manualmente features y variabilidades provenientes de interfaz de usuario a un modelo de features ya existente. Luego proponen alguna notación de interfaz de usuario (ya sea abstracta o concreta) para modelar las features provenientes de interfaz de usuario. Este conjunto de features modeladas con notación de interfaz de usuario conforman una infraestructura de bienes reutilizables. En este trabajo hemos definido una notación de interfaz de usuario abstracta con variabilidades (de dominio) y, aplicando transformación de modelos, se generan de manera automática las features y variabilidades a partir de modelos de la notación de interfaz de usuario abstracta con variabilidades. Luego, estas features son adjuntadas automáticamente a un modelo de features ya existente.

7.2.3. Reglas de transformación

Para construir el modelo final de features de la familia de aplicaciones es necesario incorporar las features y relaciones provenientes de las variabilidades identificadas y especificadas en los modelos de interfaz de usuario abstracta. Para esta tarea se deben tomar como entradas el modelo de features construido para los requisitos de la familia de aplicaciones y las partes de la interfaz de usuario con variabilidades construidas para cada caso de uso identificado en la etapa de requisitos.

Se realizó una transformación de modelos llamada *UI2FM* (en lenguaje *ATL*), la cual incluye cinco reglas de transformación y es detallada en pseudocódigo a continuación:

Regla	Mapea	En
1	P nombre de parte de UI de un caso de uso	P: Feature, P.name = "UI" SingleRelation:1..1 entre P:Feature y U:Feature, donde U es la feature correspondiente al caso de uso de P
2	V: VariabilityUI: 0..1 Variante: U: UIElement.	SingleRelation:0..1 entre P:Feature y U:Feature
3	V: VariabilityUI: n..m Variantes: U ₁ ,...,U _k : UIElement.	GroupRelation:n..m entre P:Feature y U ₁ ,...,U _k :Feature
4	U: UIElement, U es variante	UV: Feature, UV.name = U.name
5	R restricción de dependencia de tipo T entre variantes N y N'	DC de tipo T entre feature para N y feature para N'

Figura 7.4: Transformación *UI2FM* para adjuntar features y relaciones provenientes de interfaz de usuario abstracta con variabilidades a modelo de features de la familia de aplicaciones.

En la tabla de la figura 7.4 se encuentran las cinco reglas necesarias para transformar las variabilidades de los modelos de interfaz de usuario abstracta a features y relaciones entre features. Por cada caso de uso de requisitos que incluye variabilidades en la construcción de su modelo abstracto de interfaz de usuario se crea una feature con el nombre "UI", de la cual heredarán todas las features correspondientes a la interfaz de usuario del caso de uso. Esto está reflejado en la regla 1 de la tabla. Las reglas 2 y 3 mapean las relaciones entre features correspondientes a elementos variantes de interfaz de usuario y la feature con el nombre "UI", construida en la regla. Dependiendo de si el elemento variante es uno o muchos, se define el tipo de relación, que puede ser una relación simple (regla 1) o una relación de grupo (regla 2). La regla 4 mapea cada elemento variante en el modelo de interfaz de usuario a una feature. La regla 5 mapea restricciones de dependencias de tipo Requires o Excludes entre variantes de tipo UI Element a restricciones de dependencia del mismo tipo entre features.

Ejemplos. La figura 7.5 muestra el modelo de features generado para la interfaz de usuario abstracta con variabilidades para el caso de uso, perteneciente al caso de estudio de Sistema de Bibliotecas Online, *Search catalog in the OPAC*. La interfaz de usuario abstracta de este caso de uso incluye dos variabilidades

opcionales, es decir con cardinalidad 0..1, que hacen referencia a elementos de interfaz de usuario opcionales, los cuales son *item type icon* (si se incluye o no un ícono para describir el tipo de record) y *authorized value image* (si se incluye o no una imagen para describir los valores autorizados). La variabilidad adjuntada a estos dos elementos son mapeadas a sendos elementos del tipo SingleRelation con cardinalidad 0..1, utilizando la regla 1 de la tabla 7.4. Las features correspondientes a los elementos de interfaz de usuario abstracta opcionales (asociadas a la parte inferior del elemento SingleRelation) son generadas a través de la aplicación de la regla 3. Además, es generada una feature obligatoria (con cardinalidad 1..1) con la palabra “UI” que es descendiente de la feature raíz del modelo de features correspondiente al caso de uso, aplicando la regla 0. Esta es la feature raíz de la parte de features correspondiente a interfaz de usuario abstracta con variabilidades y de las cuales descenderán todas las relaciones entre features de interfaz de usuario abstracta generadas.

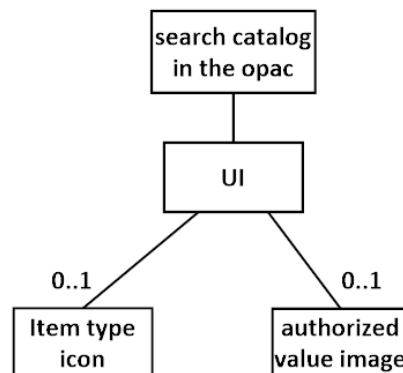


Figura 7.5: Modelo de features generado utilizando la transformación emphUI2FM para la interfaz de usuario abstracta del caso de uso *Search catalog in the OPAC*.

La figura 7.6 muestra el modelo de features generado para la interfaz de usuario abstracta con variabilidades para el caso de uso, perteneciente al caso de estudio de Sistema de Bibliotecas Online, *Show record in the OPAC*. Como en todos los casos de uso cuya interfaz de usuario abstracta incluye al menos una variabilidad, una feature obligatoria (con cardinalidad 1..1) es generada con la palabra “UI”. Al igual que para el caso de uso *Search catalog in the OPAC*, este caso de uso incluye los dos elementos de interfaz de usuario abstracta opcionales *item type icon* y *authorized value image*. Y, además, de manera análoga a estos dos elementos, otro elemento opcional llamado *checkout name* (si se incluye o no el nombre del checkout en la interfaz de usuario del proceso del caso de uso) es mapeado a una feature con el mismo nombre (regla 3), y la variabilidad adjuntada a ese elemento es mapeada a un elemento SingleRelation con cardinalidad 0..1 (regla 1).

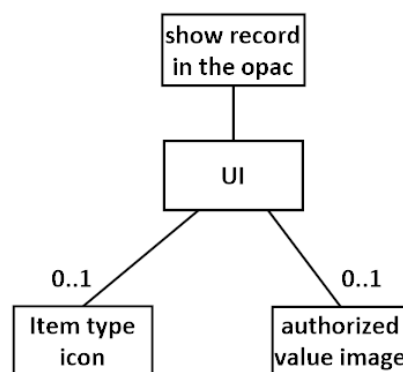


Figura 7.6: Modelo de features generado utilizando la transformación emphUI2FM para la interfaz de usuario abstracta del caso de uso *Show record in the OPAC*.

Parte III

Configuración de modelos de dominio de aplicaciones web

Capítulo 8

Proceso de configuración de modelos de dominio

8.1. Background

Cuando los modelos de dominio de la familia de aplicaciones han sido producidos, la próxima etapa en el desarrollo de líneas de productos de software es la producción de aplicaciones. Para ello, la primer y fundamental tarea es la de derivar modelos de aplicación desde modelos de dominio de la familia de aplicaciones.

Ya que “*los modelos de features son usados para configurar otros modelos, y eventualmente código, por medio de selección de features variantes*” (ver [Vranić, 2016]), y que en este trabajo obtenemos modelos de features automáticamente a partir de modelos de dominio de la familia de aplicaciones, se optó por obtener modelos de aplicación a través de un *proceso de derivación de una configuración concreta respetando un modelo de features por medio de selección y clonación de features y especificando valores de atributos*. A este proceso se lo denomina configuración (ver [Czarnecki, 2005]).

El proceso de llevar a cabo una configuración involucra dos actividades principales: obtener una configuración válida (que respete ciertas restricciones de integridad) de features que coincida con un conjunto específico de requisitos de una aplicación y, obtener modelo de features de la aplicación en base a una configuración encontrada y el modelo de features de la familia de aplicaciones. Algunos autores se refieren como configuración directamente al modelo de features configurado en base a la selección de features. En este trabajo preferimos dividir esta etapa en las dos actividades anteriormente citadas. El proceso de configuración implica el razonamiento sobre un conjunto complejo de restricciones para alcanzar un objetivo final (ver [White, 2009]) y es una tarea dura y propensa a errores (ver [Jacob, 2015]), por estos motivos en los últimos años han aparecido trabajos que buscan automatizar lo máximo posible este proceso.

Hemos distinguido en la bibliografía dos tipos de enfoques para configuración automatizada basada en modelos de features, enfoques que utilizan formalizaciones matemáticas y configuración interactiva (el usuario selecciona y elimina features, el sistema responde chequeando validez de la configuración) y enfoques basados en Model Driven Development y obtención de modelos a partir de meta-modelos (por ejemplo [Asikainen, 2006], [Bragança, 2007]).

La mayoría de los enfoques que utilizan formalizaciones matemáticas se basan en traducción del modelo de features a fórmulas de lógica proposicional o a CSP - Constrain Satisfaction Problems - (los trabajos de [Batori, 2005], [Benavides, 2005] y otros trabajos que se basan en estos como [Jacob, 2015], [White, 2009]); otros trabajos proponen, además, el uso de Binary Decision Diagrams (BDD) para representar el posible espacio de configuración de las features ((por ejemplo [Mendonca, 2008] y [Czarnecki, 2007]); otro enfoque alternativo (ver [Laguna, 2011]) propone la formalización de modelos de features a través de hipergrafos dirigidos y realiza configuración en base a ellos.

En general, los trabajos que utilizan formalizaciones matemáticas requieren traducción de modelo de features a una estructura matemática intermedia para realizar la configuración de un modelo de features. Si se desea trabajar con MDD, como en este trabajo, esto representa una tarea extra; ya que para obtener modelos de features que respeten una configuración dada hay que realizar dos transformaciones de modelo en lugar de una. Esto debido a que habría que: meta-modelar features, meta-modelar la estructura

matemática utilizada, definir una transformación de features a estructura matemática representando modelo de features, definir la configuración, configurar la estructura matemática que representa el modelo de features y, finalmente, traducir la estructura matemática configurada a modelo de features. Salvo los trabajos de [Laguna, 2011] y [Jacob, 2015]), los trabajos que utilizan formalizaciones matemáticas incluyen modelos de features basados en relaciones de descomposición (de tipo OR, AND, mandatory, optional, entre otras), distintos a los utilizados en este trabajo. La motivación de trabajar con formalizaciones matemáticas, en el caso del trabajo de [Jacob, 2015], es la de obtener configuraciones óptimas basadas en preferencias de usuarios; en este trabajo no nos concentramos en eso. El trabajo de [Laguna, 2011] justifica la traducción de features a hipergrafos dirigidos diciendo que los modelos de features pueden ser difíciles de modificar, sobre todo si un cambio afecta a una restricción de dependencia. En este trabajo los modelos de features son generados automáticamente en base a modificaciones hechas en los modelos de dominio de UML, con lo cual no encontramos necesario traducir modelos de features a hipergrafos dirigidos.

En general, los enfoques que utilizan MDD no permiten configuración interactiva (ya que se basan en producir modelos respetando un meta-modelo) y requieren transformar el modelo de features a un modelo intermedio de clases. Además, en el trabajo de [Asikainen, 2006] no se explicita cómo modelar restricciones de dependencias en modelos de features (sugiere el uso de restricciones generales en un lenguaje como OCL), se dice que cualquier tipo de restricción en modelos de features se tiene que cumplir para producir configuraciones válidas, pero no se explicita como se automatiza esto. En [Bragança, 2007], las restricciones de dependencia son modeladas con restricciones OCL sobre elementos del modelo de features y no se explicita como estas restricciones se satisfacen de manera automática en la producción de una configuración válida.

Otro asunto de interés en este trabajo, es el de configurar los modelos de dominio producidos para la familia de aplicaciones. De los métodos citados anteriormente para configuración automatizada basada en modelo de features, solo [Bragança, 2007] tiene en cuenta configuración para casos de uso de UML de dominio.

El objetivo de esta parte del trabajo es la definición de un enfoque que soporte desarrollo basado en MDD (ya que en este trabajo utilizamos un proceso basado en MDD) para configuración automatizada basada en modelos de features que permita configuración interactiva, tenga satisfacción de restricciones de dependencias de modelos de features, que no requiera traducción de modelos de features a modelos o formalizaciones intermedias para realizar la configuración y que en base a configuraciones producidas a partir de modelo de features, permita configurar modelos de dominio de UML de manera automática. Este objetivo contribuye a alcanzar la meta número 3 de la sección 1.4 del capítulo de Introducción y permite alcanzar la meta número 5 de la sección 1.4 del capítulo de Introducción.

En esta parte del trabajo se realizaron las siguientes tareas (para la definición de un proceso para automatizar configuración de modelos de features y modelos de dominio): definición de una conceptualización para modelo de features, definición de configuración y configuración válida en base a conceptualizaciones ya realizadas en la bibliografía; definición de reglas para configuración de modelos de dominio de UML; implementación de una herramienta web que basándose en las conceptualizaciones y reglas brindadas que permite automatizar por completo la producción de configuraciones válidas de manera interactiva, y configura de manera automática modelo de features y modelos de dominio, permitiendo transformar modelos de dominio a modelos de aplicación.

8.2. Trabajo relacionado

Se tuvieron en cuenta como trabajos relacionados a los enfoques que cumplen con las siguientes condiciones:

- Automatizan de manera completa la producción de configuraciones válidas.
- Automatizan de manera completa la configuración de modelos de features.
- Automatizan de manera completa la configuración de modelos de dominio, al menos para diagramas de casos de uso con variabilidades o diagramas de actividad de UML con variabilidades.
- Incluyen herramienta o plugin de herramienta para automatizar todas las tareas anteriores.

	Permite configuración interactiva	Requiere traducción a modelos intermedios	Implementa restricciones de dependencia en configuración	Modelos configurados
<i>[Bragança, 2007]</i>	Parcial	Si	No	Clases y Casos de Uso
<i>Este trabajo</i>	Total	No	Si	Features, Casos de Uso, Diagramas de actividad y Clases

Figura 8.1: Comparación entre enfoques seleccionados para configuración automatizada basada en modelos de features..

En la tabla comparativa de la figura 8.1 se incluye el trabajo de [Bragança, 2007] y el trabajo realizado aquí. El primer ítem de la tabla *Permite configuración interactiva*, hace referencia a si el usuario puede ir seleccionando y eliminando features de una manera intuitiva y el sistema respondiendo si no se satisface alguna restricción para producir una configuración válida, el trabajo de [Bragança, 2007] no especifica cómo el sistema respondería si no se cumple alguna restricción de dependencia, por lo tanto consideramos que la configuración no se produce de manera completamente interactiva. Para el segundo ítem de la tabla *Requiere traducción a modelos intermedios*, en este trabajo no se quiere traducir el modelo de features a un modelo intermedio, en el trabajo de [Bragança, 2007] el modelo de features es traducido a un modelo de clases por medio de una transformación de modelos. En cuanto al tercer ítem de la tabla, *Implementa restricciones de dependencia en configuración*, en este trabajo las restricciones de dependencia son implementadas en la herramienta y en el trabajo de [Bragança, 2007] no se brinda ningún tipo de detalle al respecto. El último ítem de la tabla *Modelos configurados* hace referencia a qué modelos son configurados en el proceso de configuración, en el trabajo de [Bragança, 2007] se configuran modelos de clases (traducción de modelos de features) y diagramas de casos de uso de UML (aunque no está incluido el detalle de la configuración), en este trabajo se configuran modelos de features, modelos de casos de uso de UML y se brindan guías para configurar diagramas de actividades y modelos de clases.

Los trabajos que utilizan formalizaciones matemáticas y el trabajo de [Asikainen, 2006] no fueron incluidos como trabajos relacionados ya que no cumplen con el ítem de automatizar de manera completa la configuración de modelos de dominio, al menos para diagramas de casos de uso con variabilidades o diagramas de actividad de UML con variabilidades. Esto es importante porque configurar un modelo de dominio de UML en base a una configuración es una tarea no trivial para los tipos de modelos de dominio definidos en este trabajo.

8.3. Descripción del proceso

En esta sección se brindan algunas nociones introductorias del proceso de configuración automatizada basada en modelos de features en base a los modelos de dominio producidos en la etapa anterior del desarrollo de familia de aplicaciones. Además se incluye un breve resumen de cada una de las etapas que constituyen el proceso de configuración de modelos de dominio.

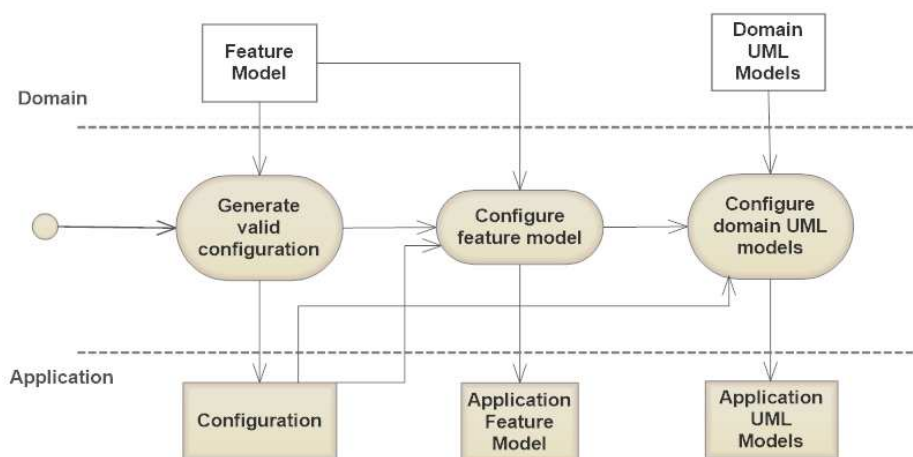


Figura 8.2: Proceso de desarrollo de modelos de una aplicación a partir de modelos de dominio de aplicaciones web..

En la figura 8.2 se brinda una descripción gráfica del proceso de configuración automatizada basada en modelos de features. La primera actividad consiste en la generación de una configuración válida del modelo de features de la familia de aplicaciones, para esta tarea el usuario debe seleccionar y eliminar de manera interactiva las features que serán incluidas en la configuración a producir. En el próximo capítulo de este trabajo se brindan conceptualizaciones de modelo de features, configuración y configuración válidas; tales definiciones fueron utilizadas para implementar la herramienta del apéndice B, la cual es utilizada para llevar a cabo dicha actividad de generar una configuración válida. Cuando se cuenta con una configuración válida, la siguiente actividad (realizada de manera automática en la herramienta del apéndice B) es la de realizar la configuración del modelo de features, para así obtener el modelo de features de la aplicación; el cual puede ser útil para validación de funcionalidades con el cliente. La actividad final, configurar modelos de dominio de UML (realizada en la herramienta del apéndice B para diagramas de casos de uso), toma como entrada la configuración producida y el modelo de dominio de UML a configurar y arroja como resultado los modelos de UML de la aplicación, que serán usados para producir la aplicación deseada. Para las dos últimas actividades, en un capítulo posterior de este capítulo, se brindan guías para producir las configuraciones, las cuales fueron utilizadas en la implementación de la herramienta del apéndice B.

Capítulo 9

Configuración de diagramas de features

9.1. Definición de configuración

En [Czarnecki, 2004] se dice que una configuración de un modelo features “*consiste de las features que fueron seleccionadas de acuerdo las restricciones de variabilidad definidas en el diagrama de features. La relación entre un diagrama de features y una configuración es comparable a la relación entre una clase y una instancia en programación orientada a objetos*”. En base a este concepto y a distintas definiciones realizadas en el trabajo de [Laguna, 2011], con la condición de definir una configuración que, al ser aplicada, dé como resultado un diagrama de features completamente configurado (es decir que ya no incluya variabilidades) en esta sección se definirá un concepto de configuración que será utilizado para poder realizar la configuración del diagrama de features de la familia de aplicaciones y de los modelos de dominios. Estos conceptos facilitan la implementación de una herramienta para realizar configuraciones.

Teniendo en cuenta los modelos de features definidos en el capítulo 3.4,

sea $FM = (rootF, F, R, D)$ un diagrama de features, tal que

$rootF$ es la feature raíz de FM ,

F es el conjunto de features y referencias de FM ,

R es el conjunto de relaciones incluidas en FM ,

D es el conjunto de restricciones de dependencia incluidas en FM ,

min la cardinalidad mínima de una relación $r \in R$ y max la cardinalidad máxima de r .

Decimos que r representa una variabilidad si

1. r es del tipo *SingleRelation* y $r.children = \{f\}$ donde f es de tipo *ParameterFeature* (*variabilidad de parámetro*)
2. r es del tipo *SingleRelation* y $min = 0, max = 1$ (*variabilidad opcional*)
3. r es del tipo *GroupRelation* y ($min \neq \#(r.children)$ o $max \neq \#(r.children)$) (*variabilidad de selección de variantes*)

Se define:

- Una *configuración de variabilidad CV* es un par (r, S) , donde $r \in R$ representa una variabilidad y $S \subseteq r.children$
- Una *configuración de un modelo de features F* es un conjunto C de configuraciones de variabilidad tal que:
 1. *Su aplicación no contiene variabilidades.* El resultado de aplicar C a F da como resultado un diagrama de features F' tal que R' no contiene relaciones que representan variabilidad.
 2. *Si una feature hija variante es seleccionada de una feature variante padre, entonces la feature variante padre debe ser seleccionada.* Dada una configuración $cv = (r, S)$ y sea f una feature,

tal que $f \in r.children$ y $f \in S$ entonces si $cv_p = (r_p, S_p)$ es una configuración y f_p una feature, tal que $f_p \in r_p.children$ y $r \in f_p.children$ entonces $f_p \in S_p$.

3. *Se resuelven variabilidades en orden jerárquico.* Dada una configuración $cv = (r, S)$ y sea f una feature, tal que $f \in r.children$ y $f \notin S$. Entonces C no debe incluir ninguna configuración cuyas features sean de la rama de descendientes de f . Esto es por motivos de eficiencia, no se deben incluir variabilidades innecesarias en la configuración, si no fue seleccionado el padre no se configura nada más de esa rama.
4. *Una variabilidad solo puede tener una configuración posible.* Dada las configuraciones $cv = (r, S)$ y $cv' = (r', S')$ tal que $r = r'$ entonces $S = S'$ y por lo tanto $cv = cv'$.
5. *Se preservan las restricciones de dependencia de tipo Requires.* Dada una configuración $cv = (r, S)$ y sea f una feature, tal que $f \in r.children$, $f \in S$ y existe $r \in R$ tal que $r.source = f$ y r es de tipo Requires, entonces todas las features del conjunto $r.target$ deben estar incluidas en alguna configuración (si $f' \in r.target$ entonces existe alguna configuración C' que incluye alguna configuración de variabilidad $cv' = (r', S')$ tal que $f' \in S$).
6. *Se preservan las restricciones de dependencia de tipo Excludes.* Dada una configuración $cv = (r, S)$ y sea f una feature, tal que $f \in r.children$, $f \in S$ y existe $r \in R$ tal que $r.source = f$ y r es de tipo Requires, entonces ninguna feature del conjunto $r.target$ deben estar presente en alguna configuración (si $f' \in r.target$ entonces no existe ninguna configuración $cv' = (r', S')$ tal que $f' \in S$).

Ejemplos. A continuación se presenta una configuración posible para el diagrama de features, generado a partir del modelo de casos de uso del caso de estudio, del Sistema integrado de manejo de bibliotecas online de las figuras 3.30 y 3.31 (features de más alto nivel, y desarrollo de features de features de más alto nivel de subsistema OPAC y subsistema My account de OPAC):

$SIMBOConf = \{ (\text{Public search? 1..1, \{Public search\}}, (\text{Cart 0..1, \{cart\}}), (\text{Comments 0..1, \{\}}),$

$(\text{Renewal 0..1, \{renewal\}}), (\text{Holds 0..1, \{holds\}}), (\text{See most checked out items 0..1, \{See most checked out items\}}), (\text{Lists 0..1, \{lists\}}), (\text{Browse subject authorities 0..1, \{\}}), (\text{Authorities records search 0..1, \{\}}), (\text{My account 0..1, \{my account\}}), (\text{Tags 0..1, \{tags\}}), (\text{Search history 0..1, \{search history\}}), (\text{Fines 0..1, \{fines\}}), (\text{Show fines summary 0..1, \{show fines summary\}}), (\text{Show links 0..1, \{show links\}}), (\text{Change my password 0..1, \{change my password\}}), (\text{Details management 1..1, \{change my details\}}), (\text{Show my privacy settings 0..1, \{show my privacy settings\}}), (\text{Show my messaging 0..1, \{show my messaging\}}), (\text{Self renewing 0..1, \{self renewing\}}), (\text{purchase suggestions 0..1, \{purchase suggestions\}}), (\text{Reading history 0..1, \{reading history\}}) \}$

La configuración $SIMBOConf$ para el diagrama de features generado a partir del modelo de casos

de uso del caso de estudio, del Sistema de Administración de Bibliotecas online incluye la resolución de todas las variabilidades que se generaron en el diagrama de features automáticamente a partir del modelo de casos de uso con variabilidades (las partes de más alto nivel y el desarrollo del subsistema *OPAC* y de los niveles más altos del subsistema *My account*). Se incluyen 2 configuraciones de variabilidades de tipo selección de variantes (con relaciones de tipo *GroupRelation*) -*Public Search?* y *Details management*- con cardinalidad 1..1 de las cuales una feature hija de la relación fue seleccionada en la configuración. El resto de las configuraciones de variabilidad incluidas son del tipo variabilidad opcional (relaciones *SingleRelation* con cardinalidad 0..1) y presentan o bien una feature hija seleccionada (como la gran mayoría ya que al representar paquetes o casos de uso de alto nivel son considerados importantes) que significa que se ha seleccionado esa característica o bien un conjunto vacío (como el caso de la relación *Browse subject authorities*) que significa que no se ha seleccionado esa característica.

A continuación se presenta una configuración posible para el diagrama de features de las figuras 6.4, 6.5 y 7.5 del caso de uso del caso de estudio *Search catalog in the OPAC*:

$SearchCatalogConf = \{ (\text{Basic search type 1..1, \{basic search for words\}}, (\text{Extended search option 1..1,$

$\{\text{advanced search}\}), (\text{Filter lost items 0..1, \{filter lost items\}}), (\text{Place multiitem hold 0..1, \{\}}), (\text{Indicate cart to add records 0..1, \{indicate cart to add records\}}), (\text{Indicate list to add records 0..1, \{indicate list to add records\}}), (\text{Results-display 1..1, \{normal view\}}), (\text{PlaceHold-enabled 0..1, \{\}}), (\text{TagsInputOnLists$

0..1, {Request add tag}), (Request add to list 0..1, {request add to list}), (Request add to cart 0..1, {request add to cart}), (Item type icon 0..1, {item type icon}), (Autorished value image 0..1, { }) }

La configuración *SearchCatalogConf* para el caso de uso del caso de estudio *Search catalog in the*

OPAC incluye la resolución de todas las variabilidades incluidas en el diagrama de features generado automáticamente para el caso de uso a partir del diagrama de actividad con variabilidades y el modelo de interfaz de usuario abstracto con variabilidades del caso de uso. Se incluyen tres configuraciones de variabilidades de tipo selección de variantes (con relaciones de tipo *GroupRelation*) -*Basic search type*, *Extended search option* y *Results-display*- con cardinalidad 1..1 de las cuales una feature hija de la relación fue seleccionada en la configuración. El resto de las configuraciones de variabilidad incluidas son del tipo variabilidad opcional (relaciones *SingleRelation* con cardinalidad 0..1) y presentan o bien una feature hija seleccionada (como el caso de la relación *Filter lost items*) que significa que se ha seleccionada esa característica o bien un conjunto vacío (como el caso de la relación *PlaceHold-enabled*) que significa que no se ha seleccionado esa característica.

A continuación se presenta una configuración posible para el diagrama de features de las figuras 6.6, 6.7 y 7.6 del caso de uso del caso de estudio *Show record in the OPAC*:

ShowRecordOpacConf = { (Biblio default view 1..1, {format in simple view}), (OPAC XSLT display view 1..1, {normal record view}), (marc flavour 1..1, {format in MARC21}), (OPAC FBRrized Editions 0..1, {others editions}), (Display other editions 0..1, {display others editions}), (Comments 0..1, { }), (Indicate list to add records 0..1, {indicate list to add records}), (PlaceHold-enabled 0..1, { }), (TagsInputOnLists 0..1, {Request add tag}), (Request add to list 0..1, {request add to list}), (Request add to cart 0..1, {request add to cart}), (Display checkout name 0..1, {display checkout name}), (Item type icon 0..1, {item type icon}), (Autorished value image 0..1, { }) }

La configuración *ShowRecordOpacConf* para el caso de uso del caso de estudio *Show record in the*

OPAC incluye la resolución de todas las variabilidades incluidas en el diagrama de features generado automáticamente para el caso de uso a partir del diagrama de actividad con variabilidades y el modelo de interfaz de usuario abstracto con variabilidades del caso de uso. Se incluyen tres configuraciones de variabilidades de tipo selección de variantes (con relaciones de tipo *GroupRelation*) -*Biblio default view*, *marc flavour* y *OPAC XSLT display view*- con cardinalidad 1..1 de las cuales una feature hija de la relación fue seleccionada en la configuración. El resto de las configuraciones de variabilidad incluidas son del tipo variabilidad opcional (relaciones *SingleRelation* con cardinalidad 0..1) y presentan o bien una feature hija seleccionada (como el caso de la relación *FBRrized Editions*) que significa que se ha seleccionada esa característica o bien un conjunto vacío (como el caso de la relación *Comments*) que significa que no se ha seleccionado esa característica.

9.2. Guías para configurar diagramas de features

Habiendo definido una configuración de la manera que se dice en la sección anterior, el siguiente paso es configurar un diagrama de features que contiene variabilidades (representando una familia de aplicaciones) utilizando una configuración, para así obtener un diagrama de features sin variabilidades (que representa todo lo que una aplicación deberá incluir), y que puede resultar de utilidad para validaciones con el cliente en un formato simple y claro como son los diagramas de features.

Si bien en este trabajo no se brinda un proceso de configuración de manera formal, se brindan algunas guías que podrán ser tenidas en cuenta para facilitar esta tarea. Dado $FM = (rootF, F, R, D)$ un diagrama de features con variabilidades y sea C una configuración, entonces:

- Si cv es una configuración de variabilidad de parámetro de C tal que $cv = (r, \{f\})$ y $r.children = \{f\}$ entonces se deben ajustar los valores de $r.min$ y $r.max$ a los valores deseados.
- Si cv es una configuración de variabilidad opcional de C tal que $cv = (r, \{f\})$ y $r.children = \{f\}$ entonces se debe ajustar el valor $r.min := 1$.
- Si cv es una configuración de variabilidad opcional de C tal que $cv = (r, \emptyset)$ y $r.children = \{f\}$ entonces se debe remover f de F , r de R y todas las restricciones de dependencia que involucren a

f.

- Sea $cv = (r, S)$ una configuración de variabilidad de selección de variantes tal que $cv = (r, S)$ y $r.children = S$ entonces se debe ajustar el valor $r.min := |S|$
- Sea $cv = (r, S)$ una configuración de variabilidad de selección de variantes tal que $cv = (r, S)$ y $S' = r.children - S$ entonces para cada feature $f \in S'$ se debe remover f de F , f de $r.children$ y todas las restricciones de dependencia que involucren a f . Además se debe ajustar $r.min := |S|$ y $r.max := |S|$. Si con el resultado de los ajustes anteriores, tenemos que $|r.children| = 1$ entonces se debe cambiar el tipo de r de GroupRelation a SingleRelation.

Ejemplos. La figura 9.1 muestra el resultado de la aplicación de la configuración *SIMBOConf* al diagrama de features de la figura 3.30. La única configuración de variabilidad de selección de variantes de la configuración *SIMBOConf* es *Public search?*, para la cual se decidió optar por el variante *Public search* que es el incluido en el diagrama de features configurado, eliminando la relación grupal y creando una relación simple con cardinalidad 1..1. El resto de las configuraciones de variabilidades de la configuración *SIMBOConf* son variabilidades opcionales, las seleccionadas fueron incluidas en el diagrama de features configurado, y las que no han sido seleccionadas (como el caso de *Comments*) fueron eliminadas las features y las relaciones correspondientes.

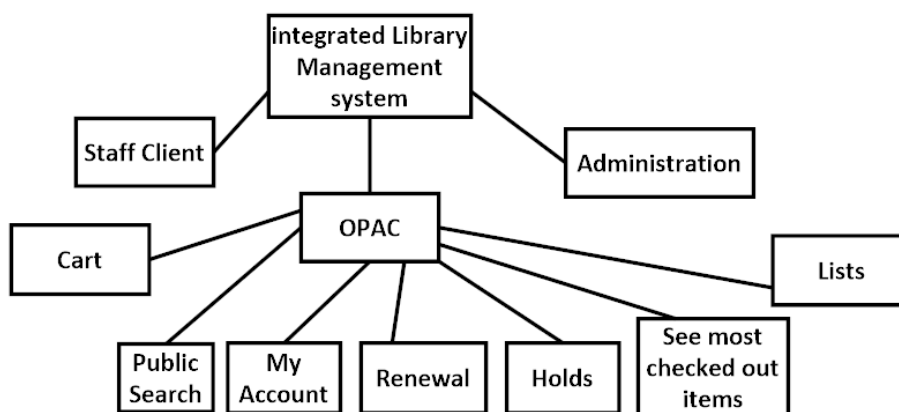


Figura 9.1: Diagrama de features configurado (aplicando la configuración *SIMBOConf*) para el caso de estudio: Sistema de Administración de Bibliotecas Online, incluyendo las features de más alto nivel y desarrollando parte del subsistema OPAC.

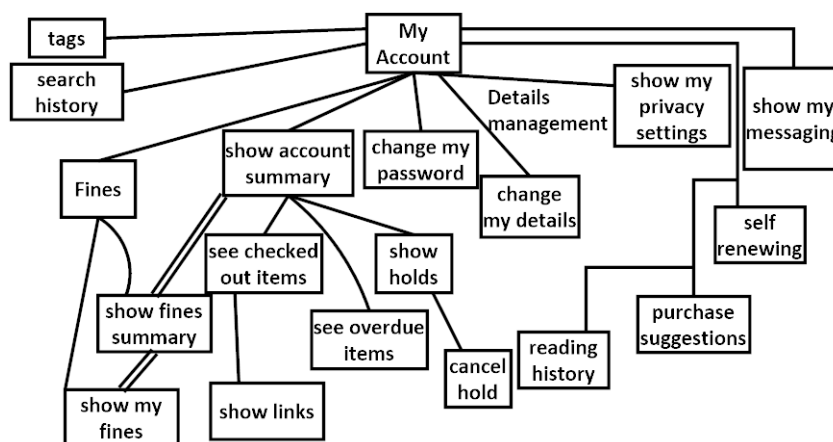


Figura 9.2: Diagrama de features configurado (aplicando la configuración *SIMBOConf*) para los niveles más altos del paquete My account perteneciente al subsistema OPAC del caso de estudio Sistema de Administración de Bibliotecas Online.

La figura 9.2 muestra el resultado de la aplicación de la configuración *SIMBOConf* al diagrama de features de la figura 3.31.

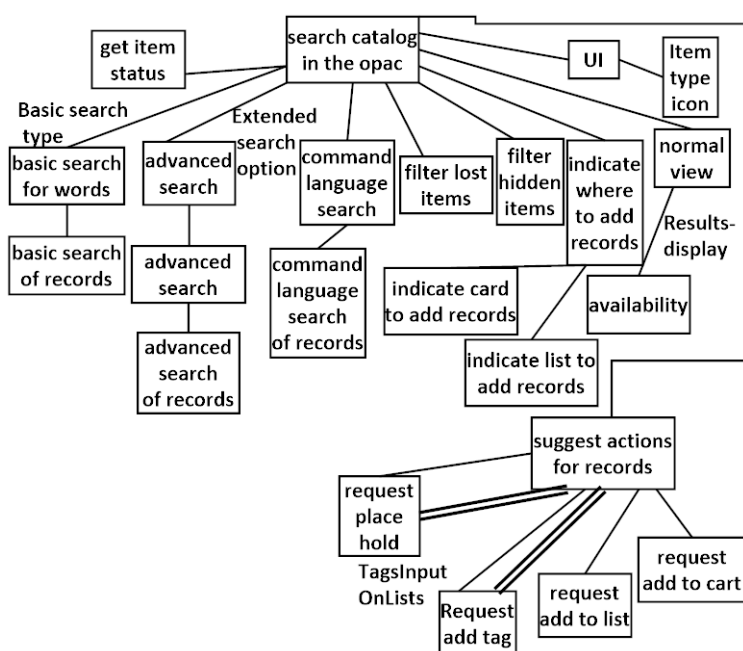


Figura 9.3: Diagrama de features configurado (aplicando la configuración *SearchCatalogConf*) para el diagrama de features generado a partir de la descripción e interfaz abstracta del caso de uso *Search catalog in the OPAC* del caso de estudio Sistema de Administración de Bibliotecas Online.

La figura 9.3 muestra el resultado de la aplicación de la configuración *SearchCatalogConf* a los diagramas de features de la figuras 6.4, .

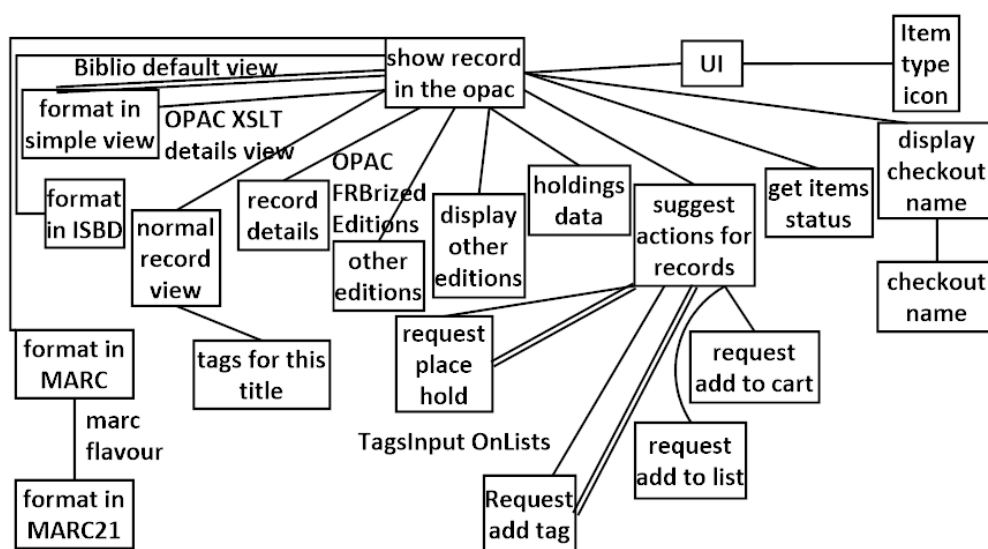


Figura 9.4: Diagrama de features configurado (aplicando la configuración *ShowRecordOpacConf*) para el diagrama de features generado a partir de la descripción e interfaz abstracta del caso de uso *Show Record in OPAC* del caso de estudio Sistema de Administración de Bibliotecas Online.

La figura 9.4 muestra el resultado de la aplicación de la configuración *SearchCatalogConf* a los diagramas de features de la figuras 6.4

Capítulo 10

Configuración de modelos de dominio

10.1. Configuración de diagramas de Casos de Uso de dominio

El siguiente paso en el proceso de desarrollo de la producción de la interfaz de usuario abstracta de una aplicación de este trabajo, es la especialización de los diagramas de casos de uso con variabilidades de la familia de aplicaciones. Los encargados de llevar a cabo esta tarea (analistas o especialistas en familias de aplicaciones) reciben con entradas el modelo de features generado automáticamente, los diagramas de casos de uso con variabilidades de la familia de aplicaciones, los requisitos de la aplicación y, opcionalmente, el SIG de la aplicación.

Cuando ya se cuenta con la definición de la configuración de la aplicación, el siguiente paso en la producción de los diagramas de casos de uso de la aplicación, es aplicar las decisiones tomadas en la configuración en los diagramas de casos de uso con variabilidades de la familia de aplicaciones.

Cada elemento de la configuración definida se corresponde (por sus nombres) con una variabilidad y un conjunto de variantes seleccionados de esa variabilidad, lo cual brinda toda la información necesaria para resolver las variabilidades de los diagramas de casos de uso de la familia de aplicaciones.

Se obtienen los diagramas de casos de uso de una aplicación aplicando sistemáticamente (hasta que no se incluya ninguna variabilidad en los mismos) las siguientes reglas (no es necesario que sean aplicadas en orden):

1. Para más eficiencia, las variabilidades deben ser resueltas en orden jerárquico del diagrama de casos de uso. Es decir si un paquete variante contiene otros paquetes variantes, primero se debe resolver la variabilidad del paquete de más alto nivel.
2. Si cv es una configuración de variabilidad opcional tal que $cv = (r, \emptyset)$ y se corresponde (de acuerdo a su nombre r) a una variabilidad del tipo *PackageVariability* con único elemento p , entonces se deben descartar del diagrama de casos de uso los siguientes elementos: la anotación de variabilidad correspondiente a p , p , los paquetes de casos de uso incluidos en p (y recursivamente sus elementos dentro), todos los casos de uso dentro de p , asociaciones y relaciones incluidas en p . Se debe hacer excepción de los elementos relacionados a otros elementos externos a p , los cuales solamente serán excluidos de p pero permanecen en el diagrama de casos de uso.
3. Si cv es una configuración de variabilidad opcional tal que $cv = (r, \{l\})$ y se corresponde (de acuerdo a su nombre r) a una variabilidad del tipo *PackageVariability* con único elemento p cuyo nombre es igual a l entonces se debe descartar del diagrama de casos de uso la anotación de variabilidad correspondiente a p y se debe incluir p en el diagrama de casos de uso de la aplicación.
4. Si cv es una configuración de variabilidad opcional tal que $cv = (r, \emptyset)$ y se corresponde (de acuerdo a su nombre r) a una variabilidad del tipo *AssociationVariability*, *ExtendVariability* o *IncludeVariability* con único elemento v , entonces se deben descartar del diagrama de casos de uso los siguientes elementos: la anotación de variabilidad correspondiente a v , la relación v incluida en la variabilidad (*Association*, *Include* o *Extensión* según corresponda) y el caso de uso relacionado a v ; el cual será

el caso de uso asociado al Association, el caso de de uso inclusión de la relación Include o el caso de uso extensión de la relación Extende, según corresponda.

5. Si cv es una configuración de variabilidad opcional tal que $cv = (r, \{l\})$ y se corresponde (de acuerdo a su nombre r) a una variabilidad del tipo *AssociationVariability*, *ExtendVariability* o *IncludeVariability* con único elemento v cuyo nombre es igual a l entonces se debe descartar del diagrama de casos de uso la anotación de variabilidad correspondiente a v y se debe incluir v y el caso de uso relacionado a v en el diagrama de casos de uso de la aplicación.
6. Sea $cv = (r, S)$ una configuración de variabilidad de selección de variantes que se corresponde (de acuerdo a su nombre) a una variabilidad del tipo *PackageVariability* que tiene asociado un conjunto P de paquetes de casos de uso se debe descartar del diagrama de casos de uso la anotación de variabilidad correspondiente. Y por cada elemento de P que no está en S (de acuerdo a su nombre) se deben descartar del diagrama de casos de uso los siguientes elementos: la anotación de variabilidad correspondiente (en caso de que ya no haya sido descartada), el paquete referenciado por la variabilidad, los paquetes de casos de uso incluidos en el paquete (y recursivamente sus elementos dentro), todos los casos de uso dentro del paquete, asociaciones y relaciones incluidas en el paquete. Se debe hacer excepción de los elementos relacionados a otros elementos externos al paquete, los cuales solamente serán excluidos del paquete pero permanecen en el digrama de casos de uso. Los elementos de P que están en S deben ser incluidos en el diagrama de casos de uso de la aplicación.
7. Sea $cv = (r, S)$ una configuración de variabilidad de selección de variantes que se corresponde (de acuerdo a su nombre) a una variabilidad del tipo *AssociationVariability*, *ExtendVariability* o *IncludeVariability* que tiene asociado un conjunto R de relaciones de algunos de los tipos anteriores, se debe descartar del diagrama de casos de uso la anotación de variabilidad correspondiente. Y por cada elemento de R que no está en S (de acuerdo a su nombre), se deben descartar del diagrama de casos de uso los siguientes elementos: la anotación de variabilidad correspondiente, la relación incluida en la variabilidad (Association, Include o Extensión según corresponda) y el caso de uso relacionado; el cual será el caso de uso asociado al Association, el caso de de uso inclusión de la relación Include o el caso de uso extensión de la relación Extend, según corresponda. Los elementos de R que están en S deben ser incluidos en el diagrama de casos de uso de la aplicación.

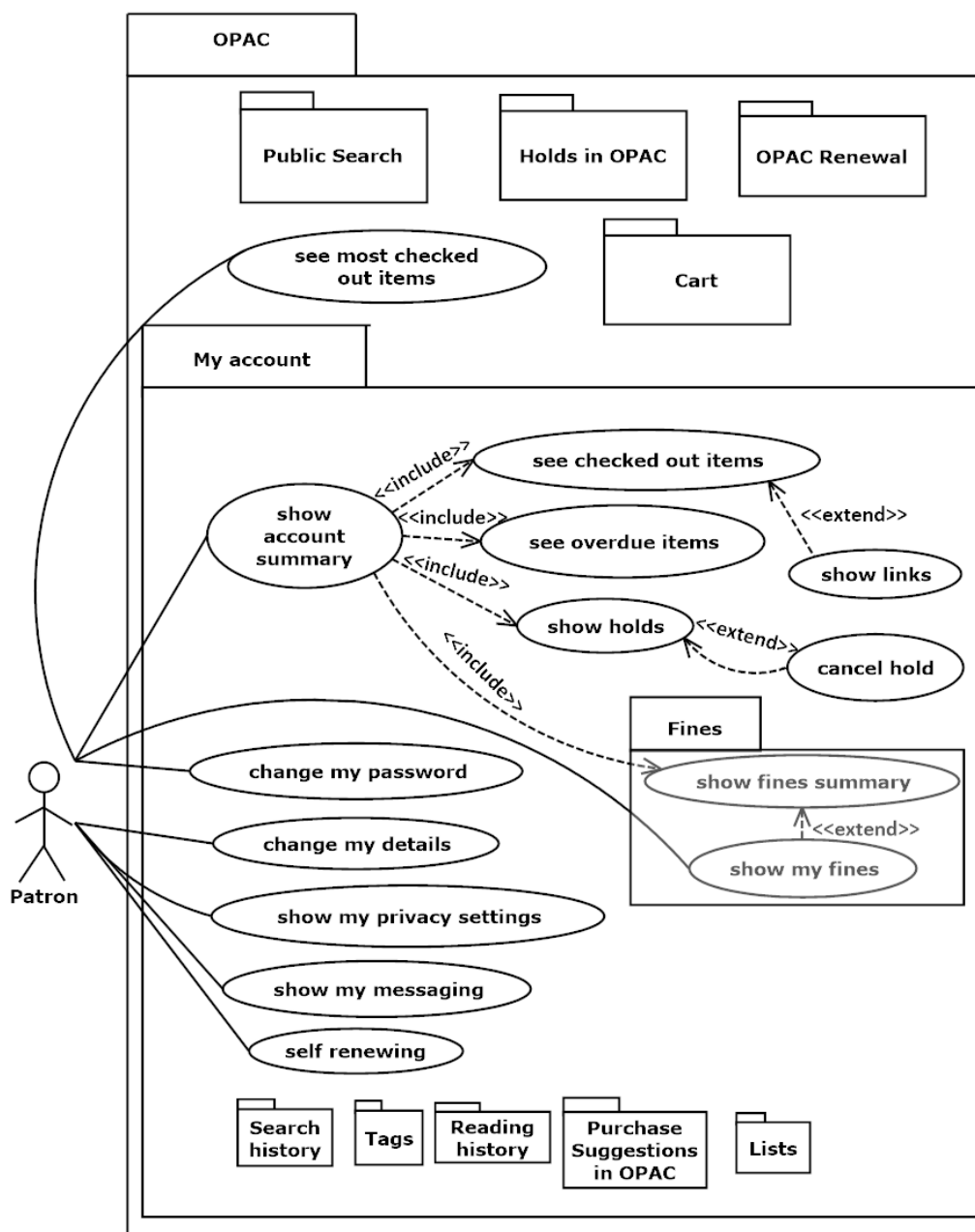


Figura 10.1: Diagrama de casos de uso configurado (aplicando la configuración *SIMBOConf*) para el paquete de casos de uso más alto nivel del subsistema OPAC del caso de estudio Sistema de Administración de Bibliotecas Online.

10.2. Configuración de diagramas de actividad de dominio

Para continuar con la producción de la interfaz de usuario abstracta de una aplicación, luego de tener completados los casos de uso de la aplicación, es necesario describir cada de uso a través de diagramas de actividades. Para esta etapa se cuenta con los casos de uso de la aplicación, los diagramas de actividad con variabilidades de la familia de aplicaciones, la configuración que se ha definido, opcionalmente el SIG de la aplicación y los requisitos de la aplicación.

De manera análoga que para casos de uso, la primera actividad del proceso de producción de diagramas de actividad de la aplicación es la configuración de los diagramas de actividad de la familia de aplicaciones, para así obtener los diagramas de actividad de la aplicación.

Dada una configuración C y un diagrama de actividad con variabilidades ADV , para aplicar la configuración C a ADV se deben considerar los siguientes criterios: 1) el tipo de variabilidad (control flow variability, data flow variability, parameter set), 2) si la variabilidad es opcional o es de selección de variantes. Típicamente, la aplicación de una configuración de variabilidad actúa sobre un diagrama de actividad eliminando del mismo los variantes no elegidos, lo cual comprende eliminar nodos y aristas correspondientes al variante y agregar las aristas necesarias para evitar que queden nodos desconectados. Cada aplicación de una configuración de variabilidad relacionada con el diagrama de actividad con variabilidades en cuestión reduce el conjunto de variantes a ser tratados, y si se elimina el último variante no elegido de la variabilidad, entonces se reduce el conjunto de variabilidades a tratar. Cuando no hay más variabilidades a tratar, finaliza el procedimiento de configuración. A continuación se brindan las reglas (no es necesario que sean aplicadas en orden) necesarias para el procedimiento de configuración:

- Si cv es una configuración de variabilidad de parámetro de C tal que $cv = (r, \{f\})$, $r.children = \{f\}$ y f se corresponde a través de su nombre a un elemento x del diagrama de actividad de tipo *ParameterSet*, entonces se deben ajustar los valores $x.min := f.minCardinality$ y $x.max := f.maxCardinality$.
- Si cv es una configuración de variabilidad opcional de C tal que $cv = (r, \{f\})$ y $r.children = \{f\}$ entonces se debe quitar la anotación de variabilidad de ADV del variante que se corresponde con la feature f a través de su nombre.
- Si cv es una configuración de variabilidad opcional de C tal que $cv = (r, \emptyset)$ y $r.children = \{f\}$ entonces se debe:
 - Quitar la anotación de variabilidad de ADV del variante que se corresponde con la feature f a través de su nombre.
 - Si r se corresponde a través de su nombre a una variabilidad de tipo *ControlFlowVariability* y f se corresponde a una *arista de actividad* x : eliminar x , el nodo sucesor a x , la arista sucesora al nodo sucesor de x de ADV y agregar una arista de actividad para conectar los nodos que han sido desconectados.
 - Si r se corresponde a través de su nombre a una variabilidad de tipo *ControlFlowVariability* y f se corresponde a un elemento *IBD* x : eliminar de ADV todos los nodos y aristas de x , el nodo de llegada a x , el nodo de salida de x y agregar una arista de actividad para conectar los nodos que han sido desconectados.
 - Si r se corresponde a través de su nombre a una variabilidad de tipo *DataFlowVariability* y f se corresponde con un *object node* x conectado con dos *object flows* o y o' : eliminar x , o y o' de ADV y agregar una arista de actividad para conectar los nodos que han sido desconectados.
 - Si r se corresponde a través de su nombre a una variabilidad de tipo *DataFlowVariability* y f se corresponde con un variante x para el cual existe *object node* x' tal que x y x' están conectados por un *object flow* c : eliminar x , x' y c de ADV y agregar una arista de actividad para conectar los nodos que han sido desconectados.
 - Si r se corresponde a través de su nombre a una variabilidad de tipo *ConditionVariability* y f se corresponde con un variante x (condición): eliminar x del elemento de ADV que corresponda.
- Si $cv = (r, S)$ es una configuración de variabilidad de selección de variantes tal que $cv = (r, S)$ y $S' = r.children - S$ entonces se debe quitar la anotación de variabilidad de ADV de los variantes correspondientes a r y para cada variante $f \in S'$ se debe:
 - Si r se corresponde a través de su nombre a una variabilidad de tipo *ControlFlowVariability* y f se corresponde a una *arista de actividad* x : eliminar x , el nodo sucesor a x , la arista sucesora al nodo sucesor de x de ADV .
 - Si r se corresponde a través de su nombre a una variabilidad de tipo *ControlFlowVariability* y f se corresponde a un *IBD* x : eliminar de ADV todos los nodos y aristas de x .

Nota: Si luego de aplicar algunas de las dos reglas anteriores el nodo de fork tiene una sola arista que sale de él, entonces se elimina el nodo fork, su nodo join correspondiente, los arcos

dentro y fuera del nodo fork y los arcos dentro y fuera del nodo de join. Los nodos predecesores y sucesores del nodo fork se reconectan con una arista de actividad y los nodos predecesores y sucesores del nodo join se reconectan con una arista de actividad.

- Si r se corresponde a través de su nombre a una variabilidad de tipo *DataFlowVariability* y f se corresponde con un *object node* x conectado con dos *object flows* o y o' : eliminar x , o y o' de *ADV* y agregar una arista de actividad para conectar los nodos que han sido desconectados.
- Si r se corresponde a través de su nombre a una variabilidad de tipo *DataFlowVariability* y f se corresponde con un variante x para el cual existe *object node* x' tal que x y x' están conectados por un *object flow* c : eliminar x , x' y c de *ADV* y agregar una arista de actividad para conectar los nodos que han sido desconectados.
- Si r se corresponde a través de su nombre a una variabilidad de tipo *ConditionVariability* y f se corresponde con un variante x (condición): eliminar x del elemento de *ADV* que corresponda.

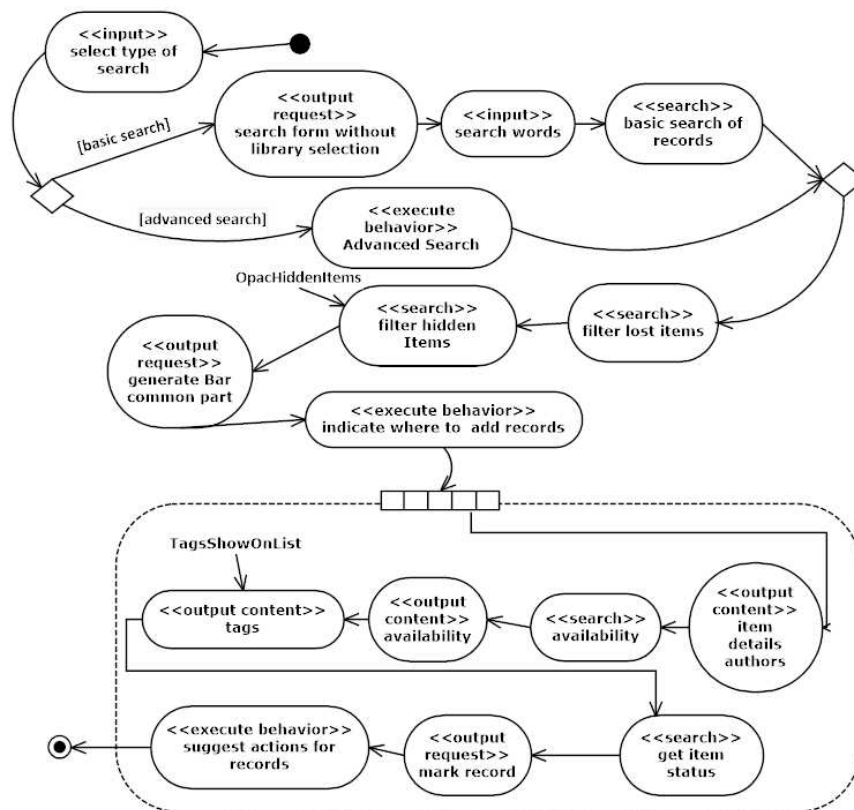


Figura 10.2: Diagrama de actividad configurado (aplicando la configuración *SearchCatalogConf*) para el caso de uso Search catalog in the OPAC del subsistema OPAC.

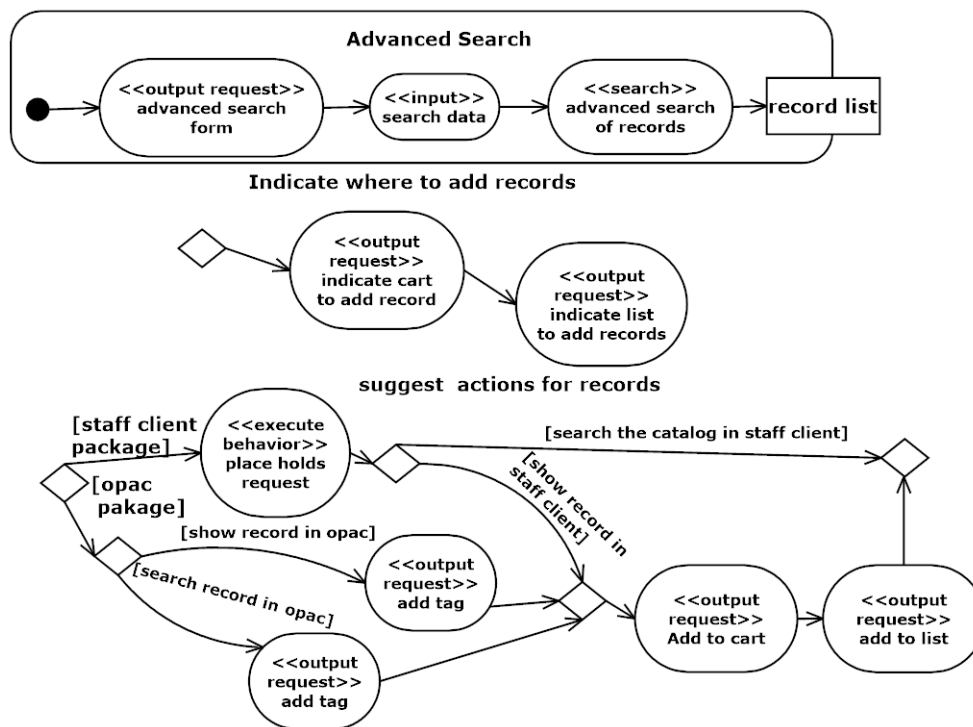


Figura 10.3: Comportamientos reutilizables configurados (aplicando la configuración *SearchCatalogConf*) que aparecen en el caso de uso Search catalog in the OPAC del subsistema OPAC.

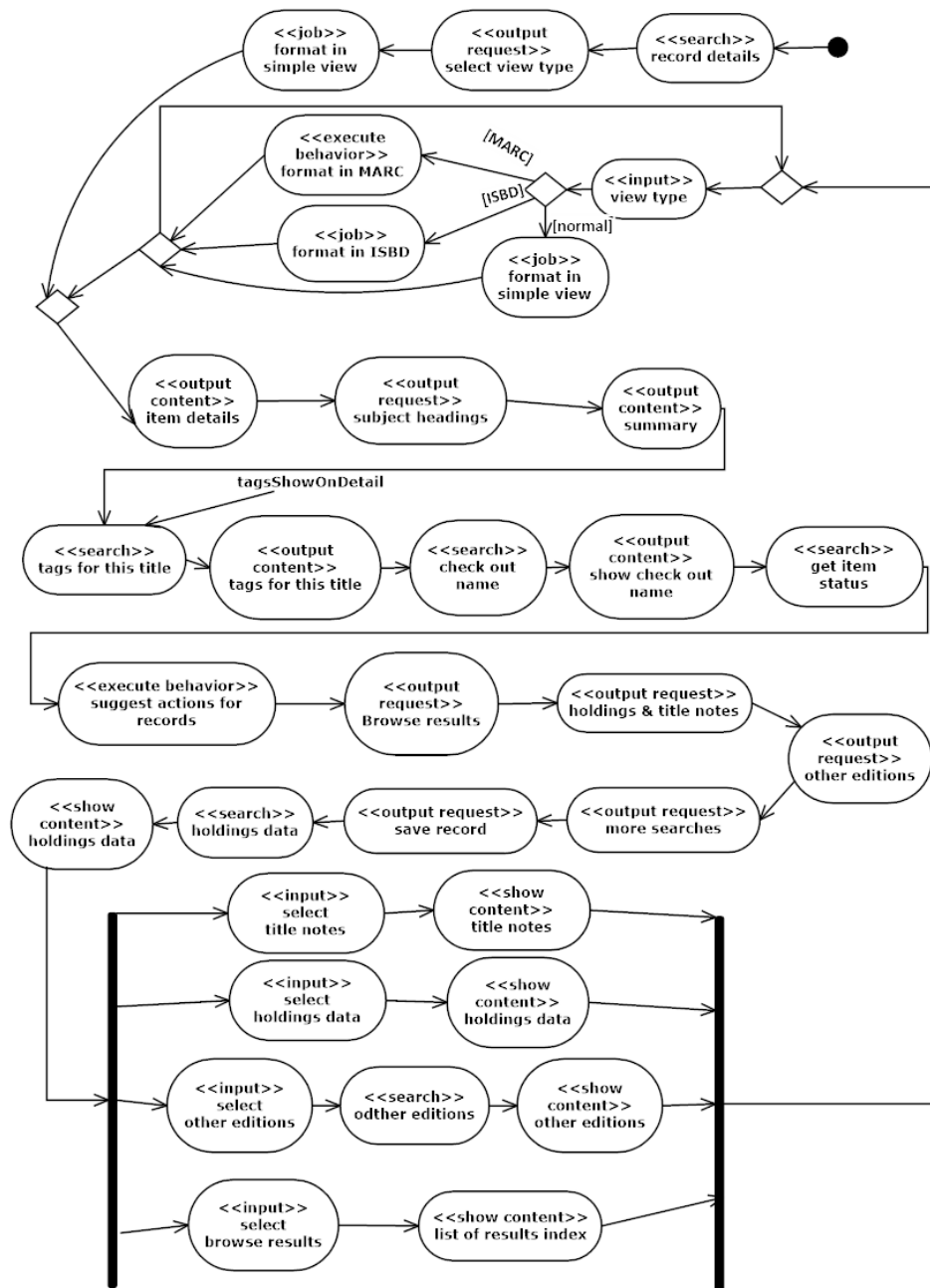


Figura 10.4: Diagrama de actividad configurado (aplicando la configuración *ShowRecordOpacConf*) para el caso de uso Show record in the OPAC del subsistema OPAC.

10.3. Configuración de modelos de interfaz de usuario abstracta de dominio

Dada una configuración C y un modelo de interfaz de usuario abstracta con variabilidades UIM , para aplicar la configuración C a UIM se deben considerar los siguientes criterios: 1) resolver las partes de interfaz de usuario marcadas con “F”, en base a las configuraciones de variabilidad de las etapas anteriores, 2) resolver las configuraciones de variabilidad de este etapa. Típicamente, la aplicación de una configuración de variabilidad actúa sobre un modelo de interfaz de usuario eliminando del mismo los variantes no elegidos, lo cual comprende eliminar elementos del modelo correspondientes al variante y eliminar las marcas que se han realizado en los elementos en base a decisiones de variabilidad de las

etapas anteriores que afectan a elementos de interfaz de usuario. A continuación se brindan las reglas (no es necesario que sean aplicadas en orden) necesarias para el procedimiento de configuración:

- Si cv es una configuración de variabilidad de C tal que $cv = (r, S)$ y existe algún elemento ui de UIM marcado con “**F**” tal que su nombre se corresponde con algún elemento de S , entonces se debe eliminar la marca “**F**” de ui .
- Si cv es una configuración de variabilidad de C tal que $cv = (r, S)$, $S' = r.children - S$ y existe algún elemento ui de UIM marcado con “**F**” tal que su nombre se corresponde con algún elemento de S' , entonces se debe eliminar la marca “**F**” de ui , se debe eliminar ui y todos los elementos contenidos por ui de UIM .
- Si cv es una configuración de variabilidad opcional de C tal que $cv = (r, \{f\})$ y $r.children = \{f\}$ y r se corresponde con una variabilidad adjuntada a algún elemento ui de UIM , entonces se debe eliminar la anotación de variabilidad con nombre r adjuntada al ui .
- Si cv es una configuración de variabilidad opcional de C tal que $cv = (r, \emptyset)$ y $r.children = \{f\}$ y r se corresponde con una variabilidad adjuntada a algún elemento ui de UIM , entonces se debe eliminar la anotación de variabilidad con nombre r adjuntada al ui y se debe eliminar ui y todos los elementos contenidos por ui de UIM .
- Sea $cv = (r, S)$ una configuración de variabilidad de selección de variantes tal que $cv = (r, S)$, $S' = r.children - S$ y r se corresponde con una variabilidad adjuntada a algún conjunto de elementos UI de UIM , entonces se debe eliminar la anotación de variabilidad con nombre r adjuntada a cada elemento de UI y por cada elemento ui de UI cuyo nombre esté presente en S' se debe eliminar ui y todos los elementos contenidos por ui de UIM .

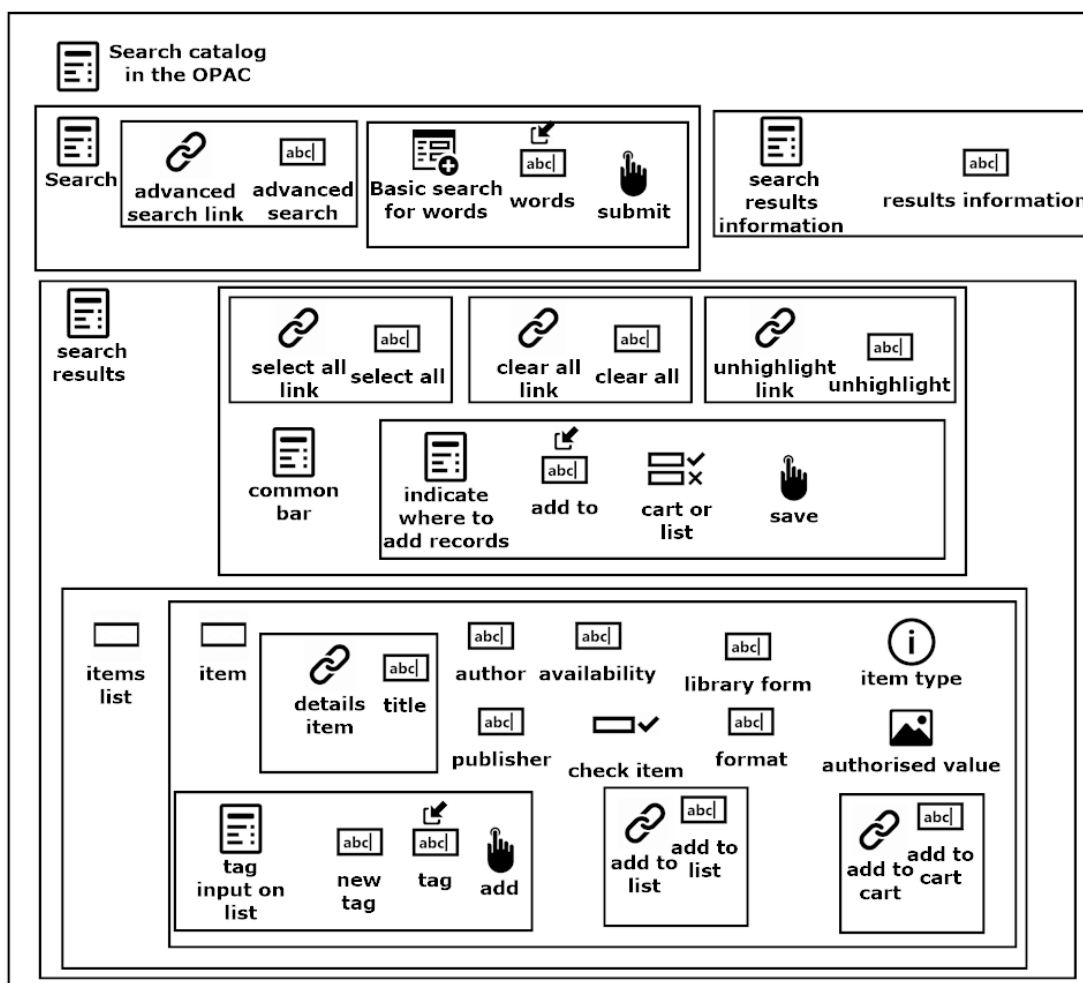


Figura 10.5: Interfaz de usuario abstracta configurada (aplicando la configuración *SearchCatalogConf*) para el caso de uso Search catalog in the OPAC del subsistema OPAC.

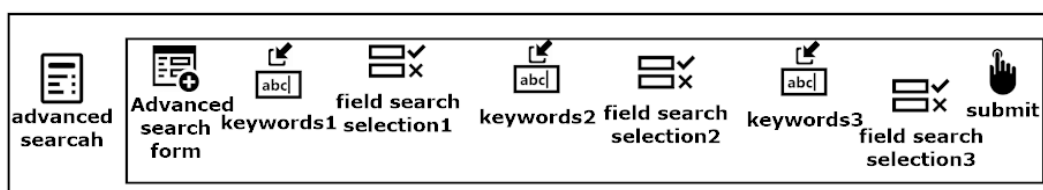


Figura 10.6: Interfaz de usuario abstracta configurada (aplicando la configuración *SearchCatalogConf*) para la funcionalidad de Advanced Search que aparece en el caso de uso Search catalog in the OPAC del subsistema OPAC.

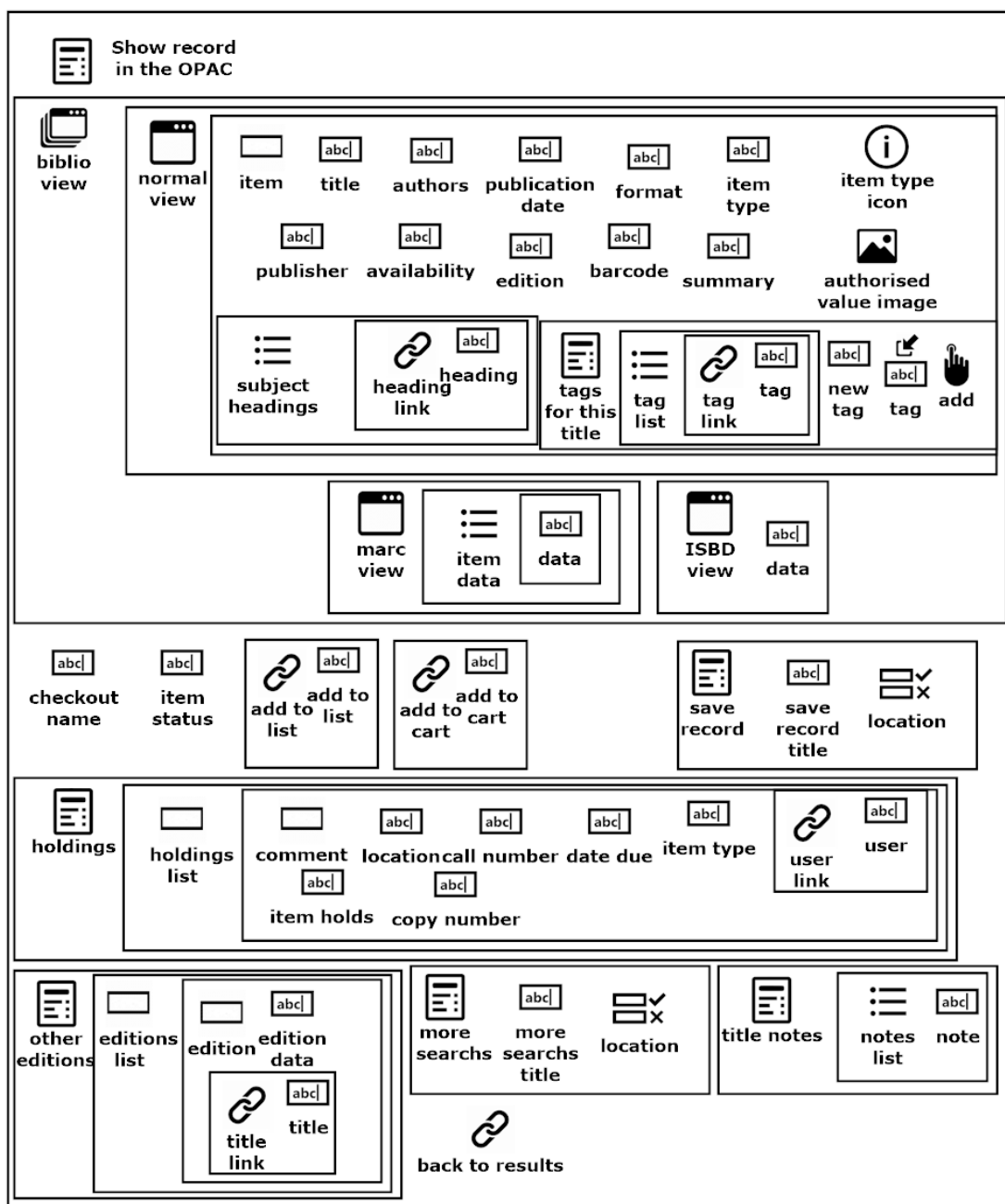


Figura 10.7: Interfaz de usuario abstracta configurada (aplicando la configuración *SearchCatalogConf*) para el caso de uso Show record in the OPAC del subsistema OPAC.

Parte IV

Desarrollo de modelos de aplicación e implementación de interfaz de usuario responsiva

Capítulo 11

Actualización de modelos configurados en base a requisitos de la aplicación

Una vez que se ha realizado la configuración de los modelos de dominio de UML, la primer actividad que se realiza en este trabajo para producir la interfaz de usuario de una aplicación es la de actualizar los modelos de UML configurados, según los requisitos de la aplicación en particular a desarrollar. Cabe aclarar que esta actividad es opcional y depende de si la aplicación a realizar tiene requisitos particulares que no están incluidos en los requisitos de la familia de aplicaciones (esto ayuda a nutrir la familia de aplicaciones con nuevos requisitos).

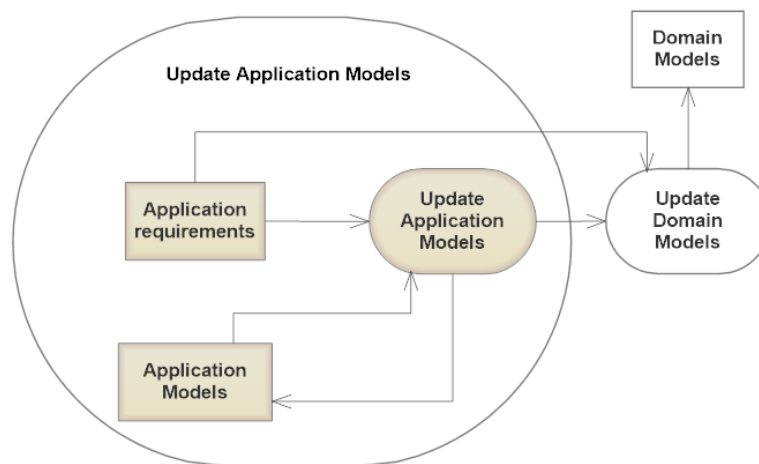


Figura 11.1: Proceso de actualización de modelos de aplicación de UML configurados.

En el caso que se cuenten con requisitos particulares de una aplicación que no están incluidos en la familia de aplicaciones, las actividades a desarrollar son las expresadas en las figura 11.1. Teniendo como entradas los requisitos particulares de la aplicación a modelar y los modelos de la aplicación configurados, se deben actualizar los modelos de la aplicación en base a los nuevos requisitos particulares. Luego, y también basándose en los requisitos particulares de la aplicación, se pueden actualizar los modelos de dominio de UML para enriquecer y aumentar la potencialidad de la familia de aplicaciones. Pero esta actividad involucra dominio y debe ser llevada a cabo como parte de desarrollo dominio. En este trabajo no prescribimos un método para dicha tarea.

En las próximas secciones se dan guías sobre cómo actualizar diagramas de casos de uso, diagramas de actividad y modelos de interfaz de usuario abstracta en base a nuevos requisitos.

11.1. Actualización de diagramas de casos de uso

A partir del documento de requisitos funcionales de la aplicación, nuevas funcionalidades no incluidas entre las funcionalidades de la familia de aplicaciones pueden ser necesarias. Para estos casos, y tomando como punto de entrada los diagramas de casos de uso configurados, nuevos casos de uso y relaciones entre los mismos deben ser agregados. Algunas consideraciones a tener en cuenta son las siguientes:

- *Definir si nuevos actores son necesarios.* En ocasiones puede pasar que al agregar nuevas funcionalidades, es necesario rever los actores que interactúan en las mismas. En el caso de hallar actores no incluidos en el diagrama de casos de uso configurado para la aplicación, éstos deben ser incorporados.
- *Definir nuevos casos de uso.* Para las funcionalidades que no están presentes en la familia de aplicaciones y si en el documento de requisitos de la aplicación, nuevos casos de uso deben ser definidos.
- *Reorganizar el modelo de casos de uso.* En caso de ser necesaria la incorporación de actores y de casos de uso al modelo de casos de uso configurado para la aplicación, nuevas consideraciones de organización de casos de uso deben ser tenidas en cuenta. Esto incluye estructuración en paquetes o inclusión de casos de uso en paquetes, definición de relaciones del tipo Include o Extend para los casos de uso incorporados.

11.2. Actualización de diagramas de actividades

Una vez que se cuenta con los diagramas de actividad configurados, es importante tener en cuenta dos consideraciones: la primera es la incorporación de nuevos casos de uso en la aplicación (a partir de la actividad de la sección 11.1) en base a los requisitos, la segunda es tener en cuenta si en los requisitos de la aplicación surge la necesidad de agregar, modificar o eliminar elementos de los diagramas de actividad generados a partir de los diagramas de actividad con variabilidades de la familia de aplicaciones. Con estas consideraciones en mente, se debe:

- *Incorporar descripciones de casos de uso agregados.* Se deben construir los diagramas de actividad necesarios para los casos de uso incorporados a partir de los requisitos de la aplicación. Para esta tarea se puede seguir cualquier tutorial para construcción de diagramas de actividad, teniendo en cuenta los elementos para reutilización definidos en este trabajo.
- *Actualizar descripciones de casos de uso existentes.* Examinando con detalle el documento de requisitos de la aplicación provisto por el cliente, puede ser necesario realizar distintas actualizaciones a las descripciones de casos (a través de diagramas de actividad) generados a partir de la configuración de los diagramas de actividad con variabilidades. Además otras actualizaciones pueden ser necesarias en base a actualizaciones realizadas a los diagramas de caso de uso de la aplicación. Estas actualizaciones pueden consistir en cambiar nombres de elementos, eliminar elementos manteniendo la consistencia de los diagramas, incorporar elementos manteniendo la consistencia de los diagramas.

11.3. Actualización de interfaz de usuario abstracta

El último paso para completar la interfaz de usuario abstracta de la aplicación que se está desarrollando es el de actualizar los modelos obtenidos a través de la configuración de modelo de interfaz de usuario abstracta con variabilidad y la incorporación, en caso de ser necesario, de nuevas partes de la interfaz de usuario en base a requisitos de la aplicación. Para esta tarea se pueden considerar las siguientes guías (convienen ser aplicadas secuencialmente):

- *Incorporar interfaz de usuario para descripciones de casos de uso agregados.* Si los requisitos de la aplicación a desarrollar dieron lugar a la incorporación de nuevos casos de uso, los cuales posteriormente han sido descritos a través de diagramas de actividad, se debe crear la interfaz de usuario abstracta para estos casos de uso viendo exhaustivamente las descripciones de estos casos de uso y los requisitos de interfaz de usuario de la aplicación.

- *Actualizar interfaz de usuario abstracta en base a modificaciones en etapas anteriores.* Es posible que en las etapas de actualización de casos de usos y diagramas de actividad de la aplicación se hayan realizado cambios a consecuencia de los requisitos de la aplicación. Estos cambios pueden ser, agregar, eliminar o modificar elementos, cambiar nombres de elementos, entre otros. Esto afecta a las interfaces de usuario generadas por medio de la configuración de interfaz de usuario abstracta con variabilidades y es necesario, realizar los cambios que sean necesarios en la misma (agregar, quitar, eliminar o actualizar elementos).
- *Actualizar interfaz de usuario abstracta en base a requisitos.* En base a los modelos de interfaz de usuario generados por medio de la configuración y habiéndoles realizados las modificaciones correspondientes en el punto anterior, la última tarea o consideración es analizar los requisitos de interfaz de usuario propios de la aplicación. Estos pueden dar lugar a modificaciones en los modelos de interfaz de usuario. A manera de ejemplo, un requisitos de interfaz de usuario podría requerir que algún campo de input contenga la característica de “autocomplete”, que no estaba contemplado en la familia de aplicaciones; por lo tanto se deben hacer los cambios necesarios en la interfaz de usuario abstracta para reflejar esto.

Capítulo 12

Desarrollo de modelo de acceso a funcionalidades y contenido

En la sección 2.2.1 motivamos y presentamos la definición de una notación para modelar acceso a funcionalidades y contenido en el ámbito de aplicaciones web y brindamos detalles de los meta-modelos definidos para modelar la parte estática y la parte dinámica del accesos a funcionalidades y contenido.

Para que los modelos de acceso a funcionalidades y contenido no representen modelos aislados dentro de un proceso de desarrollo de interfaz de usuario de aplicaciones y para que estos puedan ser construidos de una manera sistemática, es necesario también definir:

- Cómo realizar una buena transición desde notaciones de requisitos (por ejemplo diagramas de caso de uso) a modelo de acceso a funcionalidades y contenido.
- Cómo realizar una buena transición desde modelo de acceso a funcionalidades y contenido a modelo de interfaz de usuario abstracta.
- Prescribir un método de desarrollo de modelo de acceso a funcionalidades y contenido que capture la esencia de la interfaz de usuario para acceso a funcionalidades y contenido.

El objetivo de esta sección es el estudio y la provisión de guías para la transición de notaciones de requisitos (como diagramas de casos de uso) y partes del modelo de acceso a funcionalidades y contenido provistas por el cliente al modelo completo de acceso a funcionalidades y contenido; la definición de un proceso de desarrollo para acceso a funcionalidades y contenido y la transición del modelo de acceso a funcionalidades y contenido a modelo de interfaz de usuario abstracta.

12.1. Proceso de desarrollo de modelo de acceso a funcionalidades y contenido de una aplicación

Como punto de partida del proceso de desarrollo del modelo de acceso a funcionalidades y contenido, el cliente que solicita el desarrollo de la aplicación puede proveer fragmentos del modelo *WAFCA*. Luego, el analista desarrolla modelos de requisitos (en este trabajo se utilizan diagramas de caso de uso, pero podrían ser otros tipos de modelos tales como modelos de tareas). A continuación, usando los fragmentos provistos por el cliente (en caso de existir) y los modelos de requisitos, el analista desarrollo un modelo *WAFCA* completo, el cual es validado por el cliente. A partir de la validación del cliente, el analista construye un modelo *WAFCA* revisado. Finalmente, se refinan lo elementos del modelo *WAFCA* en elementos de modelos más concretos (elementos de interfaz de usuario sobre un modelo de interfaz de usuario abstracta, eventos sobre un elemento de interfaz de usuario).

- **Fragmentos del modelo *WAFCA* desarrollados por el cliente.**
Se utiliza esta fase para mejorar la satisfacción del cliente. Se asume que el analista carece de conocimientos del dominio que el cliente no puede transmitir fácilmente cuando comunica los requisitos

para una aplicación nueva (realizar esta asunción es una premisa para el área de *End-User-Software-Development*, ver [Paternò, 2013]).

El cliente puede proporcionar dos tipos de fragmentos *WAFCA*:

1. Una descomposición del *Grouping* raíz de la vista del *user role site*, considerando solo los primeros niveles de la descomposición. Se debe expresar el propósito para cada *Grouping* de la descomposición.
2. *Groupings* para conceptos innovadores que involucran elementos de tareas y contenido relacionados al contenido (algunos de ellos pueden ser accesibles a través del elemento *Content*).

■ **Requisitos provistos por el analista.**

En este trabajo utilizamos como modelos de requisitos diagramas de casos de uso. Otros ejemplos de

modelos de requisitos que pueden ser provistos por el analista son modelos de procesos de negocios, y modelos de tareas. Se optó por utilizar modelos de casos de uso de ya que *UML* es una notación ampliamente utilizada (ver [Miles, 2007]).

Los casos de uso pueden ser desarrollados considerando: a) *Groupings* para conceptos innovadores provistos por el cliente. b) Otros requisitos funcionales provistos por el cliente.

■ **Desarrollo de un modelo *WAFCA* completo por medio del analista.**

Se asume que se tienen los fragmentos del modelo *WAFCA* provistos por el cliente y un diagrama

de casos de uso para los requisitos funcionales, sin embargo, para la transición a un modelo completo *WAFCA* no se prescribe un método para esta fase. Independientemente del método que se utilice, es necesario tomar algunas decisiones en tal método, las cuales son:

- **D1:** Si se mapea directamente a un *Grouping* un paquete de casos de uso *P* (con el mismo nombre y conteniendo mapeos de sus casos de y paquetes contenidos), entonces en este caso se debe decidir el tipo del *Grouping P*.
- **D2:** Cómo tratar los paquetes de casos de uso que no son mapeados directamente a un *Grouping*.
- **D3:** Si un caso de uso es mapeado o bien a una tarea (*Task*) o bien a un acceso a agrupamiento (*Acceso to Grouping*).
- **D4:** Cuales son los elementos de contenido que no son provistos por el cliente.
- **D5:** Si la traducción de un caso de uso es accesible desde un contenido o no.
- **D6:** Cuales son los casos de uso que afectan a un elemento de contenido (que lo modifican o lo procesan).
- **D7:** Cuales son los miembros de los *Groupings* de los primeros niveles provistos por el cliente.

Dependiendo del método utilizado, estas decisiones deben ser realizadas ya sea manualmente, automáticamente o semi-automáticamente.

A continuación, el analista expresa los requisitos de un modelo *WAFCA* para cada hijo del *Grouping* raíz de una vista de *user role site* se desarrollo un diagrama de requisitos.

12.1.1. Aplicación al caso de estudio

A continuación se incluyen los diagramas desarrollados para la vista estática del modelo *WAFCA* para el caso de estudio de Sistema de Administración de Bibliotecas Online para el subsistema OPAC.

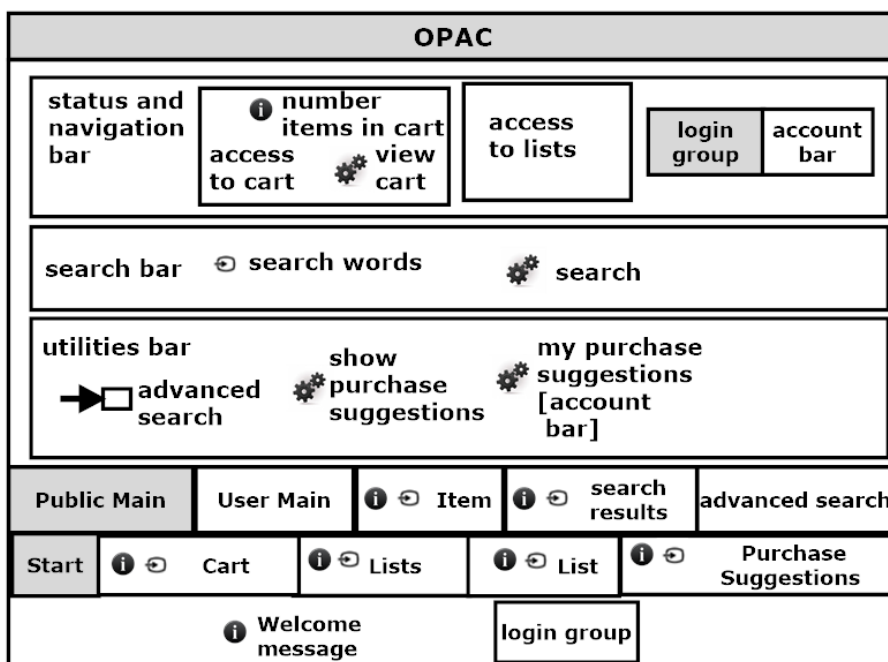


Figura 12.1: Agrupamiento inicial visualizado por un usuario de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

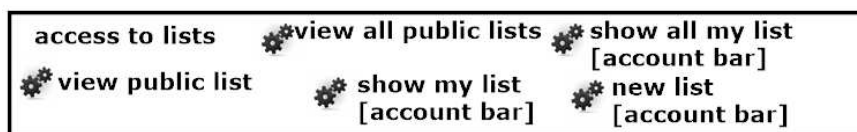


Figura 12.2: Agrupamiento para acceso a listas de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.3: Agrupamiento para ingreso al sistema de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

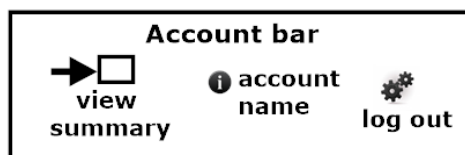


Figura 12.4: Agrupamiento para barra de usuario de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

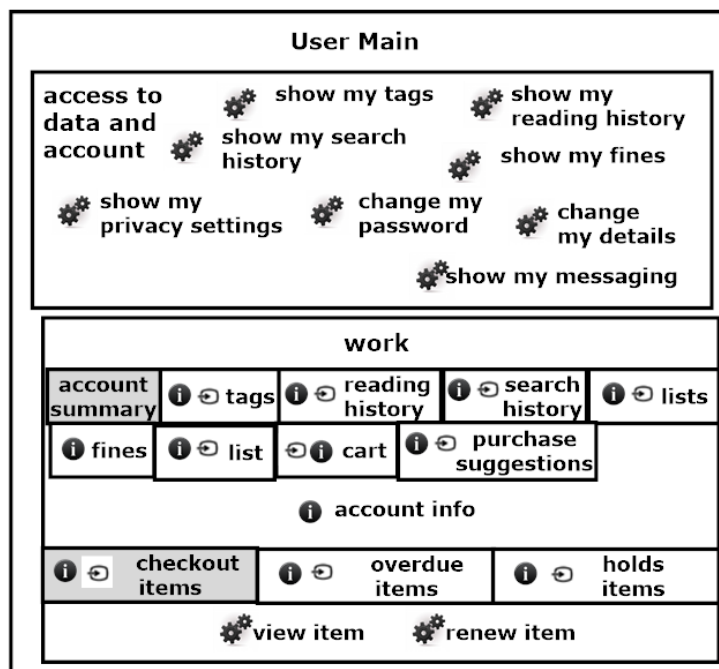


Figura 12.5: Agrupamiento principal para usuario autenticado de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.6: Agrupamiento para el contenido y funcionalidades de un ítem de biblioteca de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

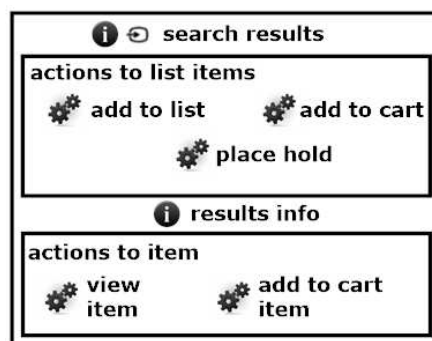


Figura 12.7: Agrupamiento para el contenido y funcionalidades del resultado de búsquedas de ítems de biblioteca de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.8: Agrupamiento para la funcionalidad de búsqueda avanzada de items de biblioteca de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

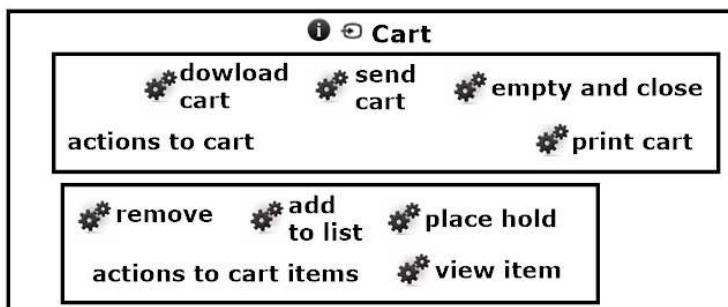


Figura 12.9: Agrupamiento para el elemento de contenido Cart que representa el carrito de préstamos de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.10: Agrupamiento para el elemento de contenido Lists que representa a listas de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

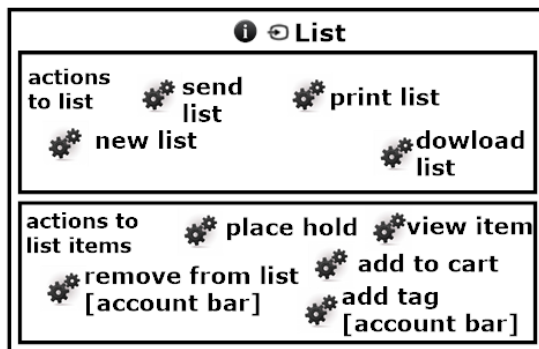


Figura 12.11: Agrupamiento para el elemento de contenido List que representa una lista de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.12: Agrupamiento para el elemento de contenido Purchase suggestions que representa sugerencias de compra de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.13: Agrupamiento para el elemento de contenido Tags que representa etiquetas de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.14: Agrupamiento para el elemento de contenido Reading History que representa historial de búsquedas en el catálogo de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

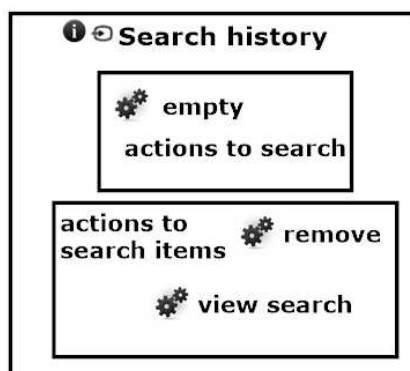


Figura 12.15: Agrupamiento para el elemento de contenido Search History que representa historial de búsquedas en el catálogo de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.



Figura 12.16: Agrupamiento para el elemento de contenido Overdue Items que representa items con retrasos de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

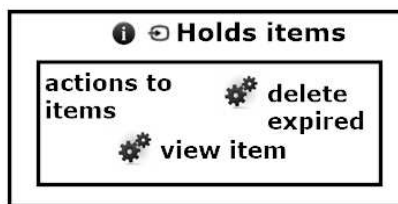


Figura 12.17: Agrupamiento para el elemento de contenido Hold items que representa retenciones de items de la vista estática del modelo WAFCA para el subsistema OPAC del caso de estudio.

A continuación se incluyen los diagramas desarrollados para la vista dinámica del modelo WAFCA para el caso de estudio de Sistema de Administración de Bibliotecas Online para el subsistema OPAC.

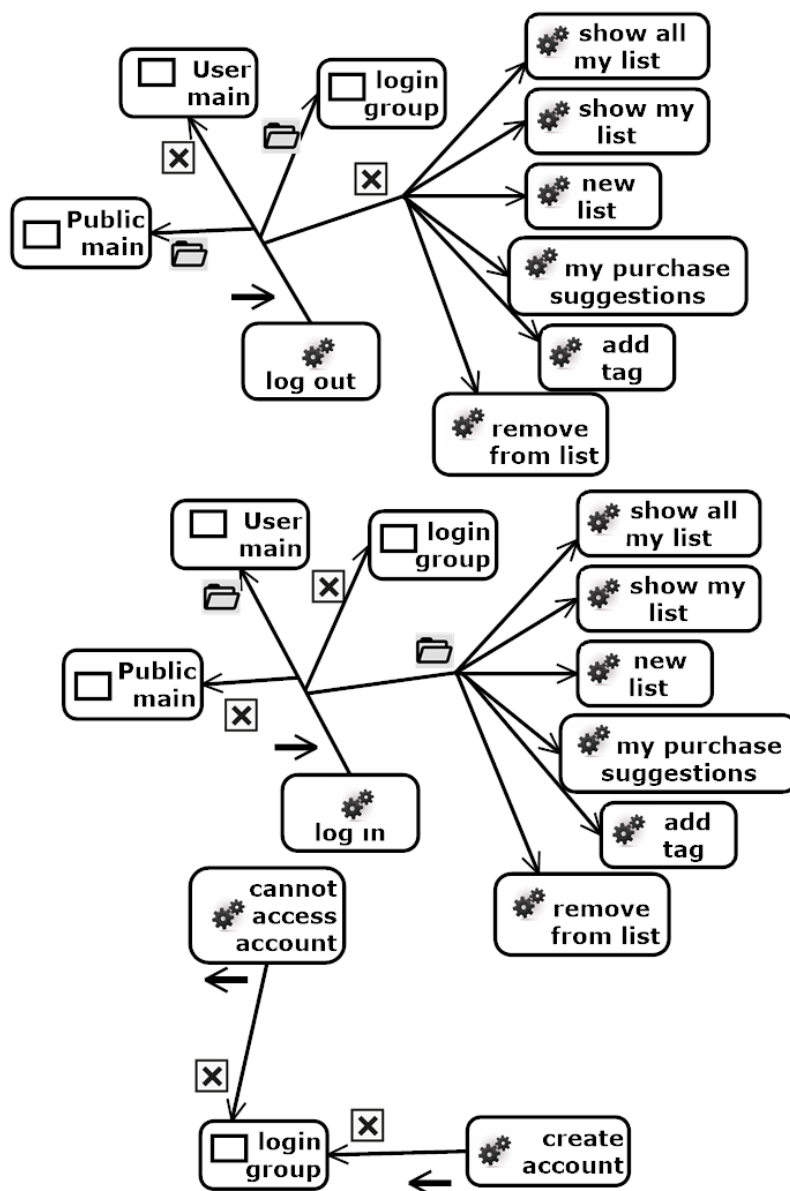


Figura 12.18: Vista dinámica de los elementos Login Group, Account Bar, Public Main, User Main (relacionados a funcionalidades de ingreso y salida del sistema) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio caso de estudio.

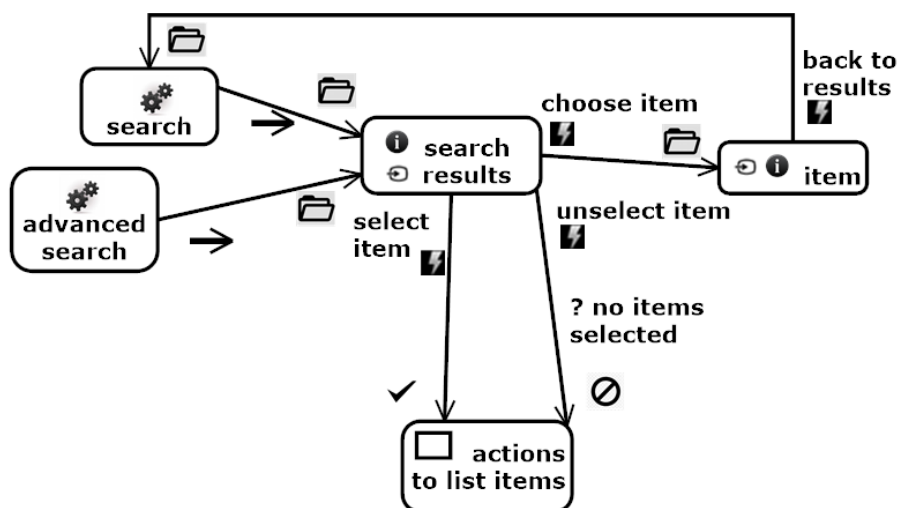


Figura 12.19: Vista dinámica de los elementos Search, Advanced Search (relacionados a funcionalidades de búsquedas de ítems) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio.

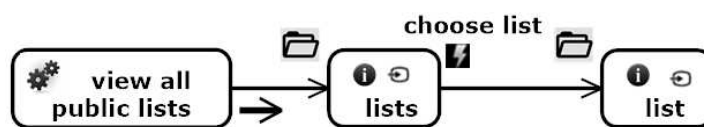


Figura 12.20: Vista dinámica del elemento view all public lists (relacionados a funcionalidades del agrupamiento access to lists) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio. Lo mismo se replica (solo cambia el nombre de la funcionalidad inicial) para show all my lists.

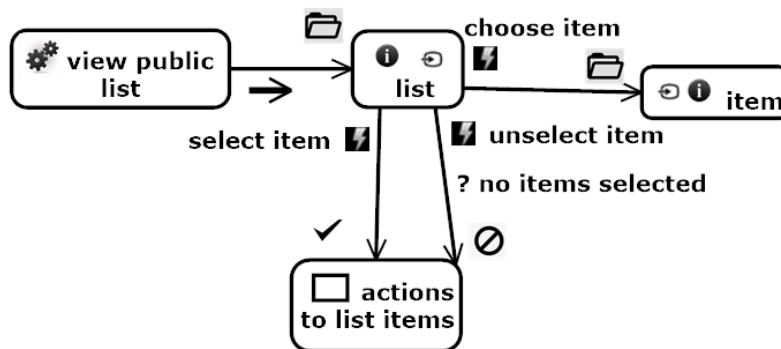


Figura 12.21: Vista dinámica del elemento view public list (relacionados a funcionalidades del agrupamiento access to lists) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio. Lo mismo se replica (solo cambia el nombre de la funcionalidad inicial) para show my list.

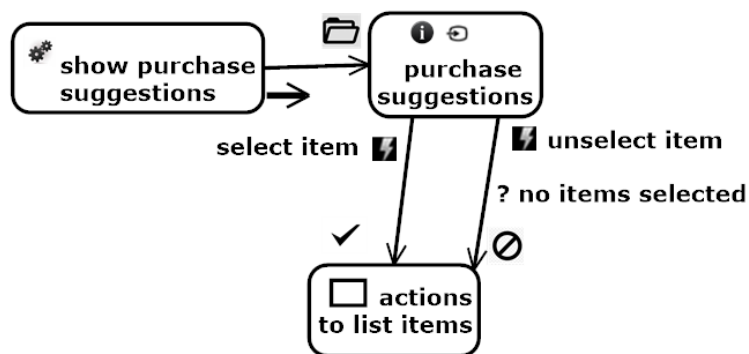


Figura 12.22: Vista dinámica del elemento show purchase suggestions (relacionados a funcionalidades del agrupamiento utilities bar) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio. Lo mismo se replica (solo cambia el nombre de la funcionalidad inicial) para my purchase suggestions.

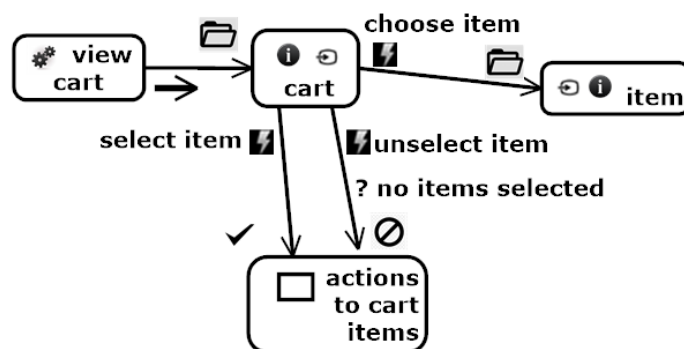


Figura 12.23: Vista dinámica del elemento view cart (relacionados a funcionalidades del agrupamiento access to cart) del grouping raíz del modelo WAFCA para el subsistema OPAC del caso de estudio.

12.2. Definición de relaciones de traza

En esta etapa son refinados los elementos de los tipos *Content* e *Input* de la vista estática del modelo WAFCA y los eventos y condiciones de la vista dinámica del modelo WAFCA. Para realizar este refinamiento se consideran las siguientes tareas:

- Si el modelo de interfaz de usuario es legible por el cliente, entonces el cliente puede proveer elementos de interfaz de usuario que refinan a elementos de contenido (por ejemplo elementos que se correspondan a conceptos innovadores).
- Las relaciones de traza entre elementos de tipos *Content* e *Input* y elementos de interfaz de usuario son construidas utilizando un modelo de interfaz de usuario. No nos preocupamos por cómo obtener estas relaciones de traza (por ejemplo, de forma automática, de forma manual).
- Son construidas las relaciones de traza entre eventos y condiciones en requisitos y eventos atómicos (posiblemente sobre elementos de interfaz de usuario) y condiciones detalladas. No se prescribe un método para obtener éstas relaciones de traza.

Refinamiento de elementos de content e input

En esta sección se consideran las relaciones de traza entre los elementos de tipo Content e Input de la vista estática del meta-modelo WAFCA y elementos del meta-modelo de la sección 2.2.2 para interfaz de usuario abstracta.

Para realizar esta tarea, se pueden seguir las siguientes guías para los elementos que pueden ser refinados del modelo estático de de acceso a funcionalidades y contenido del sistema:

- **Read only Content:** pueden ser refinados a algún elemento de estructura de contenido no editable (*ContentStructure*), a un elemento de tipo *NavList* o a un elemento de tipo *UiElement* no editable.
- **Content with interaction:** pueden ser refinados a un elemento de *ContentStructure* (quizás editable o no), a un elemento de tipo *NavList* (incluyendo posiblemente un elemento *NavigationBar*), a uno o más elementos de tipo *UiElement* no editable o a un elemento de tipo *GroupingElement* conteniendo cualquier combinación de las tres posibilidades anteriores.
- **Input Group Members:** pueden ser refinados en un elemento del tipo *UiInputStructure*.

La representación en interfaz de usuario abstracta de algunos elementos de la vista estática del modelo *WAFCA* puede haber sido definida en el diseño de interfaz de usuario de un caso de uso. Para estos elementos, no es necesario definir a que elementos de interfaz de usuario abstracta se va a realizar una traza.

Aplicación al caso de estudio. En la aplicación al caso de estudio de Sistema de Administración de Bibliotecas Online, muchos de los elementos de interfaz de usuario abstracta ya han sido definidos en los casos de uso correspondientes. Los elementos de tipo input de acceso a funcionalidades *search words* y *search advanced options* de acceso a funcionalidades se corresponden, respectivamente a los elementos *words* y *advanced search form* de interfaz de usuario abstracta previamente definida para el caso de uso *search catalog in the OPAC*. El elemento de contenido *item* de acceso a funcionalidades se corresponde al elemento de tipo record *item* de la interfaz de usuario abstracta previamente definida para el caso de uso *show record in the OPAC*. El elemento de contenido de acceso a funcionalidades *search results* se corresponde al elemento *items list* de interfaz de usuario abstracta previamente definida para el caso de uso *search catalog in the OPAC*. Los elementos de contenido de tipo Input y Read only Content definidos en el modelo estático de acceso a funcionalidades son refinados a continuación:

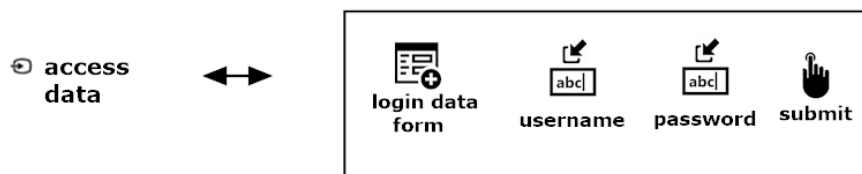


Figura 12.24: Refinamiento de elementos de tipo Input de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

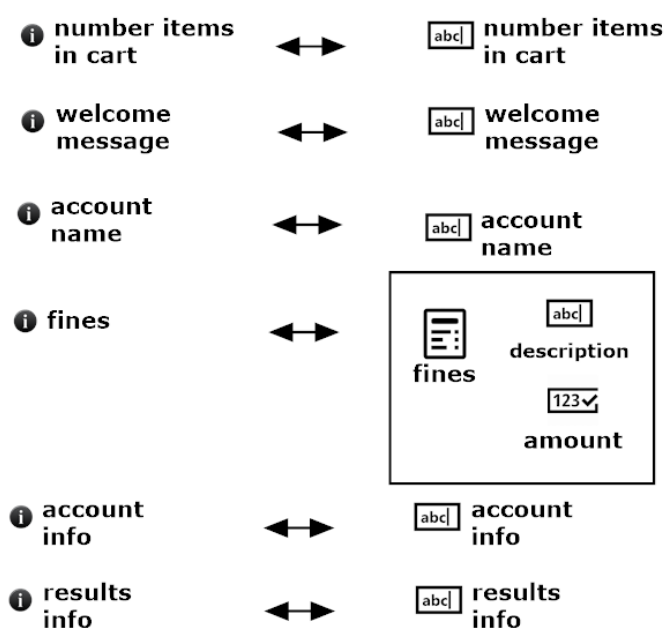


Figura 12.25: Refinamiento de elementos de tipo Read only Content de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

A continuación, se presenta el refinamiento a elementos de interfaz de usuario abstracta de elementos del tipo Content with interaction definidos en el modelo estático de acceso a funcionalidades para el subsistema OPAC del caso de estudio.

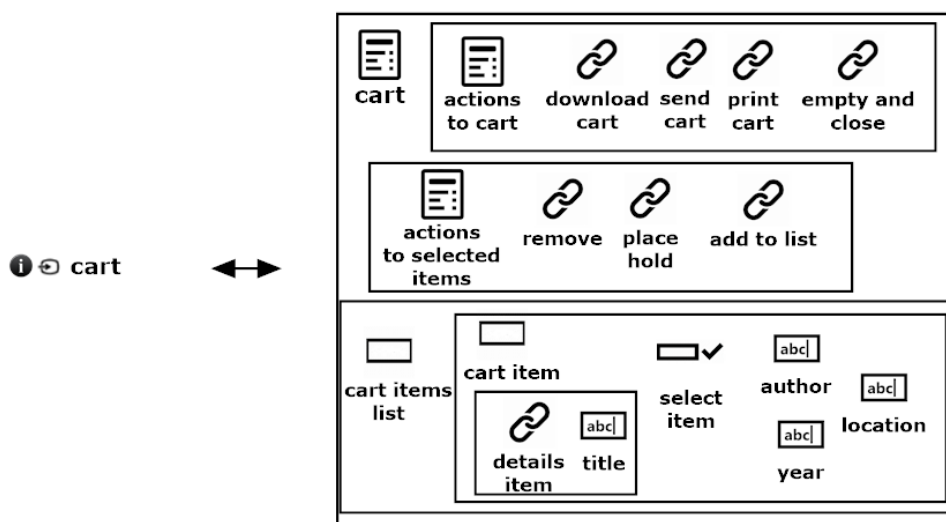


Figura 12.26: Refinamiento del elemento de tipo Content with interaction Cart de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

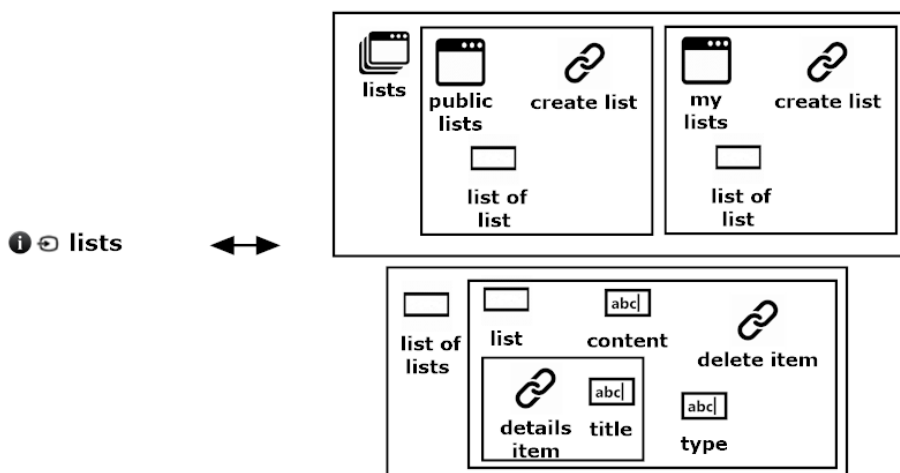


Figura 12.27: Refinamiento del elemento de tipo Content with interaction Lists de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

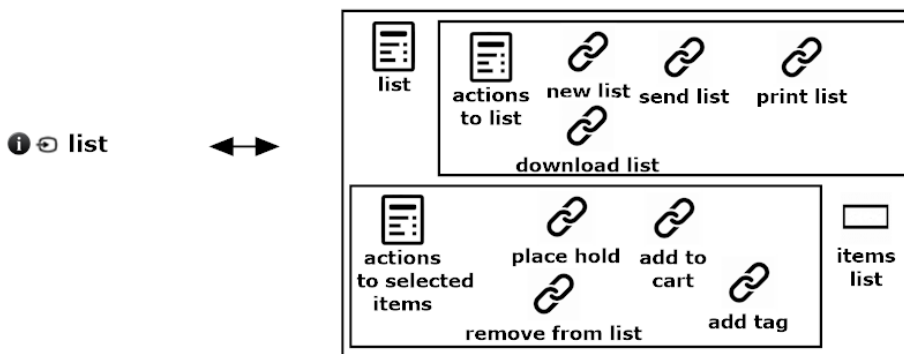


Figura 12.28: Refinamiento del elemento de tipo Content with interaction List de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

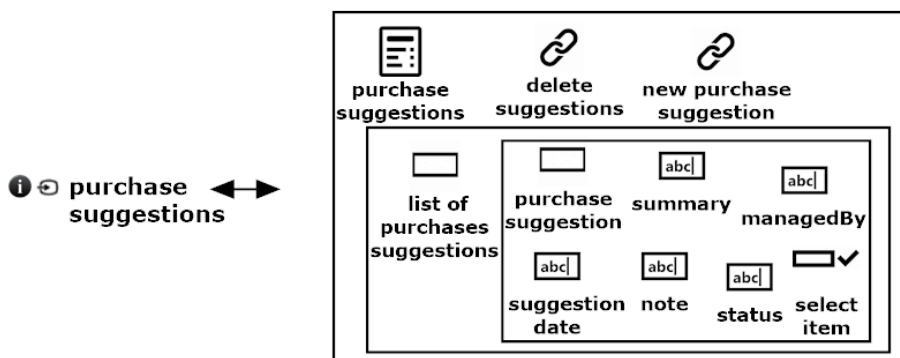


Figura 12.29: Refinamiento del elemento de tipo Content with interaction Purchase Suggestions de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

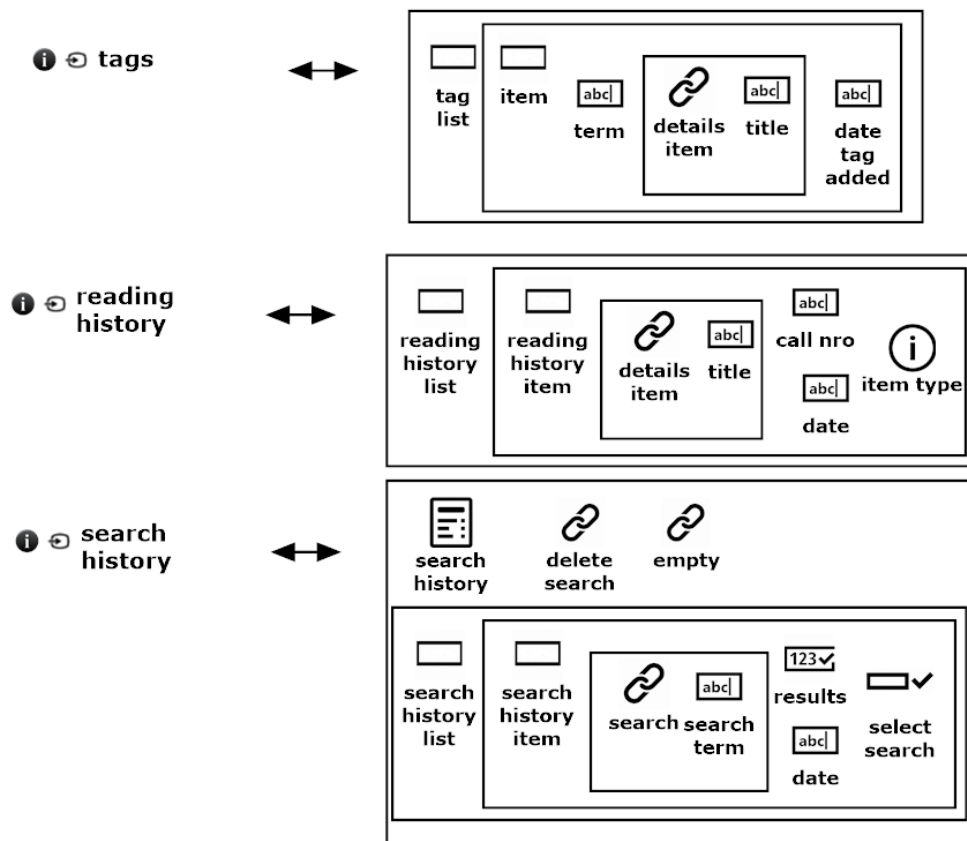


Figura 12.30: Refinamiento de los elementos de tipo Content with interaction Tags, Reading history y Search History de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

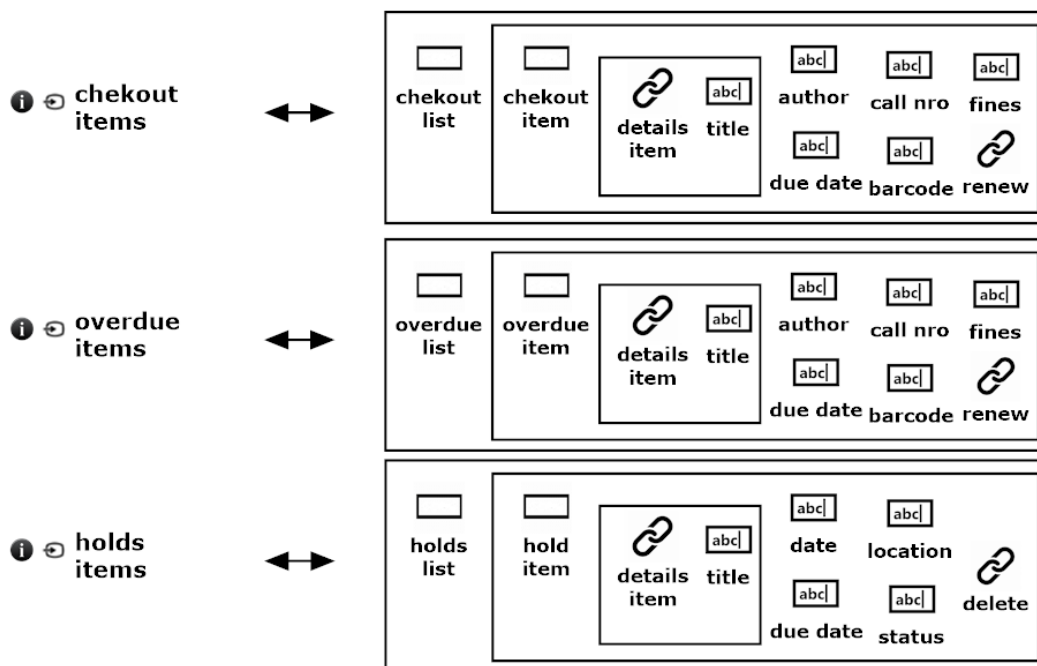


Figura 12.31: Refinamiento de los elementos de tipo Content with interaction checkout items, overdue items y holds items de acceso a funcionalidades a elementos de interfaz de usuario abstracta para el subsistema OPAC del caso de estudio.

Refinamiento de eventos y condiciones

Un *evento atómico* consiste de su nombre, su origen (por ejemplo una instancia de elemento de interfaz de usuario abstracta) y su información. Asumimos que en algún momento dado de la ejecución de la aplicación web, existe un flujo de eventos atómicos que han ocurrido. Además, por cada evento atómico en el flujo de eventos, hay una marca de tiempo correspondiente a su ocurrencia.

Utilizando la notación para definición de eventos sobre elementos de interfaz gráfica del capítulo 2.2.2, se consideran tres tipos de trazas a partir de eventos de la vista dinámica del modelo WAFCA:

- Trazas desde un elemento *Interaction* en un elemento *Access* a un evento atómico (quizás sobre un elemento de interfaz de usuario abstracta)
- Trazas desde un elemento *Event* a un evento atómico (quizás sobre un elemento de interfaz de usuario abstracta)
- Trazas desde un elemento *Condition* a una condición más específica (quizás refiriéndose un elemento de interfaz de usuario abstracta)

Aplicación al caso de estudio. Para el caso de estudio de Sistema de Administración de Bibliotecas Online, en particular para el subsistema OPAC y el modelo dinámico de acceso a funcionalidades definido anteriormente tenemos algunas de las siguientes trazas:

event “choose item” sobre el elemento Content with interaction “search results”

traza a

Press on «anchor» *details item*, para el elemento de interfaz de usuario abstracta *details item* dentro del elemento *item* a su vez dentro del elemento *list items*.

El resto de los elementos de tipo Event de la parte dinámico de acceso a funcionalidades desarrollados

aquí para el caso de estudio del tipo “choose element”, donde element puede ser un ítem o una lista tienen una traza análoga a la traza anterior.

event “back to results” sobre el elemento Content with interaction “item”

traza a

Press on «anchor» back to results, para el elemento de interfaz de usuario abstracta *back to results* dentro del elemento *show record in the OPAC*.

event “select item” sobre el elemento Content with interaction “search results”

traza a

Select on «single choice» check item, para el elemento de interfaz de usuario abstracta *check item* dentro del elemento *item* a su vez dentro del elemento *list items*.

El resto de los elementos de tipo Event de la parte dinámica de acceso a funcionalidades desarrollados

aquí para el caso de estudio del tipo “select item”, tienen una traza análoga a la traza anterior.

event “unselect item” sobre el elemento Content with interaction “search results”

traza a

Unselect on «single choice» check item, para el elemento de interfaz de usuario abstracta *check item* dentro del elemento *item* a su vez dentro del elemento *list items*.

El resto de los elementos de tipo Event de la parte dinámica de acceso a funcionalidades desarrollados

aquí para el caso de estudio del tipo “unselect item”, tienen una traza análoga a la traza anterior.

Condition “no items selected”

traza a

all items in actual list have their «single choice» select value equal to false.

Capítulo 13

Implementación de interfaz de usuario de aplicaciones web responsivas

13.1. Background

Debido al notable crecimiento del acceso a Internet a través de dispositivos móviles, tabletas y televisores, el diseño de aplicaciones web que se adaptan a diferentes tipos de dispositivos se transformó en una necesidad. En los últimos años, el diseño web responsivo (RWD, ver [Peterson, 2014]) emergió como una solución eficiente para este problema, ya que con este tipo de diseño se provee a los usuarios de un sitio web del mismo contenido y una experiencia de usuario similar en distintos dispositivos; esto redujo costos y tiempos en el mercado. Con el tiempo fueron surgiendo una gran cantidad de frameworks responsivos (por ejemplo Bootstrap, Foundation, Semantic UI), los cuales permiten hacer desarrollo responsivo de aplicaciones web.

Dada la gran variedad y diferencia de frameworks responsivos, hemos contemplado como una buena idea considerar un nivel de abstracción por encima de éstos, más específicamente, que sería deseable tener una notación de diseño de interfaz de usuario abstracta que 1) se abstraiga de widgets, layout y estilos de los elementos particulares de los frameworks responsivos, 2) sea independiente de modalidad, 3) sea independiente de tecnología de implementación. Las ventajas de respetar estos requisitos son: *a*) la posibilidad de construir varias implementaciones (de acuerdo a una tecnología específica, una modalidad o definiciones de layouts o estilos) a partir de un único modelo de diseño de interfaz de usuario; *b*) la posibilidad de tener un modelo de interfaz de usuario más concreto que los modelos de requisitos y más abstracto que la interfaz de usuario final; *c*) tener generalizaciones de los widgets de los frameworks responsivos.

Usando esta notación de diseño de interfaz de usuario se puede construir un modelo de interfaz de usuario abstracta de la aplicación, además, se puede considerar una traducción del modelo de interfaz de usuario abstracta a un framework responsivo particular.

En cuantos a trabajos que prescriban como construir una interfaz de usuario final a través de la consideración de un framework responsivo seleccionado y un modelo de interfaz de usuario abstracta, solo hemos hallado el trabajo de [Seixas, 2019], en donde se utiliza el enfoque llamado XIS-WEB (que incluye notación de modelado y una herramienta de soporte) y MDD para generar interfaz de usuario responsiva. Los problemas del trabajo de [Seixas, 2019] son: la notación de modelado de interfaz de usuario abstracta solo incluye el concepto de widgets (y no otros tipos de elementos menos concretos); no tiene en cuenta en su proceso la definición de la interfaz de usuario abstracta para el acceso a funcionalidades contenido; y no presenta un posible mapeo de elementos abstractos a widgets de más de un framework responsivo de destino.

La definición de un proceso de desarrollo que considere requisitos, modelo de interfaz de usuario abstracta y un framework responsivo para la construcción de la interfaz de usuario final de una aplicación web debe tener en cuenta la definición de la interfaz de usuario abstracta de las unidades de funcionalidad (casos de uso, user stories, tareas, comandos, servicios, procesos de negocios); pero esto no suele

ser suficiente para el diseño de la interfaz de usuario abstracta de una aplicación, también es importante tener una visión global de la interfaz de usuario para acceso a funcionalidades. Por lo tanto, se debe diseñar la interfaz de usuario final de una aplicación teniendo en cuenta la interfaz de usuario abstracta de las descripciones de las unidades de funcionalidad y la interfaz de usuario abstracta para el acceso a funcionalidades.

Los objetivos de este capítulo son: *a)* hacer una selección de la gran cantidad de frameworks responsivos existentes de acuerdo a algún criterio; *b)* realizar una comparación de la selección de frameworks responsivos y que esta comparación ayude al desarrollador a seleccionar un framework responsivo en particular para una aplicación específica; *c)* definir un proceso de desarrollo que contemple un meta-modelo de interfaz de usuario abstracta para aplicaciones web, considere diseño de interfaz de usuario abstracta, y la transición a implementación de la interfaz de usuario abstracta (considerando al menos widgets y layout). Los requisitos de este proceso son:

- Se debe comenzar con cualquier tipo de artefactos de fácil mapeo al modelo de interfaz de usuario abstracta definido en este trabajo. Pueden ser artefactos producidos por procesos de otras metodologías para aplicaciones web (por ejemplo, comenzar con modelos de navegación y contemplar transición de modelo de navegación a modelo de interfaz de usuario abstracta).
- Es necesario definir un mapeo de elementos de interfaz de usuario abstracto a widgets de frameworks responsivos o elementos de HTML5.
- Hay que realizar la construcción de modelos de interfaz de usuario para acceso a funcionalidades y unidades de funcionalidad usando el meta-modelo para interfaz de usuario abstracta.

Estos objetivos permiten alcanzar la meta número 7 de la sección 1.4 del capítulo de Introducción.

En este capítulo hemos hecho una selección entre diferentes frameworks responsivos en base a distintos criterios de popularidad, realizamos una comparación entre los mismos, confeccionamos una tabla de mapeo de elementos de interfaz de usuario abstractos (del meta-modelo definido en 2.2.2) a elementos de los frameworks responsivos seleccionados y, finalmente, desarrollamos un proceso de desarrollo de interfaz de usuario final usando un framework responsivo que utiliza la herramienta Pingendo para producir código fuente.

13.2. Trabajo relacionado

Cómo principal trabajo relacionado, hemos hallado el proceso presentado en [Seixas, 2019]. En este trabajo se utiliza MDD para generar código fuente en HTML5, CSS3 y Bootstrap; a partir de modelos abstractos. El proceso se basa en el enfoque XIS-WEB, el cual incluye dos vistas de modelado abstracto (espacio de navegación y espacio de interacción) y una herramienta en Acceleo para generar el código fuente. En cuanto a elementos de modelo de interfaz de usuario abstracta solo incluye el concepto de widgets (en este trabajo se reserva ese término para elementos concretos) y no se hace una distinción en distintos tipos de algunos elementos, como por ejemplo distintos tipos de elementos de medios o distintos tipos de elementos de input. Además, no se ocupa de manera específica de la interfaz de usuario para acceso a funcionalidades y contenido, la cual representa una visión global de la interfaz de usuario una aplicación y que consideramos importante que sea tenida en cuenta.

En [Cirilo, 2012] se presenta un enfoque para adaptar aplicaciones web ricas a grupos de dispositivos. Se construye un modelo de interfaz de usuario, llamado Rich Interface Domain, para cada grupo de dispositivos. Se utilizan adaptadores de contenido para refinar las interfaces de usuario en tiempo de ejecución de acuerdo con las características específicas del dispositivo de acceso identificado dinámicamente; los adaptadores de contenido llevan a cabo la parte dinámica de la adaptación de la interfaz. Varios de los problemas de adaptación de contenido resueltos por los adaptadores de contenido han sido resueltos mediante el uso de frameworks responsivos.

En [Manca, 2013] se presenta un enfoque que soporta adaptación de interfaz de usuario multimodal (combinando modalidad gráfica y vocal). Para dicho fin presenta una arquitectura en el servidor que genera HTML con CSS anotado con clases particulares. Parte de modelo abstracto de interfaz de usuario de MARIA y genera versiones concretas en distintos lenguajes por cada modalidad. Este enfoque es propicio para aplicaciones multimodales y para aplicaciones con diseño adaptivo, de lo cual nosotros no nos preocupamos en este trabajo.

En [Ghiani, 2014] se presenta un enfoque que partiendo de modelo abstracto de interfaz de usuario de MARIA, genera versión de interfaz concreta para una modalidad y la adapta de acuerdo de acuerdo a dispositivo mediante una arquitectura en el servidor. El enfoque soporta cuatro tipos de adaptaciones: Graphical-to-graphical adaptation, Graphical-to-multimodal adaptation, Multimodal-to-graphical adaptation y Multimodal-to-multimodal adaptation. El trabajo y la contribución del paper se centra en soporte multimodal y se ilustra como realizar ésto para modalidad vocal. Este enfoque es propicio para aplicaciones multimodales, de lo cual no nos preocupamos en este trabajo.

Nuestro trabajo se basa en diseño web responsivo que es diferente del diseño web adaptable (el servidor detecta el dispositivo, por ejemplo, un teléfono móvil, tableta o computadora de escritorio y el navegador cargará la versión del sitio que está optimizada para ese dispositivo). El diseño web adaptable se centra en el servidor, ya que todo el trabajo lo realizan los servidores antes que la aplicación llegue al cliente. Las aplicaciones responsivas tienen las siguientes propiedades: 1) no requieren una arquitectura adicional en el lado del servidor, lo que minimiza los costos en el lado del servidor; 2) son aplicaciones para Internet (las arquitecturas complejas en el lado del servidor para hacer adaptaciones son adecuadas para intranets); 3) incluyen todo el contenido de la aplicación para todos los usuarios y todos los dispositivos; el usuario ve la aplicación completa; 4) cuando se necesita una modificación, se construye una nueva versión de la aplicación responsiva (en los otros enfoques será necesario generar nuevamente el código para cada dispositivo).

En la web hemos encontrado varios documentos que comparan frameworks responsivos de acuerdo a diferentes aspectos. Hemos destacado a [Gerchev, 2018], que selecciona los cinco frameworks responsivos más populares en base a un conjunto de criterios y características de cada framework; como tamaño del framework, popularidad en la plataforma de desarrollo de software GitHub, documentación, soporte para navegadores, entre otros. En [Saravanakumar.org, 2019] se presenta un filtro de búsqueda que permite comparar frameworks responsivos de acuerdo a tipo de diseño, escritura, tipo de proyecto, entre otros aspectos.

13.3. Frameworks responsivos

13.3.1. Conceptos básicos

El *diseño web responsivo* es un enfoque de diseño de páginas web destinado a la elaboración de sitios que proporcionan una visualización óptima y una experiencia de interacción adecuada (de fácil lectura y navegación con un mínimo posible de cambios de tamaño, orientación y desplazamientos) a través de una amplia gama de dispositivos (desde monitores de computadoras de escritorio a teléfonos móviles).

Un *framework responsivo* es un conjunto de librerías que permiten a los desarrolladores realizar sitios utilizando diseño web responsivo. Dichas librerías son incluidas a través de archivos escritos utilizando *JavaScript* y *CSS* insertados en archivos *HTML*.

El desarrollo de varias versiones de una aplicación para diferentes dispositivos no suele ser una buena opción para algunos tipos de aplicaciones, ya que es costoso en cuestiones de tiempo y de dinero. Por esta razón, durante los últimos años, surgieron gran cantidad de frameworks para diseño web responsivo. Estos frameworks responsivos, en general, proveen soporte para todo tipo de dispositivos actuales (móviles, tabletas y computadoras de escritorio) a través de las siguientes características:

1. **Flexibilidad en imágenes.** Poseen atributos especiales en etiquetas HTML de tipo `image` para que la visualización de una imagen se ajuste al ancho del dispositivo. De esta manera, imágenes de tamaño superior al ancho de resolución de pantalla son adaptadas a esa resolución.
2. **Sistema fluido de grillas.** Proveen un sistema de grillas responsivas que escala apropiadamente a una cantidad de filas y columnas de acuerdo al tamaño de resolución de pantalla. Para esto, incluyen clases predefinidas con opciones para facilitar el layout de la aplicación. Ayudan a organizar el contenido de una aplicación de una manera más intuitiva.
3. **Inclusión de un conjunto rico de widgets.** Incluyen un conjunto de widgets predefinidos (utilizando HTML y JavaScript) para cuestiones de interfaz de usuario utilizadas habitualmente. Esto abarca, widgets para estructuras de contenido, widgets para estructuras de acceso basado en enla-

ces, widgets para estructuras de acceso no sólo basados en enlaces -es decir, que contienen, además de los enlaces, controles para elementos de contenido-, conjuntos de botones y conjunto de íconos.

4. **Uso de atributos CSS3 para controlar distintas definiciones de estilos para mismos elementos.** Estos atributos permiten definir distintas reglas de estilo de acuerdo a la orientación, el alto y el ancho del dispositivo de acceso.

13.3.2. Selección de frameworks responsivos

En base a un estudio de popularidad de frameworks responsivos, que se basa en número de resultados de búsqueda en los buscadores web, cantidad de bibliotecas externas definidas utilizando el framework responsivo y valoración en la plataforma de desarrollo colaborativo GitHub, en este trabajo hemos seleccionado cuatro frameworks responsivos con los que se podrá realizar el mapeo de elementos de interfaz de usuario abstracta. Estos son (en orden de popularidad):

1. **Bootstrap.** Surgió en el año 2011 y se convirtió en el framework responsivo más popular y elegido por los desarrolladores, así lo demuestran su gran cantidad de menciones en Internet y las librerías externas que de este framework se han hecho. Sus principales características son su facilidad de uso (sus elementos tienen una sintaxis clara y fácil de comprender), su buena responsividad (a través de sistemas de grillas), la gran cantidad de elementos predefinidos que incluye (widgets y componentes) y el excelente soporte de documentación y herramientas (bootply.com, bootsnipp.com, entre otras) que posee.
2. **Foundation.** También surgió en el año 2011 es considerado, después de *Bootstrap*, el segundo framework más popular. Sus características más valoradas son su riqueza de elementos predefinidos (posee gran cantidad de widgets y componentes únicos y además, a diferencia de *Bootstrap*, incluye un conjunto de íconos propios), su soporte online para personalización (incluye una herramienta para personalizar componentes y widgets y generar una versión minimizada y personalizada) y la buena cantidad de recursos y documentación que posee.
3. **Semantic UI.** Surgió en el año 2013 y es uno de los frameworks que más ha crecido a nivel de popularidad en los últimos años. Sus aspectos a destacar son el uso de principios de lenguaje natural que hacen un código más legible y fácil de entender, su conjunto de componentes y widgets únicos (incluyen componentes para estadísticas, figuras, publicidad, entre otros; que no están presentes en otros frameworks) y su buena documentación incluida (posee un sitio web separado dedicado a documentación). Al hacer uso de herramientas como lenguaje natural para su construcción posee algunos componentes particulares, como por ejemplo el elemento *Dimmer*, que es un contenedor que trata de evitar distracción del usuario para enfocar la atención en un contenido particular.
4. **Pure.** Surgió en el año 2013 por medio de *Yahoo*, y al igual que *Semantic UI*, ha tenido un gran crecimiento en los últimos años. Su principal característica es que es un framework extremadamente liviano (sus archivos tienen un tamaño de alrededor de 3.8 Kb, a diferencia de los frameworks que sobrepasan los 200 Kb). Además, algunas de sus otras características son su diseño modular que permite utilizar sus componentes separadamente de acuerdo a las necesidades y que es escrito puramente en CSS (de ahí proviene su nombre). Provee un buen soporte de documentación y una herramienta para personalizar sus componentes. A diferencia de los otros frameworks no posee un gran número de widgets y componentes, solo un conjunto minimal con los asuntos considerados fundamentales. De todas maneras, existen aportes externos que incluyen widgets y componentes para este framework.

Además de estos frameworks que hemos seleccionado, existe una gran cantidad de otros frameworks con buena popularidad y características, pero por motivos de delimitar las posibles de mapeos en este trabajo no han sido incluidos. Algunos de ellos son: *UIKit*, *Material UI*, *Materialize*, *Skeleton*, *Miligram* y *HTML Kickstart*. Para aplicaciones para dispositivos móviles denominadas híbridas (aplicaciones web que hacen uso de funcionalidades nativas del dispositivo -por ejemplo el uso de la cámara, el acelerómetro-) el uso de un framework responsivo no es suficiente, ya que los frameworks responsivos no permiten acceder a funcionalidades nativas. Para este fin existen otros frameworks tales como *Ionic*, *Intel XDK* y *Appcelerator Titanium*, entre otros.

13.4. Mapeo de elementos de interfaz de usuario abstracta a elementos de interfaz de usuario de frameworks responsivos

Una vez que se selecciona un framework responsivo para implementar una aplicación, es necesario elegir con qué elementos de interfaz gráfica de usuario de los frameworks responsivos se implementarán los elementos de interfaz de usuario definidos para la interfaz de usuario abstracta mediante el meta-modelo *WAAUID*. Para ayudar a realizar esta tarea, se construyó una tabla, dividida en dos partes (tabla 13.1 y tabla 13.2), que para cada elemento de interfaz de usuario abstracta estructural del meta-modelo *WAAUID* provee cuales elementos de los frameworks responsivos pueden ser considerados para la implementación del elemento abstracto. Los elementos concretos para implementar los elementos abstractos son obtenidos de la siguiente forma: si el framework responsivo tiene elementos asociados al elemento de *WAAUID*, entonces se debe elegir entre alguno de ellos, de acuerdo a cual es más conveniente para las necesidades de la aplicación. Si el framework responsivo no posee elementos asociados al elemento de *WAAUID*, el diseñador web debe elegir una implementación utilizando *HTML* o elegir elementos de alguna librería externa.

Ui Category	WAFCA element	Bootstrap	Foundation	Semantic UI	Pure	
Output	Message	Tooltips, Popovers, Alerts (primary, secondary, info, light, dark), Badges	Tooltips	Message, Label	-	
	Notification	Error Warning Success	Alerts (success, warning, danger)	Message (warning, success, error)	-	
		Type = status	Progress	Progress Bars	Loader, Statistic, Progress	-
	Dialog	Modal	Reveal Modal	Modal	-	
Data Input	UInputStructure	InputGroup	Forms	Input	Forms	
	Form	Forms	Forms	Form	Forms	
Content	Content structures -CS- for images	Carousel	Orbit slider, Block Grid	-	Grids + Pure-img	
	CS (filterable = true)	-	-	-	-	
	Grid	Table	Tables	Tables	Table	Tables
		Other	Grids, Media objects (media list)	Grid, Block Grid	Grid	Grids
	List	List group, Media object (media list)	Inline lists	List, Item	-	
	Alt	Collapse	Tabs, Accordion	Accordion, Tab	-	
	Record	Media object (record)	Thumbnails	Feed	-	
	Tree	-	-	-	-	

Figura 13.1: Tabla de mapeos de elementos de *WAAUID* a elementos de Frameworks responsivos. Parte 1.

	Elemento WAAUID		Bootstrap	Foundation	Semantic UI	Pure
Access	<i>Links based (2 or more anchors)</i>		Pagination	Pagination	List	-
	Menu	<i>with nesting</i>	Button Group, Navs	Off canvas	Menu, Dropdown	Vertical Menu, Horizontal Menu, Dropdowns
		<i>no nesting</i>	Buttons, Dropdowns	Icon Bar, Side Nav, Sub Nav, Button Groups, Split Buttons, DropDown buttons, Inline lists	Button	Buttons
		<i>bookmark link type</i>	-	-	-	-
	<i>Breadcumb</i>		Breadcumb	Breadcrumbs	Breadcumb	-
	<i>NavAltBlocks</i>		Collapse	Tabs, Accordion	Accordion, Tab	-
	Nav Grid	<i>NavList</i>	List Group (con âncoras)	Clearing lightbox	-	-
		<i>bookmark links</i>	-	Joyride	-	-
	<i>NavigationBar</i>		NavBar	Top bar	Sidebar	-
	Layout / Containers	<i>Grouping element -GE-</i>		Jumbotron, Card, Grids	Reveal modal, Panels, Grid	Grid, Container, Sticky, Rail, Card, Dimmer
<i>GE de GE</i>		Grids	Grid	Grid, Container	Grids	
<i>GE of images</i>		-	Block Grids	-	-	
<i>GE of guided tours</i>		Carousel	Orbit Slider	-	-	

Figura 13.2: Tabla de mapeos de elementos de WAAUID a elementos de Frameworks responsivos. Parte 2.

Además, para los elementos de acceso a funcionalidades de tipo *Grouping*, se brinda los siguientes posibles mapeos:

- *root grouping*: pueden ser mapeados de la misma manera que elementos de tipo Grouping Element de WAAUID.
- *alternative grouping*: pueden ser mapeados de la misma manera que elementos de tipo NavAltBlock de WAAUID o bien de la misma manera que elementos de tipo Grouping Element de WAAUID (elementos que se visualizan alternativamente de acuerdo a interacción con el usuario).
- *all grouping*: pueden ser mapeados de la misma manera que elementos de tipo Grouping Element de WAAUID, o bien de la misma manera que elementos de tipo Menu de WAAUID, o bien de la misma manera que elementos de tipo Breadcumb de WAAUID, o bien de la misma manera que elementos de tipo NavigationBar de WAAUID.

13.5. Proceso de desarrollo de interfaz de usuario final usando un framework responsivo

Una vez que se ha completado el diseño de la interfaz de usuario abstracta de los modelos de requisitos y diseño de la aplicación para las unidades de funcionalidad (casos de uso en este trabajo) y del modelo de

acceso a funcionalidades y contenido de la aplicación a desarrollar, están dadas las condiciones necesarias para pensar en el desarrollo de la interfaz de usuario final de la aplicación.

En la actualidad, existen dos enfoques para construir aplicaciones web: aplicaciones web tradicionales que realizan la mayor parte de la lógica de la aplicación en el servidor, y aplicaciones de una sola página que realizan la mayor parte de la lógica de interfaz de usuario en el navegador web, comunicándose con el servidor web primariamente usando APIs web. También es posible un enfoque híbrido.

Con la aparición de *HTML5* (quinta revisión importante del lenguaje básico de la World Wide Web, HTML), la aparición de ECMAScript 2019 (especificación de lenguaje de programación, JavaScript está implementado bajo este estándar), el auge de los servicios web, el almacenamiento en nubes de datos y el surgimiento de numerosos frameworks en JavaScript para construir aplicaciones web; las aplicaciones web de una sola página o de enfoque híbrido han ido creciendo mucho en los últimos años en proporción a las aplicaciones web tradicionales. Por este motivo, en este trabajo nos centraremos, en particular, en la construcción de la interfaz de usuario final de una aplicación web y no nos ocuparemos de cuestiones concernientes al servidor, ya que cada vez más esto se está encapsulando en servicios que pueden ser consumidos desde los clientes.

13.5.1. Implementando interfaz de usuario final utilizando una herramienta para frameworks responsivos

Una vez que se han tomado las decisiones acerca del mapeo de los elementos de interfaz de usuario abstracta a los elementos de *HTML* y del framework responsivo seleccionado, el siguiente paso es realizar la implementación de la interfaz de usuario final de la aplicación. Para este fin, hemos dividido el trabajo en dos tareas: en primer lugar se crean porciones de interfaz de usuario para unidades de funcionalidad utilizando lenguaje *HTML* y widgets del framework responsivo seleccionado, luego y en base al modelo dinámico de acceso a funcionalidades y contenido, se especifica como interconectar las porciones de interfaz de usuario a través de enlaces y eventos de JavaScript o jQuery.

Creando estructuras y partes de interfaz de usuario

En esta sección se presenta un listado de algunas tareas que deberían ser llevadas a cabo para implementar la interfaz gráfica de usuario final de una aplicación web responsiva. A manera ilustrativa para llevar a cabo las tareas de implementación en este trabajo se utiliza *Bootstrap* como framework responsivo y *Pingendo* (que es una herramienta gratis) como herramienta para construir la interfaz gráfica de usuario final.

Las tareas propuestas que deben ser realizadas son:

1. **Crear una página en blanco.** En *Pingendo* se debe seleccionar la opción *Default empty page*.

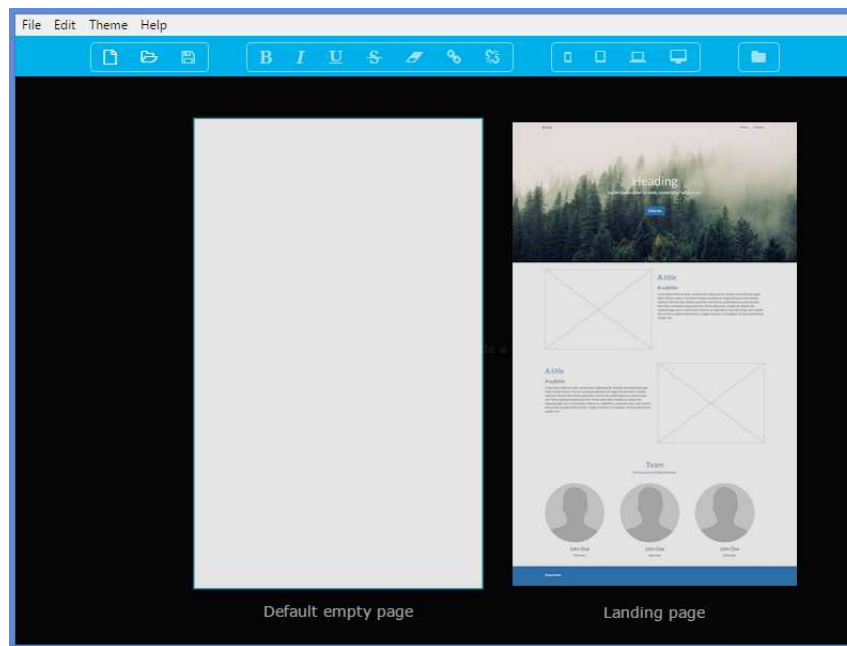


Figura 13.3: Creando una página en blanco en *Pingendo*.

2. Teniendo en cuenta los requisitos de interfaz de usuario provistos por el cliente y la definición del elemento contenedor de más alto nivel del modelo estático de acceso a funcionalidades, **definir el layout de la aplicación web responsiva** en términos de contenedores, filas y columnas. Para este fin, *Pingendo* incluye estructuras de layouts predefinidas (en la pestaña *Sections* y el apartado *Structures* como se puede ver en la figura 13.4). Si el layout deseado no se encuentra dentro de las estructuras predefinidas, se deben utilizar los elementos de la sección *Layout* dentro de la pestaña *Components* de la herramienta. En general, para ésta tarea es frecuente definir un elemento *Container* por cada barra de navegación o contenido siempre visible en la aplicación y un elemento de tipo *Container* para el contenido de la aplicación. Para el contenedor de contenido de la aplicación se debe considerar dentro del mismo una disposición de filas y columnas acorde al diseño de layout deseado (por ejemplo dos columnas, cada una de las cuales incluye una cierta cantidad de filas).

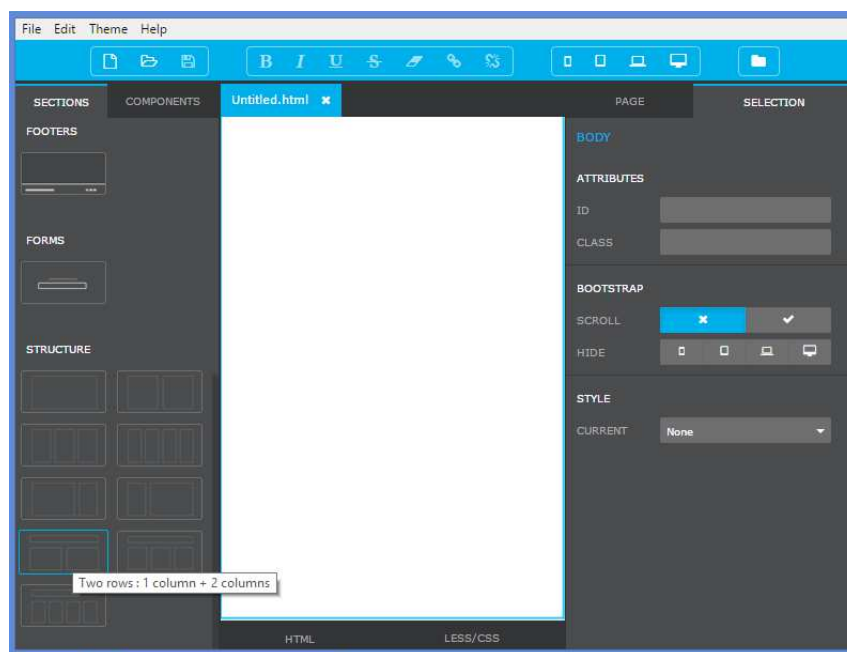


Figura 13.4: Estructuras de layout predefinidas en *Pingendo*.

Para la aplicación web del subsistema OPAC del caso de estudio de Biblioteca Online, en base a la definición del modelo estático de acceso funcionalidades de la figura 12.1 se decidió implementar un layout de 4 filas y una columna (4,1), donde la primer fila es una barra de navegación de la aplicación (implementada con un Container), la segunda una barra de búsqueda, la tercera una lista de utilidades y la cuarta fila es donde se visualiza el contenido de la aplicación de manera alternativa.

3. **Incluir los elementos para acceso a funcionalidades.** Teniendo en cuenta un par válido (fila, columna) del layout definido en el punto anterior, se deben incluir los elementos de interfaz gráfica de usuario y estructuras de acceso que se utilicen para acceder a funcionalidades de la aplicación. También puede ser necesario incluir elementos contenedores o de layout dentro de cada fila y columna, de acuerdo a la división de contenedores definida en el modelo estático de acceso a funcionalidades. Para la tarea de definición de elementos de acceso, en *Pingendo* se incluyen secciones llamadas *Navigation* y *Buttons* dentro de la pestaña *Components*, que contienen los elementos más frecuentemente utilizados para este fin, como lo son *Button*, *Button DropDown*, *NavBar* y *Breadcumb*, además de las tradicionales áncoras de HTML.

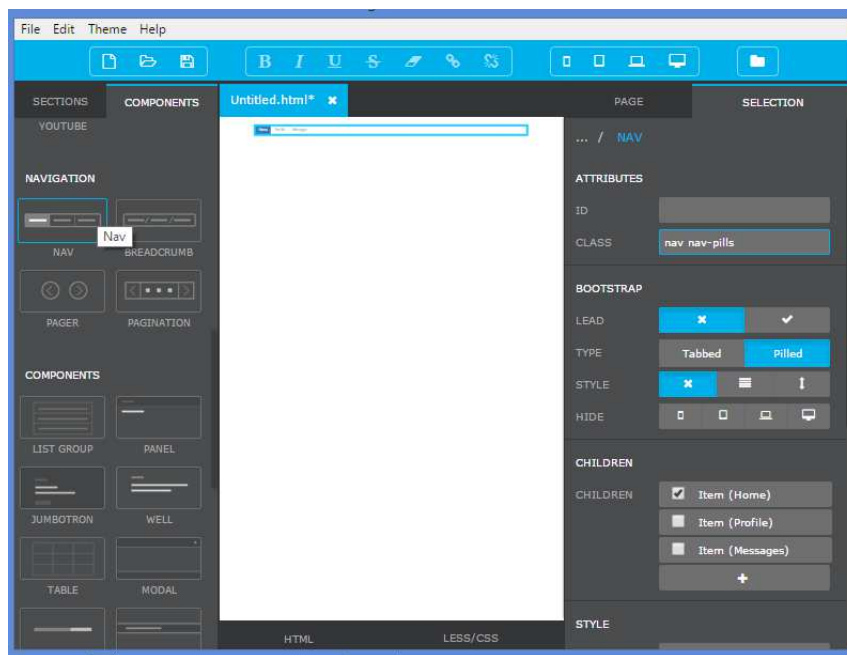


Figura 13.5: Incluyendo una barra de navegación en *Pingendo*.

4. **Para cada unidad de funcionalidad, crear una página en blanco.** Posteriormente, de la misma manera que el punto 2. definir el layout de esta unidad de funcionalidad, para luego (como en el punto 3.), incluir los elementos de acceso a funcionalidades de la unidad dentro del layout definido.
5. **Los elementos de interfaz de usuario abstracta para los cuales no exista una correspondencia directa en widgets del framework responsivo seleccionado, implementar estos elementos utilizando o bien librerías externas al framework responsivo o bien código *HTML5*.** Para realizar ésto, en *Pingendo* se debe utilizar la funcionalidad de edición de código fuente *HTML* que la herramienta posee, y desde allí agregar los fragmentos de códigos necesarios, ya sea para adjuntar librerías y sus widgets o para insertar tags de *HTML* sin uso de librerías externas.

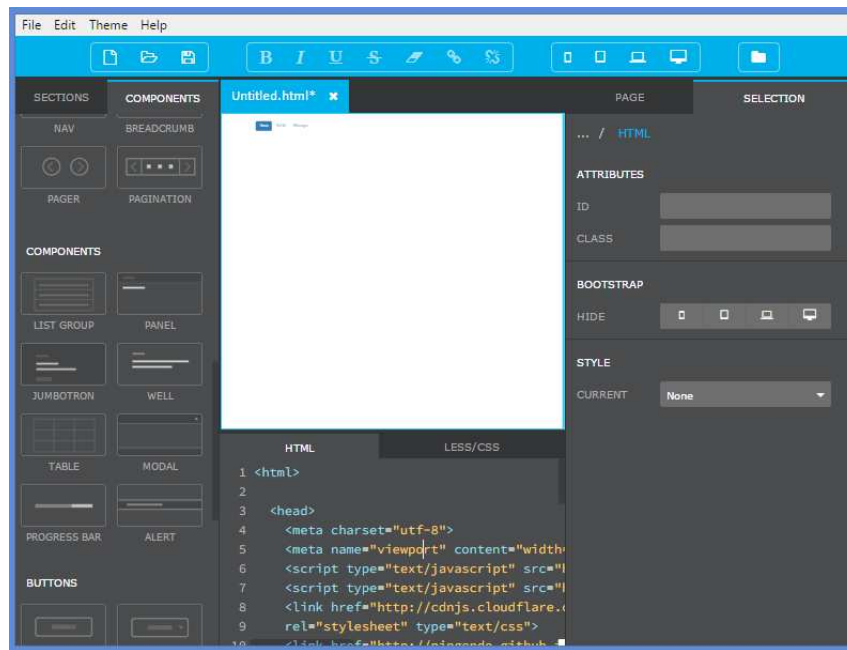


Figura 13.6: Edición de código *HTML* en *Pingendo*.

Interconectando la estructura y partes de interfaz de usuario

Como consecuencia de la actividad anterior, se cuenta con un conjunto de partes de interfaz de usuario gráfica de la aplicación, las cuales deben ser interconectadas de acuerdo a eventos de interfaz de usuario que han sido contemplados en el modelo dinámico de acceso a funcionalidades y contenido o en los eventos definidos en la descripción abstracta de la interfaz de usuario (también puede ayudar mirar las descripciones detalladas de los casos de uso a través de diagramas de actividades). Se deben definir las conexiones entre partes de interfaz definidas para acceso a funcionalidad y partes de interfaz para unidades de funcionalidad. A su vez, dentro de las unidades de funcionalidad también puede ser necesario definir conexiones de partes de interfaz gráfica de usuario.

Para los eventos más sofisticados que selección de áncoras tradicionales en este trabajo utilizamos jQuery y prescribimos realizar las siguientes acciones:

1. **Asociar los enlaces entre dos partes que representen áncoras tradicionales.** Si la aplicación web incluye características de aplicaciones web tradicionales de cliente-servidor, entonces el servidor enviará como resultado la interfaz de usuario correspondiente a la unidad de funcionalidad solicitada por medio de una solicitud realizada por el usuario a través de presionar un áncora. Para ésto se debe editar código *HTML*, crear una etiqueta de tipo A (puede ser un button también) de html y en el atributo href de la misma incluir la dirección *URL* correspondiente la unidad de funcionalidad. Ejemplo:

```
<div id="bloque-parte-1">
<a href="parte-2.html">
</div>
```

2. **Asociar los enlaces entre dos partes que representen áncoras de tipo comando.** Si la aplicación posee características RIA y realiza peticiones que no requieren recarga de página, estas deben ser implementadas capturando el evento de dar click sobre un áncora o un elemento de *HTML* utilizando jQuery y una petición utilizando *AJAX*, y actualizar la interfaz de usuario de acuerdo a la respuesta recibida a la petición. Ejemplo:

```
<div id="bloque-parte-1">
```



```
<div id="bloque-parte-2"></div>
<button id="boton-parte-2">Parte 2</button>
</div>
<script>
$( '#boton-parte-2' ).click(function(){
$.ajax
({
url: parte-2-servidor,
data: parametros,
type: 'post',
success: function(result)
{
$( '#bloque-parte-2' ).text(result);
}
});
});
</script>
```

3. **Asociar dos partes como consecuencia de un evento de interfaz de usuario.** Otras partes de la interfaz de usuario pueden ser mostradas, ocultadas o asociadas utilizando el resto de eventos de interfaz de usuario (hover, focus, mousedown, select, entre otros). Para esto, es necesario capturar el evento sobre una parte de interfaz de usuario, realizar una petición AJAX y como resultado de la misma realizar alguna acción en la interfaz de usuario que involucra a otra parte de interfaz de usuario de la aplicación (puede ser visualizar, ocultar, etcétera).

Ejemplo:

```
<div id="parte-1"><p id="elemento-parte1"></p>...</div>
<div id="parte-2"></div>
<script>
$( "#elemento-parte1" ).hover(function(){
$.ajax
({
url: parte2-servidor,
data: parametros,
type: 'post',
success: function(result)
{
if (result == "ok"){
$( '#parte-2' ).show();
}
}
});
});
</script>
```

Parte V

Conclusión

Capítulo 14

Conclusión

14.1. Resumen de contribuciones

A continuación se listan las principales contribuciones, que fueron publicadas en ámbitos académicos con referato, realizadas a lo largo de la realización de este trabajo:

- **Definición de una notación para modelar acceso a funcionalidades/contenido.**
En la sección 2.2.1 hemos mencionado por qué es importante modelar la interfaz de usuario para acceso a funcionalidades/contenido y hemos considerado seis beneficios, reflejados en la notación definida, que esto trae aparejado.
- **Definición de meta-modelo para interfaz de usuario abstracta que tiene en cuenta los últimos adelantos tecnológicos.**
En la sección 2.2.2 hablamos de la importancia de tener una notación para modelar interfaz de usuario de aplicaciones web de manera abstracta que sea independiente de modalidad, independiente de tecnología de implementación y contemple abstracciones de widgets y elementos utilizados en los últimos adelantos tecnológicos para interfaces de usuario, como lo son los frameworks responsivos y las aplicaciones RIA. El meta-modelo definido reúne las características mencionadas.
- **Extensión de notación para diagrama de casos de uso con variabilidades.**
En la sección 3.1.1 evidenciamos cuatro problemas existentes en las notaciones que modelan diagramas de casos de uso con variabilidades. Los problemas fueron resueltos en la extensión definida y tratan de restricciones de dependencia no consideradas anteriormente, situaciones de variabilidad no contempladas y situaciones de variabilidad que no fueron resueltas apropiadamente.
- **Definición de una notación de diagramas de actividad para capturar variabilidades.**
En la sección 3.2.1 hemos identificado seis problemas acerca de situaciones de variabilidad que no fueron consideradas y situaciones de variabilidad que no fueron resueltas apropiadamente en las notaciones existentes de diagramas de actividades con variabilidades. Estos problemas y situaciones fueron resueltos en la notación definida.
- **Definición de una notación NFR con variabilidades que considera los beneficios del campo de NFR.**
En la sección 3.3.1 hemos notado la importancia de modelar requisitos no funcionales en familia de aplicaciones y la importancia de modelar variabilidades en requisitos no funcionales. Además, hemos identificado algunos problemas existentes en las notaciones que modelan NFR con variabilidades. La notación de NFR con variabilidades definida resuelve los problemas hallados.
- **Definición de un enfoque MDD para transformación de notaciones de requisitos UML con variabilidades a un modelo de features, respetando un meta-modelo de features apropiado.**
En la sección 4.1 hemos hablado de la necesidad de contar, además de los modelos de dominio de UML, con un modelo de features para modelar una familia de aplicaciones. También hemos argumentado por qué optamos por generar automáticamente el modelo de la features de la familia

de aplicaciones a partir de los modelos de dominio y los beneficios que trae aparejado definir un proceso de desarrollo de este tipo, como aquí hemos hecho.

■ **Definición de un proceso de desarrollo de interfaz de usuario de aplicaciones web que considera modelo abstracto de interfaz de usuario y transición a implementación.**

En la sección 13.1 hemos mencionado como un asunto no considerado en la bibliografía la definición de un proceso de construcción de interfaz de usuario partiendo de modelo abstracto de interfaz de usuario, considerando la transición a un framework responsivo y brindando patrones de implementación comunes.

Como ya se dijo, las contribuciones previamente señaladas fueron incluidas en trabajos de investigación, realizados durante las distintas etapas de la realización de este trabajo. Todos estos trabajos fueron presentados en ámbitos académicos con referato.

A continuación se incluye el listado de publicaciones realizadas:

- *Requirements engineering of web application product lines*. Ver [Casalánguida, 2011].
- *Automatic Generation of Feature Models from UML Requirement Models*. Ver [Casalánguida, 2012].
- *A Method for Integrating Process Description and User Interface Use during Design of RIA Applications*. Ver [Casalánguida, 2013].
- *Development of a Design Model for Functionality and Content Access from Rich Internet Application Requirements*. Ver [Casalánguida, 2015] .
- *User interface design for responsive web applications*. Ver [Casalánguida, 2015b].

Además, otras contribuciones propias de este trabajo se citan a continuación:

■ **Definición de un proceso de desarrollo de familias de aplicaciones web que incluye etapa de modelado de dominio, etapa de configuración, etapa de modelado de aplicación y que satisface las metas de 1 a 6 de la sección de Introducción.**

En la sección 1.3 (Estado del arte, SPL en el dominio de aplicaciones web) del capítulo 1 de Introducción hemos encontrado algunos problemas que no han sido tenidos en cuenta en los trabajos de la bibliografía que intentan combinar SPL con aplicaciones web y que el proceso definido en este trabajo resuelve.

■ **Extensión de notación definida para modelar interfaz de usuario abstracta para contemplar variabilidades en interfaz de usuario.** En la sección 1.3 (Estado del arte, SPL en el dominio de aplicaciones web) del capítulo 1 de introducción se justificó la importancia de modelar variabilidades en interfaz de usuario.

■ **Extensión de enfoque MDD definido para transformación de notación interfaz de usuario abstracta con variabilidades a un modelo de features, respetando un meta-modelo de features apropiado.**

No hemos hallado en la bibliografía ningún trabajo que considere transformación de variabilidades en interfaz de usuario a modelo de features. Ya que para aplicaciones web la interfaz de usuario es un asunto relevante, es necesario que sus variabilidades estén incluidas en el modelo de features para poder ser validadas con el cliente; lo cual es lo que permite la transformación definida.

■ **Definición de un proceso de configuración de modelo de features y modelos de dominio.**

En la sección 8.1 hemos identificado dos tipos de enfoques para configurar modelos de dominio (en base a formalizaciones matemáticas y en base a transformación de modelos) y hemos hallado algunos problemas de esos enfoques para realizar configuración en base a los modelos de dominio que hemos definido en este trabajo. El proceso de configuración definido en este trabajo resuelve estos problemas.

14.2. Discusión

En este trabajo se discutió acerca de la producción de un proceso de desarrollo de familia de aplicaciones web desde cero. El ámbito de aplicabilidad del proceso de desarrollo está dado, principalmente, para organizaciones de mediana o gran escala (debido a que es más costoso que desarrollar a partir de productos variantes) que van a realizar una familia de aplicaciones web desde cero o para organizaciones de cualquier escala que van a hacer un desarrollo en base a ingeniería inversa de productos open source. Además, se delimitan los siguientes casos de aplicabilidad: sistemas de software de escritorio de gran tamaño que se desean migrar a la web como familia de aplicaciones, sistemas web de gran escala altamente configurables (sistemas de biblioteca, ERPs, CRMs, por ejemplo), sistemas web open source de gran escala elaborados de manera colaborativa (por ejemplo mediante el repositorio gitHub), extensión de familia de aplicaciones web existentes. La viabilidad del proceso aquí presentado ha sido validada a través de un caso de estudio principal, correspondiente a un Sistema de Administración de Bibliotecas Online altamente configurable (del estilo del sistema de bibliotecas Koha) y con casos de estudios menores para situaciones específicas durante la presentación de notaciones de modelado.

Para identificar potenciales contribuciones y obtener resultados de interés en cuanto al asunto de investigación, se realizó un estudio del estado del arte en familia de aplicaciones web y un estudio del estado del arte más específico de cada etapa de desarrollo. Los resultados y contribuciones más importantes obtenidos a través del estudio del estado de arte fueron publicados en cinco trabajos de conferencia (sección 14.1).

Durante la realización del trabajo se tuvieron en cuenta las etapas principales de la ingeniería de SPL para aplicaciones web, que fueron estructuradas en desarrollo de modelos de dominio de familias de aplicaciones web (parte II), configuración de modelos de dominio de aplicaciones web (parte III) y desarrollo de modelos de aplicación e implementación de interfaz de usuario responsiva (parte IV).

Con respecto a la notación para modelar acceso a funcionalidades/contenido, para identificar los elementos del meta-modelo se consideraron los casos de estudio de: una aplicación e-commerce, una aplicación de correo electrónico y el caso de estudio de este trabajo. Se verificó la importancia de modelar este asunto ya que permite tener una visión global de la aplicación web, brinda beneficios en la validación con el cliente (el cliente puede elaborar fragmentos y además puede tener una noción del comportamiento dinámico de la aplicación) y da la posibilidad de aplicar patrones de navegación a la navegación global de la aplicación; estos asuntos solo son considerados con el uso de esta notación.

Para el meta-modelo de interfaz de usuario abstracta se estudiaron distintos widgets RIA de la tecnología y los widgets y elementos de los frameworks responsivos más importantes de la actualidad, además de los elementos básicos de HTML5. Se creó una sintaxis concreta que permite identificar a los elementos de manera más intuitiva, lo cual la hace más fácil de entender e interpretar por los clientes. De los elementos de más bajo nivel de abstracción del meta-modelo, un total de 9 de 25 elementos se obtuvieron del estudio de widgets RIA y frameworks responsivos y no representan elementos incluidos en HTML; esto permite evidenciar la importancia del estudio realizado.

Sobre la notación extendida de casos de uso con variabilidades, se resolvieron cuatro problemas hallados en la bibliografía y se evitó hacer uso de elementos de diagramas de casos de uso para proveer información de variabilidad (esto no es considerado una buena práctica); además, una evaluación fue incluida en la sección 3.1.4.

En cuanto a la definición de notación de diagramas de actividad para capturar variabilidades se resolvieron seis problemas hallados en la bibliografía y se evitó hacer uso de elementos de diagramas de actividad para proveer información de variabilidad de flujo de control; además, una evaluación fue incluida en la sección 3.2.4.

Con respecto a la definición de notación NFR para capturar variabilidades se estudiaron cualidades importantes en el dominio de aplicaciones web, esto permitió identificar distintas soluciones en la tecnología para modelar descomposiciones de cualidades no funcionales las cuales deben ser tratadas como variabilidades. En los ejemplos y aplicación al caso de estudio realizados, de un total de 6 descomposiciones, 5 de ellas incluyeron variabilidades. Esto permite notar que existen consideraciones (obtenidas de derivaciones de requisitos no funcionales) con distintas soluciones posibles, en el ámbito de aplicaciones web, que necesitan ser modeladas con variabilidades.

Para la definición del enfoque MDD que transforma modelos de requisitos UML con variabilidades a modelo de features se consideraron cinco asuntos que no habían sido tenidos en cuenta en la bibliografía y se mejoró, con respecto a los enfoques hallados en la bibliografía, en cuanto a la consideración de ocho

asuntos. Se comprobó que el modelo de features obtenido como resultado de la transformación no requiere revisión manual, incluye varios niveles de abstracción en features y es útil para validación de requisitos con el cliente. Una evaluación de las reglas de transformación fue incluida en la sección 6.3.

Con respecto a la extensión del enfoque MDD para transformar modelo de interfaz de usuario abstracta con variabilidades a modelo de features no se halló otro trabajo que realice este tipo de transformación. En los casos de uso más grandes del caso de estudio se obtuvieron en promedio, 2 variabilidades de interfaz de usuario que necesitaran ser transformadas a modelo de features, lo cual justifica que es un asunto que debe ser tenido en cuenta.

Para el asunto de definición de proceso de configuración de modelo de features y modelos de dominio en UML se definió un proceso que combina transformación de modelos y configuración automática basada en modelo de features, que contempla restricciones de dependencia en las transformaciones y para las cuales, no es necesario definir modelo matemáticos intermedios. Las transformaciones se adaptan a los modelos definidos en este trabajo tanto para features como para modelos de dominio con variabilidades. Se definió una herramienta web para las transformaciones de modelo, esto permite que la única tarea del usuario sea la selección de las features correspondientes a la aplicación deseada a desarrollar.

Sobre la definición de proceso de desarrollo de interfaz de usuario para aplicaciones web se definió una tabla para mapear elementos de modelo de interfaz de usuario abstracta a elementos de tecnología (widgets de frameworks o HTML5), se brindaron patrones de código para interconectar interfaz de usuario correspondiente a modelo de acceso a funcionalidades/contenido con interfaz de usuario proveniente de desarrollo de casos de uso y se dieron guías para implementación utilizando la herramienta Pingendo. Siguiendo este proceso se realizó la implementación de la parte OPAC del caso de estudio de Biblioteca Online, esto permitió observar que el proceso definido permite agilizar la implementación de una interfaz de usuario (de interfaz de usuario abstracta se mapea a elementos de tecnología sin modelos intermedios) y minimiza la posibilidad de cometer errores durante esta etapa (siguiendo tablas de mapeo, reglas de uso de la herramienta Pingendo y patrones de código fuente).

14.3. Perspectivas y trabajo futuro

A continuación se presenta una lista de algunas posibles continuaciones, extensiones o mejoras al trabajo presentado aquí que pensamos que pueden ser interesantes de investigar:

- Mejora de niveles de abstracción en el modelo de features generado a partir de diagrama de casos de uso de dominio. Para dicho fin se puede generar de manera automática una versión del diagrama de casos de uso de dominio que estructure en paquetes teniendo en cuenta niveles de abstracción a través de herramientas de procesamiento de lenguaje natural, por ejemplo con agrupamientos de casos de uso basados en merónimos.
- Inclusión de requisitos no funcionales con variabilidades como parte del proceso de desarrollo de familia de aplicaciones web utilizando la notación definida en este trabajo. En este trabajo presentamos una notación que extiende el framework NFR para modelar variabilidades pero no hemos incluido esta notación dentro del proceso de desarrollo de familia de aplicaciones web que presentamos. Esta notación puede ser utilizada (incluso sin la extensión con variabilidades presentada aquí) para contemplar requisitos no funcionales que pueden ser tenidos en cuenta en distintas etapas del desarrollo de familia de aplicaciones web.
- Consideración de productos variantes para definir familia de aplicaciones web. En este trabajo hemos seguido un método de desarrollo de familia de aplicaciones desde cero, otra posibilidad sería utilizar productos variantes.
- Realización de un estudio y análisis de variabilidades en interfaces de usuario en diferentes dominio de aplicaciones web (por ejemplo aplicaciones web de clima, aplicaciones web de biblioteca, aplicaciones web de noticias). En este trabajo hemos identificado algunas situaciones interesantes en interfaces de usuario que pueden ser modeladas con variabilidades pero no hemos hecho un estudio pormenorizado que identifique aspectos comúnmente variables en interfaces de usuario de distintos dominio de aplicaciones web, ya que las interfaces pueden variar bastante de acuerdo al dominio.
- Traducción de modelo de acceso a funcionalidades y contenido a prototipos que puedan ser validados por clientes. El modelo de acceso a funcionalidades y contenido modela de manera abstracta la

interfaz de usuario global de una aplicación web. Una posibilidad de continuación del trabajo es traducir este modelo a un modelo más concreto basado en prototipos que puedan ser validados por los clientes.

- Extensión del proceso de desarrollo de aplicaciones incorporando consideraciones de arquitectura y acceso a datos. En el trabajo que presentamos aquí incluimos un proceso para implementar y definir interfaces de usuario, en este proceso se podría extender para tener en cuenta consideraciones de acceso a información o arquitectura de la aplicación web.
- Generación automática de código fuente a partir de modelo de interfaz de usuario abstracta y modelo de acceso a funcionalidades y contenido. Durante el proceso que presentamos aquí para implementar y definir interfaces de usuario se mapean elementos de interfaz de usuario abstracta a widgets de frameworks responsivos, esta tarea se puede automatizar (y extender al modelo de acceso a funcionalidades y contenido) mediante una transformación de modelos a tecnología de implementación.
- Extensión de las notaciones definidas y el proceso de desarrollo para contemplar familia de aplicaciones móviles. Este trabajo se centra en el dominio de aplicaciones web. El auge de los dispositivos móviles hicieron crecer el desarrollo de aplicaciones para estos dispositivos en los últimos años, por lo cual consideramos una buena alternativa ampliar el alcance del proceso definido aquí para el dominio de aplicaciones móviles.

14.4. Tendencias y avances en el período 2017–2020 en el ámbito de aplicabilidad del trabajo

Si bien se tuvo en cuenta la bibliografía más reciente –previa a la finalización de la tesis– en la revisión bibliográfica realizada para identificar trabajos relacionados en las contribuciones realizadas, vale la pena mencionar algunos avances y tendencias en los asuntos donde se encuentran los principales contribuciones de este trabajo.

En cuanto a *familias de aplicaciones web*, han surgido distintas propuestas que presentan enfoques que hacen uso de los beneficios de familias de aplicaciones para dominios particulares de aplicaciones web. En general, estos tipos de trabajos muestran cómo aplicar enfoques existentes de familia de aplicaciones (o combinaciones de ellos) a dominios delimitados y específicos de aplicaciones web. Entre estos trabajos se encuentran: [Cortinas, 2017] y [Cortinas, 2017b], que desarrollan una familia de productos para el área de Sistemas de Información Geográfica basados en web; [Zaki, 2017], en donde se realiza un análisis de partes en común y variables para construir un modelo de features y un diagrama de casos de uso de dominio para familia de sistemas de aprendizaje y enseñanza en línea; [Wolfart, 2019], que hace una investigación y análisis de productos de software o aplicaciones web de código abierto y explica cómo este tipo de desarrollo puede utilizarse para la adopción de SPL; y [Sridhar, 2020], en donde presentan una familia de líneas de productos de software como una forma de abordar el desafío de modelar una familia de sistemas de aprendizaje electrónico y demostrarlo para el caso de alfabetización de adultos en India.

Para el área de *requisitos y generación de features para familias de aplicaciones* surgieron algunos trabajos que extraen features o generan modelos de features a partir de descripciones de productos. Los trabajos de [Li, 2018] y [Sree-Kumar, 2018] utilizan técnicas de procesamiento de lenguaje natural para construir, de manera semi-automática, modelos de features de una línea de productos de software. El trabajo de [Kaindl, 2018] utiliza machine learning de generalización inductiva a partir de ejemplos para generar modelos de features de una familia de aplicaciones a partir de diagramas de features configurados de productos.

En cuando a *diseño de interfaz de usuario responsiva*, han surgido algunos trabajo que exploran distintos aspectos a tener en cuenta en la adopción de este enfoque de diseño. El trabajo de [Almeida, 2017] realiza un interesante estudio sobre el área y explora las principales ventajas y limitaciones de la adopción de diseño responsivo para interfaces de usuario. En [Cosgrove, 2018], se estudia la usabilidad y aspectos de experiencia de usuario de un sitio web de manejo de emergencias y seguridad, enfatizando en la importancia de la práctica de diseño web responsivo. En [Qasim, 2017] se hace un estudio de la

adopción de diseño responsivo para construir sitios web, se mencionan algunas carencias (como la personalización) y se dan algunas ideas para solucionarlas.

Sobre *interfaz de usuario para acceso a funcionalidades y contenido* se mantiene una cierta tendencia de años anteriores de especificar algunos aspectos de accesos a funcionalidades y contenido con notaciones informales, tales como mockups y wireframes, para luego generar código fuente. Como ejemplos se pueden mencionar los trabajos de [Bajammal, 2018], [Hassan, 2018] y [Beltramelli, 2018] (el cual utiliza redes neuronales).

Appendices

Apéndice A

Código fuente transformaciones en ATL

Este apéndice muestra el código fuente de las transformaciones realizadas en ATL para generar el modelo de features de la familia de aplicaciones. Se incluyen la transformación que toma como entrada el modelo de casos de uso de la familia de aplicaciones y genera la primera versión del modelo de features, la transformación que toma como entrada a los diagramas de actividad de la familia de aplicaciones y genera la segunda versión del modelo de features y, finalmente, la transformación que toma como entrada las partes de interfaz de usuario abstracta correspondientes a casos de uso de la familia de aplicaciones y genera el diagrama de features de la familia de aplicaciones.

A.1. Transformación de diagramas de casos de uso de dominio a modelos de features

```
module ucTofm; create OUT : FM from IN : UC;

helper context UC!UseCase def: repetedFeature() : Boolean =
if FM!Feature.allInstances()->exists(f | f.name = self.name and
FM!Relation.allInstances()->exists(c | c.childs->exists(f|
f.oclIsKindOf(FM!Feature) and f.name = self.name)))
then true
else false
endif;

helper context UC!Package def: repetedFeaturePK() : Boolean =
if FM!Feature.allInstances()->exists(f | f.name = self.name and
FM!Relation.allInstances()->exists(c | c.childs->exists(f|
f.oclIsKindOf(FM!Feature) and f.name = self.name)))
then true
else false
endif;

rule system2system{
from
input:UC!UCSystem
to
output:FM!FeatureModel(
name <- input.name,
rootFeature <- rootF
),
rootF:FM!Feature(
```

```

name <- input.name)
}

rule UCtoFeature{
from
input:UC!UseCase(not UC!Package.allInstances()->exists(pk |
pk.usecases->exists(uc | uc = input)))
to
output:FM!Feature(
name <- input.name
)
}

rule PackeageToFeature{
from
input:UC!Package(not UC!PackageVariability.allInstances()->exists(var |
var.optionsPK->exists(pk | pk = input)) and
(UC!Package.allInstances()->forall(pk | pk.nestedPackage->forall(pkn |
pkn <> input))))
to
output:FM!Feature(
name <- input.name
),
relation:FM!SingleRelation(
parent <- thisModule.resolveTemp(UC!UCSystem.allInstances()->first(),'rootF'),
minCardinality <- 1,
maxCardinality <- 1,
childs <- input
)
}

rule PackeageWithVarToFeature{
from
input:UC!Package(UC!PackageVariability.allInstances()->exists(var |
var.optionsPK->exists(pk | pk = input))
and UC!Package.allInstances()->forall(pk | pk.nestedPackage->forall(pkn |
pkn <> input)))
to
output:FM!Feature(
name <- input.name
)
}

rule PackeageWithVarToFeature{
from
input:UC!Package(UC!PackageVariability.allInstances()->exists(var |
var.optionsPK->exists(pk | pk = input))
and UC!Package.allInstances()->forall(pk | pk.nestedPackage->forall(pkn |
pkn <> input)))
to
output:FM!Feature(
name <- input.name
)
}

rule nestedPackageWithVar{

```

```

from
input:UC!Package(UC!Package.allInstances()->exists(pk |
pk.nestedPackage->exists(pkn | pkn = input))
and (UC!PackageVariability.allInstances()->exists(var |
var.optionsPK->exists(pk | pk = input)))
)
to
output:FM!Feature(
name <- input.name
)
}

rule AssociationUCToRelation{
from
input:UC!Association(input.usecase <> OclUndefined and
not UC!AssociationVariability.allInstances()->exists(var |
var.options->exists(as | as = input))
and not UC!Package.allInstances()->exists(pk |
pk.usecases->exists(uc | uc = input.usecase)))
to
relation:FM!SingleRelation(
parent <- thisModule.resolveTemp(UC!UCSystem.allInstances()->first(),'rootF'),
minCardinality <- 1,
maxCardinality <- 1,
childs <- if input.usecase.repetedFeature() then
thisModule.createFeatureFromRepetedAssociation(
input.actor.name.concat(' uses ').concat(input.usecase.name) )
else input.usecase
endif
)
}

rule UCinPackagetToFeatureWithoutVar{
from
input:UC!UseCase(UC!Package.allInstances()->exists(pk |
pk.usecases->exists(uc | uc = input))
and not UC!AssociationVariability.allInstances()->exists(var |
var.options->exists(ac | ac.usecase = input))
and not UC!ExtendVariability.allInstances()->exists(var |
var.options->exists(uc2 | uc2.extension = input))
and not UC!IncludeVariability.allInstances()->exists(var |
var.options->exists(uc2 | uc2.addition = input))
and UC!Association.allInstances()->exists(as | as.usecase = input))
to
output:FM!Feature(
name <- input.name
),
relation:FM!SingleRelation(
parent <- UC!Package.allInstances()->select(pk |
pk.usecases->exists(uc | uc = input))->first(),
minCardinality <- 1,
maxCardinality <- 1,
childs <- input
)
}

```

```

rule UCinPackagetoFeatureWithVar{
from
input:UC!UseCase(UC!Package.allInstances()->exists(pk |
pk.usecases->exists(uc | uc = input))
and (UC!AssociationVariability.allInstances()->exists(var |
var.options->exists(ac | ac.usecase = input))
or UC!ExtendVariability.allInstances()->exists(var |
var.options->exists(uc2 | uc2.extension = input))
or UC!IncludeVariability.allInstances()->exists(var |
var.options->exists(uc2 | uc2.addition = input))
or not UC!Association.allInstances()->exists(as | as.usecase = input)
))
to
output:FM!Feature(
name <- input.name
),
relation:FM!SingleRelation(
parent <- UC!Package.allInstances()->select(pk |
pk.usecases->exists(uc | uc = input))->first(),
minCardinality <- 1,
maxCardinality <- 1,
childs <- input
)
}

rule IncludeToFeature{
from
input:UC!Include(not UC!IncludeVariability.allInstances()->exists(var |
var.options->exists(ic | ic = input)))
to
relation:FM!SingleRelation(
parent <- input.includingCase,
minCardinality <- 1,
maxCardinality <- 1,
childs <- if input.addition.repetedFeature() then
thisModule.uc2reference(input.addition) else input.addition endif
)
}

rule IncludeVariabilityToSingleFeature{
from
input:UC!IncludeVariability(input.options.size() < 2)
to
relation:FM!SingleRelation(
parent <- input.options.first().includingCase,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.options->iterate(
at; acc : Sequence(UC!UseCase)=Sequence{} |
if at.addition.repetedFeature() then
acc->including(thisModule.uc2reference(at.addition))
else acc->including(at.addition) endif
)
)
}

```

```
rule IncludeVariabilityToGroupFeature{
from
input:UC!IncludeVariability(input.options.size() > 1)
to
relation:FM!GroupRelation(
parent <- input.options.first().includingCase,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.options->iterate(
at; acc : Sequence(UC!UseCase)=Sequence{} |
if at.addition.repetedFeature() then
acc->including(thisModule.uc2reference(at.addition))
else acc->including(at.addition) endif
)
)
}

rule ExtendToFeature{
from
input:UC!Extend(not UC!ExtendVariability.allInstances()->exists(var |
var.options->exists(ex | ex = input)))
to
relation:FM!SingleRelation(
parent <- input.extendedCase,
minCardinality <- 1,
maxCardinality <- 1,
childs <- if input.extension.repetedFeature() then
thisModule.uc2reference(input.extension) else input.extension endif
)
}

rule ExtendVariabilityToSingleFeature{
from
input:UC!ExtendVariability(input.options.size() < 2)
to
relation:FM!SingleRelation(
parent <- input.options.first().extendedCase,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.options->iterate(
at; acc : Sequence(UC!UseCase)=Sequence{} |
if at.extension.repetedFeature() then
acc->including(thisModule.uc2reference(at.extension))
else acc->including(at.extension)
endif
)
)
}

rule ExtendVariabilityToGroupFeature{
from
input:UC!ExtendVariability(input.options.size() > 1)
to
relation:FM!GroupRelation(
parent <- input.options.first().extendedCase,
minCardinality <- input.min,
```

```

maxCardinality <- input.max,
childs <- input.options->iterate(
at; acc : Sequence(UC!UseCase)=Sequence{} |
if at.extension.repetedFeature() then
acc->including(thisModule.uc2reference(at.extension))
else acc->including(at.extension)
endif)
)
}

rule VariabilityToSingleRelation{
from
input:UC!AssociationVariability(input.options.size() < 2)
to
output:FM!SingleRelation(
parent <- if not UC!Package.allInstances()->exists(pk | pk.usecases->exists(uc |
uc = input.options->first().usecase))
then thisModule.resolveTemp(UC!UCSystem.allInstances()->first(),'rootF')
else UC!Package.allInstances()->select(pk | pk.usecases->exists(uc |
uc = input.options->first().usecase))->first()
endif,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.options->iterate(
at; acc : Sequence(UC!Association)=Sequence{} |
if at.usecase.repetedFeature() then
acc->including(thisModule.createFeatureFromRepetedAssociation
(at.actor.name.concat(' uses ').concat(at.name) ) )
else acc->including(at.usecase)
endif)
)
}

rule VariabilityToGroupRelation{
from
input:UC!AssociationVariability(input.options.size() > 1)
to
output:FM!GroupRelation(
parent <- if not UC!Package.allInstances()->exists(pk | pk.usecases->exists(uc |
uc = input.options->first().usecase))
then thisModule.resolveTemp(UC!UCSystem.allInstances()->first(),'rootF')
else UC!Package.allInstances()->select(pk | pk.usecases->exists(uc |
uc = input.options->first().usecase))->first()
endif,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.options->iterate(
at; acc : Sequence(UC!Association)=Sequence{} |
if at.usecase.repetedFeature() then
acc->including(thisModule.createFeatureFromRepetedAssociation(
at.actor.name.concat(' uses ').concat(at.name) ) )
else acc->including(at.usecase)
endif)
)
}

```

```

rule PackageVariabilityToSingleRelation{
from
input:UC!PackageVariability(input.optionsPK.size() < 2)
to
output:FM!SingleRelation(
parent <- if UC!Package.allInstances()->forall(pk | pk.nestedPackage->forall(pkn |
  pkn <> input.optionsPK.first()))
then FM!Feature.allInstances()->first()
else UC!Package.allInstances()->select(pk | pk.nestedPackage->exists(pkn |
  pkn = input.optionsPK.first()))->first()
endif,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.optionsPK->iterate(
at; acc : Sequence(UC!Package)=Sequence{} |
if at.repetedFeaturePK() then acc->including(thisModule.pk2reference(at))
else acc->including(at)
endif)
)
}

```

```

rule PackageVariabilityToGroupRelation{
from
input:UC!PackageVariability(input.optionsPK.size() > 1)
to
output:FM!GroupRelation(
parent <- if not UC!Package.allInstances()->exists(pk |
  pk.nestedPackage->exists(pkn | pkn = input.optionsPK.first()))
then FM!Feature.allInstances()->first()
else UC!Package.allInstances()->select(pk | pk.nestedPackage->exists(pkn |
  pkn = input.optionsPK.first()))->first()
endif,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.optionsPK->iterate(
at; acc : Sequence(UC!Package)=Sequence{} |
if at.repetedFeaturePK() then acc->including(thisModule.pk2reference(at))
else acc->including(at)
endif)
)
}

```

```

lazy rule createFeatureFromRepetedAssociation{
from
inputA: String
to
output:FM!Feature(
name <- inputA
)
}

```

```

lazy rule uc2reference{
from
inputA: UC!UseCase
to

```



```

output:FM!Reference(
feature <- inputA
)
}

```

```

lazy rule pk2reference{
from
inputA: UC!Package
to
output:FM!Reference(
feature <- inputA
)
}

```

A.2. Transformación de diagramas de actividad de dominio a modelos de features

```

rule activityToFeature{
from
input:UC!Activity(input.countADVariability() < 4 and input.inBehavior.oclIsUndefined())

to
output:FM!Feature(
name <- input.name
)
}

```

```

rule ControlFlowVariabilityEdgeToFeature{
from
input:UC!ActivityEdge(UC!ControlFlowVariability.allInstances()->exists(cf |
cf.optionsAE->exists(edge | edge = input)))
to
output:FM!Feature(
name <- input.target.name
)
}

```

```

rule ControlFlowVariabilityAGToFeature{
from
input:UC!VariabilityActivityGroup(UC!ControlFlowVariability.allInstances()->exists(cf |
cf.optionsAG->exists(ag | ag = input)))
to
output:FM!Feature(
name <- input.name
)
}

```

```

rule DataFlowVariabilityToFeature{
from
input:UC!ObjectNode(UC!DataFlowVariability.allInstances()->exists(df |
df.options->exists(on | on = input)))
to
output:FM!Feature(
name <- input.name
)
}

```

```

}

rule ActionToFeatureWithoutVar{
from
input:UC!Action(not UC!ControlFlowVariability.allInstances()->exists(cf |
cf.optionsAE->exists(ae | ae.target = input))
and ((input.stereotype = #job) or (input.stereotype = #search)))
to
output:FM!Feature(
name <- input.name
),
relation:FM!SingleRelation(
parent <- if (input.inGroup.notEmpty()) then input.inGroup.first()
else if (input.activity.inBehavior <> OclUndefined) then input.activity.inBehavior
else input.activity
endif endif,
minCardinality <- 1,
maxCardinality <- 1,
childs <- input
)
}

rule AGToFeatureWithoutVar{
from
input:UC!ActivityGroup(not UC!ControlFlowVariability.allInstances()->exists(cf |
cf.optionsAG->exists(ag | ag = input)))
to
output:FM!Feature(
name <- input.name
),
relation:FM!SingleRelation(
parent <-if (input.superGroup <> OclUndefined) then input.superGroup
else if (input.inActivity.inBehavior <> OclUndefined) then input.inActivity.inBehavior
else input.inActivity
endif endif,
minCardinality <- 1,
maxCardinality <- 1,
childs <- input
)
}

rule ControlFlowVariabilityToSingleRelation{
from
input:UC!ControlFlowVariability((input.optionsAE.size() + input.optionsAG.size()) = 1)
to
relation:FM!SingleRelation(
parent <- if input.optionsAE.size() = 1
then if (input.optionsAE.first().inGroup.notEmpty()) then
input.optionsAE.first().inGroup.first()
else if (input.optionsAE.first().activity.inBehavior <> OclUndefined) then
input.optionsAE.first().activity.inBehavior
else input.optionsAE.first().activity
endif endif
else if (input.optionsAG.first().superGroup <> OclUndefined) then
input.optionsAG.first().superGroup
else if (input.optionsAG.first().inActivity.inBehavior <> OclUndefined)

```

```

then input.optionsAG.first().inActivity.inBehavior
else input.optionsAG.first().inActivity
endif endif
endif,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- if input.optionsAE.size() = 1
then input.optionsAE.first()
else input.optionsAG.first()
endif,
name <- input.name
)
}

rule ControlFlowVaribilityToGroupRelation{
from
input:UC!ControlFlowVariability((input.optionsAE.size() + input.optionsAG.size()) > 1)
to
relation:FM!GroupRelation(
parent <- if input.optionsAE.first() <> OclUndefined
then if (input.optionsAE.first().inGroup.notEmpty()) then
input.optionsAE.first().inGroup.first()
else if (input.optionsAE.first().activity.inBehavior <> OclUndefined) then
input.optionsAE.first().activity.inBehavior
else input.optionsAE.first().activity
endif endif
else if (input.optionsAG.first().superGroup <> OclUndefined) then
input.optionsAG.first().superGroup
else if (input.optionsAG.first().inActivity.inBehavior <> OclUndefined) then
input.optionsAG.first().inActivity.inBehavior
else input.optionsAG.first().inActivity
endif endif
endif,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.optionsAE->union(input.optionsAG),
name <- input.name
)
}

rule DataFlowVaribilityToSingleRelation{
from
input:UC!DataFlowVariability(input.options.size() = 1)
to
output:FM!Feature(
name <- input.name
),
relation1:FM!SingleRelation(
parent <- if (input.options.first().inGroup.notEmpty()) then
input.options.first().inGroup.first()
else if (input.options.first().activity.inBehavior <> OclUndefined) then
input.options.first().activity.inBehavior
else input.options.first().activity
endif endif,
minCardinality <- if input.min = 0 then 0 else 1 endif,
maxCardinality <- 1,

```

```
childs <- output
),
relation2:FM!SingleRelation(
parent <- output,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.options.first()
)
}

rule DataFlowVariabilityToGroupRelation{
from
input:UC!DataFlowVariability(input.options.size() > 1 )
to
output:FM!Feature(
name <- input.name
),
relation1:FM!SingleRelation(
parent <- if (input.options.first().inGroup.notEmpty()) then
input.options.first().inGroup.first()
else if (input.options.first().activity.inBehavior <> OclUndefined)
then input.options.first().activity.inBehavior
else input.options.first().activity
endif endif,
minCardinality <- if input.min = 0 then 0 else 1 endif,
maxCardinality <- 1,
childs <- output
),
relation2:FM!GroupRelation(
parent <- output,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.options
)
}

rule ParameterSetToFeature{
from
input:UC!ParameterSet
to
output:FM!Feature(
name <- input.name
),
relation:FM!SingleRelation(
parent <- input.options.first().activity,
minCardinality <- 1,
maxCardinality <- 1,
childs <- output
)
}
```

A.3. Transformación de restricciones de dependencia en modelos de dominio a restricciones de dependencia en modelos de features

```
rule DependenceReqToDependenceFeature{
from
input:UC!DependenceConstrain(input.dType = #Requires)
to
dependence:FM!Require(
source <- input.dependent,
target <- input.dependenceOn
)
}
```

```
rule DependenceExToDependenceFeature{
from
input:UC!DependenceConstrain(input.dType = #Excludes)
to
dependence:FM!Exclude(
source <- input.dependent,
target <- input.dependenceOn
)
}
```

A.4. Reglas para mapeo de partes de interfaz de usuario con variabilidades a modelo de features

```
rule uiPartToFeature{
from
input:UC!UIParts
to
output:FM!Feature(
name <- 'UI'
),
relation:FM!SingleRelation(
parent <- thisModule.resolveTemp(UC!UCSystem.allInstances()->select(uc |
uc.name = input.name)->first(),input.name),
minCardinality <- 1,
maxCardinality <- 1,
childs <- input
)
}
```

```
rule UIVariabilityToSingleRelation{
from
input:UC!VariabilityUI(input.optionsUI.size() < 2)
to
output:FM!SingleRelation(
parent <- input.uiPart,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.optionsUI
)
}
```

```
rule UIVariabilityToSingleRelation{
from
input:UC!VariabilityUI(input.optionsUI.size() < 2)
to
output:FM!SingleRelation(
parent <- input.uiPart,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.optionsUI
)
}

rule UIVariabilityToGroupRelation{
from
input:UC!VariabilityUI(input.optionsUI.size() > 1)
to
output:FM!GroupRelation(
parent <- input.uiPart,
minCardinality <- input.min,
maxCardinality <- input.max,
childs <- input.optionsUI
)
}

rule UIElementToFeature{
from
input:UC!UiElement(UC!VariabilityUI.allInstances()->exists(var |
var.optionsUI->exists(ui | ui = input)))
to
output:FM!Feature(
name <- input.name
)
}
```

Apéndice B

Implementación de configuraciones

Para implementar la producción de una configuración válida y, en base a la misma, configurar modelos de features y los modelos de dominio de una familia de aplicaciones web se diseñó una herramienta web en Php que toma como entradas los archivos XMI producidos (para modelos de dominio) y generados (para modelo de features) utilizando la herramienta Eclipse, el framework EMF y el lenguaje de transformaciones ATL. En base al archivo XMI del modelo de features de la familia de aplicaciones generado automáticamente, la herramienta diseñada parsea y visualiza las features variantes para que el usuario realice la selección de features deseada y genera una configuración en una estructura de Array que es utilizada para configurar el modelo de features de la familia y los modelos de dominio de la familia de aplicaciones. Los resultados son visualizados en pantalla.

B.1. Implementando la producción de una configuración

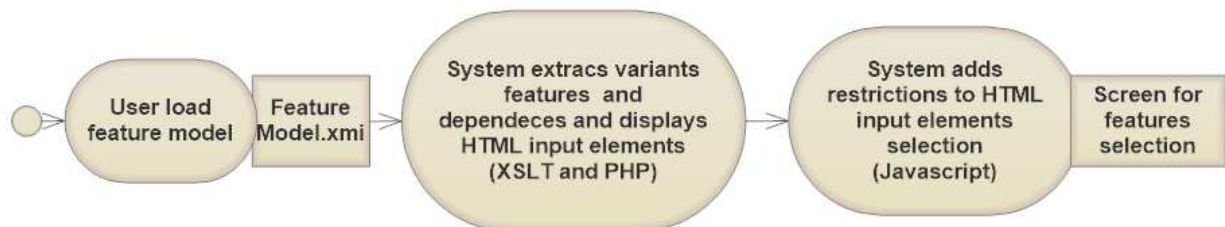


Figura B.1: Detalle de actividades para el proceso de producción de una configuración válida utilizando la herramienta realizada.

En la figura B.1 se muestra el proceso que se lleva a cabo para la producción por parte del usuario de una configuración válida, a través de selección de variantes. El proceso consta de dos actividades principales llevadas a cabo por el sistema; extracción de variabilidades y dependencias para conversión de variantes a elementos de input de selección de HTML y codificación de restricciones en JavaScript para alertar al usuario si la selección de un variante viola alguna regla para generar una configuración válida. La primer actividad toma como entrada el modelo de features en XMI generado en ATL con Eclipse que cumple con el meta-modelo de la figura 3.29. A continuación, solo a manera de ejemplificación, mostramos el diagrama de features (figura B.2) y el contenido del archivo de extensión xmi para un ejemplo sencillo de modelo de features correspondiente a características de un automóvil.

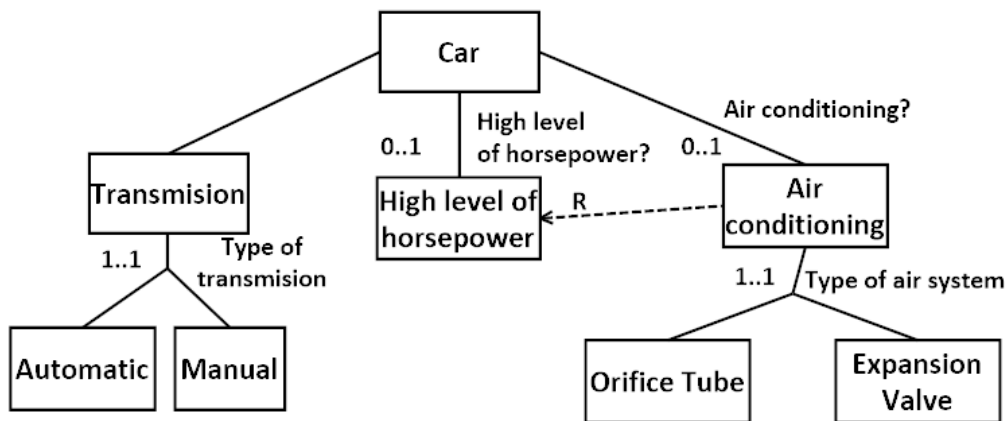


Figura B.2: Ejemplo simple de modelo de features para algunas características de un automóvil.

```

<?xml version="1.0" encoding="ASCII"?>
<features:FeatureModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<rootFeature name="Car">
<childs xsi:type="features:SingleRelation" minCardinality="1" maxCardinality="1">
<childs xsi:type="features:Feature" name="Transmission">
<childs xsi:type="features:GroupRelation" minCardinality="1" maxCardinality="1"
name="Type of transmission">
<childs xsi:type="features:Feature" name="Automatic"/>
<childs xsi:type="features:Feature" name="Manual"/>
</childs>
</childs>
</childs>
<childs xsi:type="features:SingleRelation" minCardinality="0" maxCardinality="1"
name="High level of horsepower?">
<childs xsi:type="features:Feature" name="High level of horsepower"/>
</childs>
<childs xsi:type="features:SingleRelation" minCardinality="0" maxCardinality="1"
name="Air acondionating?">
<childs xsi:type="features:Feature" name="Air conditioning">
<childs xsi:type="features:GroupRelation" minCardinality="1" maxCardinality="1"
name="Type of air system">
<childs xsi:type="features:Feature" name="Orifice Tube"/>
<childs xsi:type="features:Feature" name="Expansion Valve"/>
</childs>
</childs>
</childs>
</rootFeature>
<dependeces xsi:type="features:Require" source="//@rootFeature/@childs.2/@childs.0"
target="//@rootFeature/@childs.1/@childs.0"/>
</features:FeatureModel>

```

Utilizando una plantilla XSLT, se convierten las variabilidades en listas de HTML en forma de árbol donde en cada nodos se incluyen variantes por medio de elementos input de tipo checkbox para las features tradicionales y elementos input de tipo text para features parámetro. A continuación se muestra el código de la plantilla que realiza la transformación del modelo de features en xmi con variabilidades a HTML.

```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

```



```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<!-- Se copian todo los elementos del archivo de entrada original -->
<xsl:template match="rootFeature">
<ul>
<li><xsl:value-of select="./@name"/>
<xsl:apply-templates/>
</li>
</ul>
</xsl:template>

<!-- Si se encuentra un FeatureParametro se crea un input text de HTML -->
<xsl:template match="childs[@xsi:type = \'features:ParameterFeature\']">
<ul>
<li>
<xsl:value-of select="@valueSet"/><xsl:value-of select="@minCardinality"/>&lt;
<input type="text" rel="{./@name}" name="ParameterFeature-{@valueSet}-{@name}"
valueSet="{@valueSet}" value="{@minCardinality}" min="{@minCardinality}"
max="{@maxCardinality}" onblur="checkValueParam(this)" size="4" />&lt;<xsl:value-of
select="@maxCardinality"/>
<xsl:apply-templates/>
</li>
</ul>
</xsl:template>

<!-- Si se encuentra una relación SingleRelation con cardinalidad opcional,
se crea una lista con un item que es la feature opcional y
un input checkbox de HTML para seleccionarla o no -->
<xsl:template match="childs[@xsi:type = \'features:SingleRelation\'
and @minCardinality = \'0\']">
<ul><li type="SingleRelation" name="{@name}">0..1 <xsl:value-of select="./@name"/>
<xsl:for-each select="childs[@xsi:type = \'features:Feature\' or
@xsi:type = \'features:Reference\']">
<ul>
<li>
<input type="checkbox" rel="{./@name}" name="SingleRelation-{./@name}"
value="{@name}" min="{./@minCardinality}" max="{./@maxCardinality}"
onclick="checkValue(this)"/>
<xsl:value-of select="@name"/><xsl:apply-templates/>
</li>
</ul>
</xsl:for-each>
<xsl:for-each select="childs[@xsi:type = \'features:ParameterFeature\']">
<ul>
<li>
<input type="checkbox" rel="{./@name}" name="SingleRelation-{./@name}"
value="{@name}" min="{./@minCardinality}" max="{./@maxCardinality}"
onclick="checkValue(this)"/>
<xsl:value-of select="@name"/><xsl:apply-templates/>
<ul>
<li>
<xsl:value-of select="@valueSet"/><xsl:value-of select="@minCardinality"/>&lt;
<input type="text" rel="{./@name}" name="ParameterFeature-{@valueSet}-{@name}"
valueSet="{@valueSet}" value="{@minCardinality}" min="{@minCardinality}"
max="{@maxCardinality}" onblur="checkValueParam(this)" size="4" />
&lt;<xsl:value-of select="@maxCardinality"/>

```

```

<xsl:apply-templates/>
</li>
</ul>
</li>
</ul>
</xsl:for-each>
</li></ul>
</xsl:template>

<!-- Si se encuentra una relación GroupRelation de selección de variantes,
se crea una lista donde los items son las features seleccionables, para las
cuales se crea un input checkbox de HTML para seleccionarla o no -->

<xsl:template match="childs[@xsi:type = \'features:GroupRelation\' and
(@minCardinality != count(/childs) or @maxCardinality != count(/childs))]">
<ul><li type="GroupRelation" name="{@name}" min="{@minCardinality}"
max="{@maxCardinality}"><xsl:value-of
select="concat(@minCardinality,\'..\',@maxCardinality,\' \')"/> <xsl:value-of
select="./@name"/>
<xsl:for-each select="childs[@xsi:type = \'features:Feature\' or
@xsi:type = \'features:Reference\']">
<ul>
<li>
<input type="checkbox" rel="{./@name}" name="GroupRelation-{./@name}-{@name}"
value="{@name}" min="{./@minCardinality}" max="{./@maxCardinality}"
onclick="checkValue(this)"/>
<xsl:value-of select="@name"/><xsl:apply-templates/>
</li>
</ul>
</xsl:for-each>
<xsl:for-each select="childs[@xsi:type = \'features:ParameterFeature\']">
<ul>
<li>
<input type="checkbox" rel="{./@name}" name="SingleRelation-{./@name}"
value="{@name}" min="{./@minCardinality}" max="{./@maxCardinality}"
onclick="checkValue(this)"/>
<xsl:value-of select="@name"/><xsl:apply-templates/>
<ul>
<li>
<xsl:value-of select="@valueSet"/><xsl:value-of select="@minCardinality"/>&lt;
<input type="text" rel="{./@name}" name="ParameterFeature-{@valueSet}-{@name}"
valueSet="{@valueSet}" value="{@minCardinality}" min="{@minCardinality}"
max="{@maxCardinality}" onblur="checkValueParam(this)" size="4" />&lt;<xsl:value-of
select="@maxCardinality"/>
<xsl:apply-templates/>
</li>
</ul>
</li>
</ul>
</xsl:for-each>
</li></ul>
</xsl:template>

</xsl:stylesheet>

```

Aplicando la plantilla anterior y obteniendo en una estructura las restricciones de dependencia (a través de recorrer los elementos de tipo dependeces utilizando funcionalidades de Php para leer un elemento de

XML) se visualizan en pantalla los resultados obtenidos de la siguiente manera:

Features Configuration

- Car
 - 1..1 Type of transmission
 - Automatic
 - Manual
 - 0..1 High level of horsepower?
 - High level of horsepower
 - 0..1 Air acondicionating?
 - Air conditioning
 - 1..1 Type of air system
 - Orifice Tube
 - Expansion Valve

Requires:

- Air conditioning => High level of horsepower

Excludes:

Configure

Figura B.3: Pantalla generada para selección de variantes en ejemplo simple de modelo de features de un automóvil.

Posteriormente, se añadieron restricciones en JavaScript a los elementos HTML de input para que; al seleccionar un variante en tiempo de ejecución, si éste viola alguna regla para producir una configuración válida, se alerte al usuario de ello. A manera de ejemplificación, a continuación se muestra el código en JavaScript que alerta a un usuario, si éste está incluyendo valores fuera de rango para la configuración de una feature parámetro:

```
function checkValueParam(check){
  //se recibe como input el elemento input con atributos min, max y valueSet
  var min = Number(check.getAttribute("min"));
  var max = Number(check.getAttribute("max"));
  var valueSet = check.getAttribute("valueSet");
  //Si los valores no están en rango se muestra un mensaje y setea el valor min
  if (check.value > max || check.value < min){
    alert("Atención!\r\n\r\nEl valor de " + valueSet + " debe estar entre " + min +
      " y " + max);
    check.value = min;
  }
}
```

```

    }
}

```

B.2. Configurando modelos de features

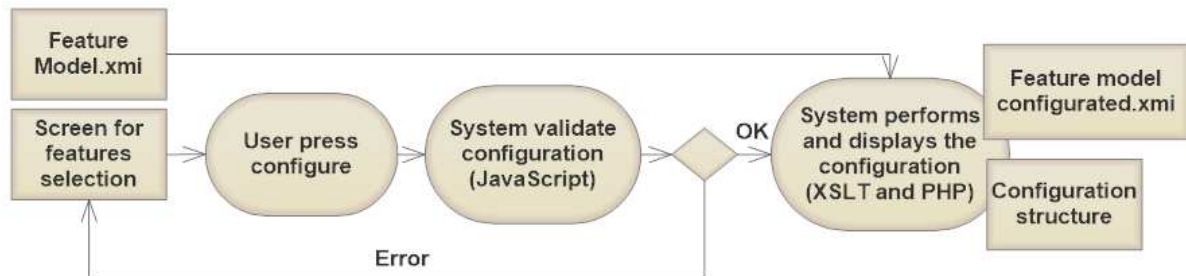


Figura B.4: Detalle de actividades para el proceso de configurar un modelo de features.

Una vez que el usuario seleccionó features para realizar su configuración deseada, el mismo debe presionar el botón *Configure* para llevar a cabo la configuración del diagrama de features proporcionado como entrada. Para realizar dicha tarea, como se puede apreciar en la siguiente figura, el sistema realiza dos actividades principales, validar la configuración en JavaScript y en caso de validación exitosa, se lleva a cabo y se visualiza la configuración, dando como resultado una estructura de Array de la configuración que será utilizada para configurar los modelos de UML con variabilidades.

Para la primera actividad se implementó una función en JavaScript que en base a los elementos HTML correspondientes a la selección de variantes del modelo de features chequea reglas y garantiza la producción de una configuración válida. A manera de ilustración a continuación incluimos una de las restricciones, incluidas en la función de JavaScript, la cual alerta a un usuario que, en caso de haber realizado una selección de variantes incorrecta, se debe seleccionar una cantidad de variantes en el rango de la cardinalidad mínima y la cardinalidad máxima y no lleva a cabo la configuración.

```

//Se obtienen todos los elementos correspondientes a GroupRelation
//y por cada variabilidad se cuentan la cantidad de fetures seleccionadas
var varsGR = document.forms["getConfig"].querySelectorAll("li[type='GroupRelation']");
for (i = 0; i < varsGR.length; i++) {
    if (varsGR[i].parentElement.style.display != "none"){
        var childrens = varsGR[i].querySelectorAll("input[rel='\" +
        varsGR[i].getAttribute("name") + "\']");
        var checkeds = 0;
        for (j = 0; j < childrens.length; j++) {
            if (childrens[j].checked){
                checkeds++;
            }
        }
    }
}
//si la cantidad de features seleccionada no es correcta se muestra un mensaje
//de alerta al usuario
if (checkeds > varsGR[i].getAttribute("max") ||
checkeds < varsGR[i].getAttribute("min")){
    alert("Atención!\r\n\r\n" + varsGR[i].getAttribute("name") +
": tiene que seleccionar entre " + varsGR[i].getAttribute("min") +
" y " + varsGR[i].getAttribute("max") + " features.");
    return false;
}
}
}

```

Por ejemplo, si en la relación de grupo del ejemplo del automóvil *Type of transmission* no se selecciona ningún variante (las cardinalidades mínima y máxima son 1) la herramienta responderá como se aprecia en la siguiente figura al presionar el botón *Configure* y evitará que se lleve a cabo la configuración hasta que no se haya seleccionado una configuración válida:

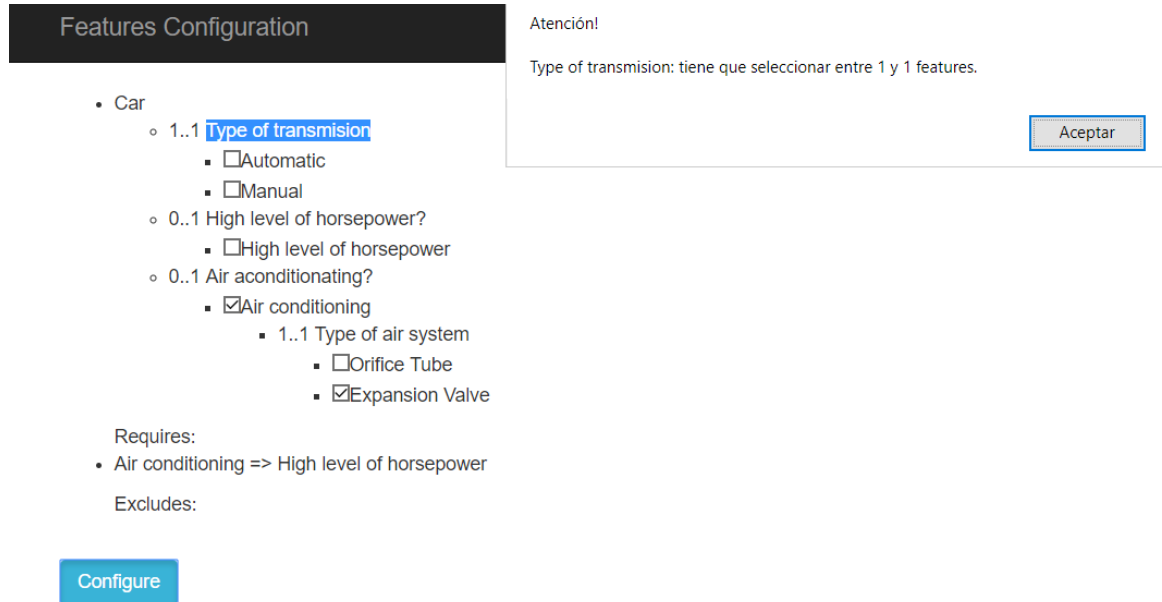


Figura B.5: Pantalla que se visualiza en caso de no cumplirse con la restricción de selección de variantes.

En la siguiente actividad, el sistema en base al modelo features en XMI de entrada, crea una estructura de configuración en un Array y a través de una plantilla de transformación XSLT y dicha configuración lleva a cabo la configuración del modelo de features y visualiza el resultado en pantalla. La plantilla XSLT para realizar la configuración, básicamente copia las features que han sido seleccionadas en la configuración, elimina las que no han sido seleccionadas, ajusta el tipo de relaciones de ser necesario y elimina las restricciones de dependencia. A continuación se visualiza el resultado de esta actividad para una configuración válida del ejemplo del automóvil:

Features Configuration

Configuration

```
Array ( [0] => Type_of_transmission [1] => GroupRelation [2] => Array ( [0] => Manual ) )
Array ( [0] => High_level_of_horsepower? [1] => SingleRelation [2] => Array ( [0] => High level of horsepower ) )
Array ( [0] => Air_aconditionating? [1] => SingleRelation [2] => Array ( [0] => Air conditioning ) )
Array ( [0] => Type_of_air_system [1] => GroupRelation [2] => Array ( [0] => Expansion Valve ) )
```

Feature Model configured

- - Car
 - Transmission
 - Type of transmission
 - Manual
 - High level of horsepower?
 - High level of horsepower
 - Air aconditionating?
 - Air conditioning
 - Type of air system
 - Expansion Valve

Figura B.6: Resultado de la configuración utilizando la herramienta realizada en ejemplo simple de modelo de features de un automóvil.

B.3. Configurando diagramas de de casos de uso de dominio

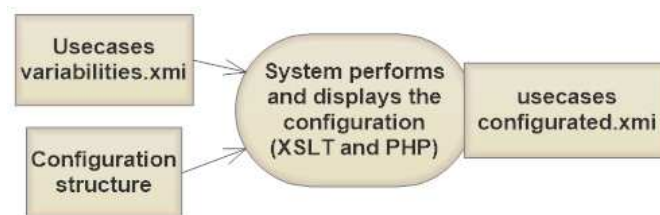


Figura B.7: Detalle de actividades para el proceso de configurar un diagrama de casos de uso con variabilidades.

Si el usuario proveyó al sistema el diagrama de casos de uso con variabilidades en formato XMI (al cargar el modelo de features el sistema permite luego cargar el diagrama de casos de uso con variabilidades), automáticamente el sistema a través de una plantilla de transformación XSLT y de la estructura de configuración ya generada lleva a cabo la configuración del diagrama de casos de uso con variabilidades y visualiza el resultado en pantalla. A continuación se visualiza el archivo XMI para el modelo de casos de uso con variabilidades correspondiente al ejemplo del automóvil.

```
<?xml version="1.0" encoding="ASCII"?>
<usecases:UCSystem xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:usecases="http://usecases/1.0">
<usecases name="Level of horsepower">
```

```

<actorassociation actor="//@actors.0" name="horse"/>
</usecases>
<usecases name="Air conditioning">
<actorassociation actor="//@actors.0" name="air"/>
</usecases>
<usecases name="Orifice Tube">
<extend extendedCase="//@usecases.1"/>
</usecases>
<usecases name="Expansion Valve">
<extend extendedCase="//@usecases.1"/>
</usecases>
<usecases name="Transmission">
<include addition="//@usecases.5"/>
<include addition="//@usecases.6"/>
<actorassociation actor="//@actors.0" name="transmision"/>
</usecases>
<usecases name="Automatic"/>
<usecases name="Manual"/>
<usecases name="Test">
<include addition="//@usecases.0"/>
</usecases>
<actors name="User"/>
<variablity xsi:type="usecases:AssociationVariability"
name="Level of horsepower?" max="1" min="0"
options="//@usecases.0/@actorassociation.0"/>
<variablity xsi:type="usecases:AssociationVariability"
name="Air aconditionating?" max="1" min="0"
options="//@usecases.1/@actorassociation.0"/>
<variablity xsi:type="usecases:ExtendVariability"
name="Type of air system" max="1" min="1"
options="//@usecases.2/@extend.0 //@usecases.3/@extend.0"/>
<variablity xsi:type="usecases:IncludeVariability"
name="Type of transmision" max="1" min="1"
options="//@usecases.4/@include.0 //@usecases.4/@include.1"/>
</usecases:UCSystem>

```

La plantilla XSLT para realizar la configuración, básicamente copia los casos de uso que han sido seleccionados en la configuración, elimina las que no han sido seleccionados, elimina todas las variabilidades y ajusta las relaciones entre casos de uso. Para copiar la estructura del diagrama de casos de uso y los casos de uso seleccionados se utiliza la siguiente regla en XSLT:

```

<xsl:template match="@*|node()">
<xsl:copy>
<xsl:apply-templates select="@*|node()"/>
</xsl:copy>
</xsl:template>

```

Para eliminar las variabilidades del modelo de casos de uso con variabilidades de entrada se utiliza la siguiente regla en XSLT:

```

<xsl:template match="variablity"/>

```

En base a recorrer la estructura de Array de configuración se van agregando reglas para eliminar casos de uso o paquetes (y relaciones de ser necesario). Por ejemplo la siguiente regla elimina un paquete no seleccionado en la estructura de configuración del diagrama de casos de uso configurado:

```

<xsl:template match="packages[@name = \''. $confValue[2][0] .'\']"/>

```

A continuación se visualiza el resultado de esta actividad para una configuración válida del ejemplo del automóvil:

UC configured

- USystem:Car
 - usecases:Level of horsepower
 - actorassociation -> User
 - usecases:Air conditioning
 - actorassociation -> User
 - usecases:Expansion Valve
 - extend to Air conditioning
 - usecases:Transmision
 - include => Manual
 - actorassociation -> User
 - usecases:Manual
 - usecases:Test
 - include => Level of horsepower
 - actors:User

Figura B.8: Resultado de la configuración aplicada a diagrama de casos de uso con variabilidades en ejemplo simple de modelo de features de un automóvil.

Bibliografía

- [Al-Msie'Deen, 2014] R. AL-Msie'Deen. *Reverse Engineering Feature Models from Software Variants to Build Software Product Lines*. PhD thesis, PhD thesis, University of Montpellier, 2014.
- [Almeida, 2017] F. Almeida, J. A. Monteiro. *The Role of Responsive Design in Web Development*. Webology 14. 2017.
- [Alves, 2008] V. Alves, Ch. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, Ch. Pohl, A. Rummler. *An Exploratory Study of Information Retrieval Techniques in Domain Analysis*. In Proceedings of the 12 th Intl. Software Product Line Conf. (SPLC'08), IEEE Computer Society, 2008, pp. 67-76.
- [António, 2009] S. António, J. Araújo, C. Silva. *Adapting the i* Framework for Software Product Lines*. In: ER'09. LNCS 5833, pp. 286-295, Springer-Verlag. 2009.
- [Asikainen, 2006] T. Asikainen, T. Mannisto, T. Soininen. *A Unified Conceptual Foundation for Feature Modeling*. Presented at SPLC'06, Baltimore, 2006.
- [Azevedo, 2012] S. Azevedo, R. Machado, A. Bragança, H. Ribeiro. *On the refinement of use case models with variability support*. ISSE 8(1), 51-64. 2012.
- [Bajammal, 2018] M. Bajammal, D. Mazinianian, A. Mesbah. *Generating reusable web components from mockups*. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018). Association for Computing Machinery, New York, NY, USA, 601?611. DOI:<https://doi.org/10.1145/3238147.3238194>. 2018.
- [Balzerani, 2005] L. Balzerani, D. Di Ruscio, A. Pierantonio. *A Product Line Architecture for Web Applications*. ACM Symposium on Applied Computing. 2005.
- [Bartholdt, 2010] J. Bartholdt, R. Oberhauser, A. Rytina, M. Medak. *Integrating Quality Modeling in Software Product Lines*. International Journal On Advances in Software, vol. 3, no 1 and 2, pp. 161-174, 2010.
- [Batori, 2005] D. Batori. *Feature Models, Grammars, and Propositional Formulas*. Springer-Verlag, p. 7-20 SPLC'05, 2005.
- [Beltramelli, 2018] T. Beltramelli. *Pix2code: Generating Code from a Graphical User Interface Screenshot*. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '18). Association for Computing Machinery, New York, NY, USA, Article 3, 1?6. DOI:<https://doi.org/10.1145/3220134.3220135>. 2018.
- [Benavides, 2005] D. Benavides, P. Trinidad, A. Ruiz-Cortés. *Automated reasoning on feature models*. In Advanced Information Systems Engineering, pp. 491-503, Springer Berlin Heidelberg. 2005.
- [Benlarabi, 2015] A. Benlarabi, A. Khtira, B. Asri. *Analyzing Trends in Software Product Lines Evolution Using a Cladistics Based Approach*. Information, 6(3):550-563, 2015.
- [Beuche, 2009] D. Beuche. *Transforming legacy systems into software product lines*. In Proceedings of the 13th International Software Product Line Conference, SPLC'09, pages 321- 321, Pittsburgh, PA, USA, 2009.

- [Blanes, 2014] D. Blanes, J. Gonzalez-Huerta, E. Insfran. *A multimodel approach for specifying the requirements variability on software product lines*. In: ISD'14, pp. 329-336, 2014.
- [Bosch, 2011] J. Bosch, P. M. Bosch-Sijtsema. *Introducing agile customer-centered development in a legacy software product line*. *Softw. Pract. Exper.*, 41(8):871-882. 2011.
- [Bragança, 2007] A. Bragança, R. J. Machado. *Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines*, In: Proceedings of the 6th Intl. Conf. on Software Product Lines (SPLC'07), IEEE Computer Society, 2007, pp 3-12. 2007.
- [Bragança, 2008] A. Bragança. *Methodological approaches and techniques for model driven development of software product lines*. PhD Thesis, 2008.
- [Brambilla, 2010] M. Brambilla, P. Fraternali, E. Molteni. *A Tool for Model-driven Design of Rich Internet Applications based on AJAX*. Handbook of Research on Web 2.0, 3.0, and X.0: Technologies, Business, and Social Applications, San Murugesan (ed.), pp. 96-118, IGI Global. 2010.
- [Bühne, 2003] S. Bühne, G. Halmans, K. Pohl. *Modeling Dependencies between Variation Points in Use Case Diagrams*. In: Proceedings of 9th Intl. Workshop on Requirements Engineering - Foundations for Software Quality, 59-69. 2003.
- [Bry, 2006] F. Bry, P. P?trânjan. *Reactivity on the Web: Paradigms and Applications of the Language XChange*. *J. Web Eng.* 5(1): 3-24. 2006.
- [Casalánguida, 2011] H. Casalánguida, J. E. Durán. *Requirements Engineering of Web Application Product Lines*. WEBIST 2011, Proceedings of the 7th International Conference on Web Information Systems and Technologies, Noordwijkerhout, The Netherlands, 6-9 May, 2011.
- [Casalánguida, 2012] H. Casalánguida, J. E. Durán. *Automatic Generation of Feature Models from UML Requirement Models*. In: 1st International Workshop on Requirements Engineering Practices on Software Product Line Engineering, Proceedings of 11th Intl. Conf on Software Product Lines (SPLC'12), Vol. 2. ACM, pp 10-17. 2012.
- [Casalánguida, 2013] H. Casalánguida, J. E. Durán, *A Method for Integrating Process Description and User Interface Use during Design of RIA Applications*. International Conference on Web Engineering 2013, Model Driven Web Engineering WorkshopAt: Aalborg, DenmarkVolume: Current Trends in Web Engineering- ICWE 2013 Workshops and PhD Symposium. 2013.
- [Casalánguida, 2015] H. Casalánguida, J. E. Durán, *Development of a Design Model for Functionality and Content Access from Rich Internet Application Requirements*. 11th International Conference on Web Information Systems and TechnologiesAt: Lisbon, Portugal. 2015.
- [Casalánguida, 2015b] H. Casalánguida, J. E. Durán, *User Interface Design for Responsive Web Applications*. 11th International Conference on Web Information Systems and TechnologiesAt: Lisbon, Portugal. 2015.
- [Chung, 2000] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos. *Non functional Requirements in Software Engineering*. Kluwer Academic Publisher, Boston. 2000.
- [Chung, 2000b] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos. *The NFR Framework in Action*. In: *Non-Functional Requirements in Software Engineering*. International Series in Software Engineering, vol 5. Springer, Boston, MA. 2000.
- [Cirilo, 2012] C. E. Cirilo, A. F. do Prado, W. Lopes de Souza, L. A. Martinez Zaina. *Building Adaptive Rich Interfaces for Interactive Ubiquitous Applications*. *Interactive Multimedia*", book edited by Ioannis Deliyannis, ISBN 978-953-51-0224-3, Chapter 11. 2012.
- [Clements, 2002] P. Clements, L. Northrop. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.

- [Cortinas, 2017] A. Cortiñas, M. R. Luaces, O. Pedreira, A S. Places, J- Pérez. *Web-based Geographic Information Systems SPL E: Domain Analysis and Experience Report*. In Proceedings of the 21st International Systems and Software Product Line Conference - Volume A (SPLC '17). Association for Computing Machinery, New York, NY, USA, 190-194. DOI:<https://doi.org/10.1145/3106195.3106222>. 2017.
- [Cortinas, 2017b] A. Cortiñas, M. R. Luaces, O. Pedreira, A S. Places. *Scaffolding and in-browser generation of web-based GIS applications in a SPL tool*. In Proceedings of the 21st International Systems and Software Product Line Conference - Volume B (SPLC '17). Association for Computing Machinery, New York, NY, USA, 46-49. DOI:<https://doi.org/10.1145/3109729.3109759>. 2017
- [Cosgrove, 2018] S. Cosgrove. *Exploring usability and user-centered design through emergency management websites: advocating responsive web design*. Commun. Des. Q. Rev 6, 2 (October 2018), 93-102. DOI:<https://doi.org/10.1145/3282665.3282674>. 2018.
- [Czarnecki, 2004] K. Czarnecki, S. Helsen, U. Eisenecker. *Staged Configuration Using Feature Models*. SPLC'04. LNCS 3154, pp. 266-283, 2004.
- [Czarnecki, 2005] K. Czarnecki, S. Helsen, U. Eisenecker. *Formalizing cardinality-based feature models and their specialization*. Software Process Improvement and Practice, 10(1):7-29, jan/mar 2005.
- [Czarnecki, 2007] Czarnecki, K., Wasowski, A. *Feature models and logics: There and back again*. In SPLC 2007. IEEE Press. 2007.
- [Dolog, 2006] P. Dolog. *Engineering Adaptive Web Applications*. PH-D thesis. Universität Hannover. 2006.
- [Fadhilillah, 2018] H. Fadhilillah, D. Adianto, A. Azurat, S. Sakinah. *Generating adaptable user interface in SPL E: using delta-oriented programming and interaction flow modeling language*. Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 2. Gothenburg, Sweden. 2018.
- [Filho, 2009] O. Filho, J. Ribeiro. *UWE-R: An Extension to a Web Engineering Methodology for Rich Internet Applications*. WSEAS Trans. Info. Sci. and App. 6(4): 601-610. Apr. 2009.
- [France, 2007] R. France, B. Rumpe. *Model-driven Development of Complex Software: A Research Roadmap*. Future of Software Engineering, pp. 37-54. 2007.
- [Gabillon, 2013] Y. Gabillon, N. Biri, B. Otjacques. *Methodology to Integrate Multi-context UI Variations into a Feature Model*. Proceedings of the 17th International Software Product Line Conference Co-located WorkshopsAt: Tokyo, Japan. 2013.
- [Gabillon, 2015] Y. Gabillon, N. Biri, B. Otjacques. *Designing an Adaptive User Interface According to Software Product Line Engineering*. The Eighth International Conference on Advances in Computer-Human Interactions (ACHI'15)At: Lisbon, Portugal. 2015.
- [Gerchev, 2018] I. Gerchev. *The 5 Most Popular Front-end Frameworks Compared*, <https://www.sitepoint.com/most-popular-frontend-frameworks-compared/>. 2018.
- [Ghiani, 2014] G. Ghiani, M. Manca, F. Paternò, C. Porta. *Beyond Responsive Design: Context-Dependent Multimodal Augmentation of Web Applications*. MobiWIS 2014: 71-85. 2014.
- [Gomaa, 2004] H. Gomaa. *Designing Software Product Lines with UML. From Use Cases to Pattern-Based Software Architectures*. Addison Wesley. 2004.
- [Gonzalez-Huerta, 2012] J. González-Huerta, E. Insfran, S. Abrahão. *A Multimodel for Integrating Quality Assessment in Model-Driven Engineering*. In Proceedings of Quatic2012, 2012.
- [Gonzalez-Huerta, 2012b] J. Gonzalez-Huerta, E. Insfran, S. Abrahão, J. Mcgregor. *Non-functional requirements in model-driven software product line engineering*. Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages (NFPinDSML 2012), co-located with MODELS. 2012.

- [Griss, 1998] M. L. Griss, J. Favaro, M. d'Álessandro. *Integrating Feature Modeling with the RSEB*. In Proceedings of 5th Intl. Conf. on Software Reuse ICSR '98 (2-5 June). 1998.
- [Hassan, 2018] S. Hassan, M. Arya, U. Bhardwaj, S. Kole. *Extraction and Classification of User Interface Components from an Image*. International Journal of Pure and Applied Mathematics. Volume 118 No. 24. 2018.
- [Heuer, 2010] A. Heuer, Ch. Budnik, S. Konrad, K. Lauenroth, K. Pohl. *Formal Definition of Syntax and Semantics for Documenting Variability in Activity Diagrams*. In Proceedings of the 9th Intl. Conf. on Software Product Lines (SPLC'10), vol. 6287 of LNCS, Springer, pp 62-76. 2010.
- [Hwang, 2017] B. Hwang, Y. Jin. *Application of Software Product Line Engineering for Developing Web Application Families*. DOI: 10.7838/jsebs.2017.22.2.039. 2017.
- [Jacob, 2015] J. Stein. *Supporting feature model configuration based on multi-stakeholder preferences*, Master's Thesis, Universidade Federal do Rio Grande do Sul, 2015.
- [Kaindl, 2018] H. Kaindl, S. Kramer, R. Hoch. *An inductive learning perspective on automated generation of feature models from given product specifications*. In Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1 (SPLC '18). Association for Computing Machinery, New York, NY, USA, 25?30. DOI:https://doi.org/10.1145/3233027.3233031. 2018.
- [Kang, 1990] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S Peterson. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report CMU/SEI-90-TR-021, SEI, Carnegie Mellon University. 1990.
- [Kang, 1998] K. Kang, S. Kim, J. Lee. *FORM: A Feature Oriented Reuse Method with Domain-Specific Reference Architectures*. Annals of Software Engineering 5 (1): 143-168. 1998.
- [Kapova, 2009] L. Kapova, T. Goldschmidt. *Automated Feature Model-based Generation of Refinement Transformations*. In Proceedings of 35th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA'09), IEEE, pp 141-148. 2009.
- [Khalique, 2017] F. Khalique, W. Haider Butt, S. Ahmad Khan. *Creating Domain Non-Functional Requirements in Software Product Line Engineering using Model Transformations*. International Conference on Frontiers of Information Technology. 2017.
- [Koch, 2008] N. Koch, A. Knapp, G. Zhang, H. Baumeister. *UML-Based Web Engineering. An Approach Based on Standards*. In Web Engineering: Modelling and Implementing Web Applications, Human Computer Interaction Series, Springer, pp. 157-191. 2008.
- [Koch, 2012] N. Koch, S. Kozuruba. *Requirements Models as First Class Entities in Model-Driven Web Engineering*. In ICWE'12, 12th Intl. Conf. on Web Engineering, LNCS vol. 7703: pp. 158-169, Springer Verlag. 2012.
- [Korherr, 2007] B. Korherr, B. List. *A UML 2 Profile for Variability Models and their Dependency to Business Processes*. In: Proceedings of the 18th Intl. Conf. on Database and Expert Systems Applications (DEXA'07), IEEE, pp 829-834. 2007.
- [Kulesza, 2005] U. Kulesza, A. García, F. Bleasby, C. Lucena. *Instantiating and Customizing Product Line Architectures using Aspects and Crosscutting Feature Models*. In EA'05, Workshop on Early Aspects. 2005.
- [Laguna, 2011] M. Laguna, J. Marqués, G. Rodríguez-Cano. *Feature diagram formalization based on directed hypergraphs*. Comput. Sci. Inf. Syst. 8(3): 611-633, 2011.
- [Lee, 2006] S. Lee. *Modeling Variant User Interfaces for Web-Based Software Product Lines*. International Journal of Information Technology and Web Engineering (IJITWE). 2006.
- [Li, 2018] Y. Li, S. Schulze, G. Saake. *Reverse engineering variability from requirement documents based on probabilistic relevance and word embedding*. In Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1 (SPLC '18). Association for Computing Machinery, New York, NY, USA, 121?131. DOI:https://doi.org/10.1145/3233027.3233033. 2018.

- [Linden, 2002] F. van der Linden. *Software product families in europe: the esaps and cafe projects*. Software, IEEE, 19(4):41-49. TY - JOUR. 2002.
- [Linden, 2008] F. van der Linden, J. Bermejo, B. Lundell. *Combining the Advantages of Product Lines and Open Source*. Dagstuhl Seminar 08142. 2008.
- [Manca, 2013] M. Manca, F. Paternò, C. Santoro, L. D. Spano. *Generation of Multi-Device Adaptive MultiModal Web Applications*. MobiWIS 2013: 218-232. 2013.
- [Martinez, 2009] J. Martinez, C. Lopez, E. Ulacia, M. del Hierro. *Towards a Model-Driven Product Line for Web systems*. In: 5th Model-Driven Web Engineering Workshop MDWE. 2009.
- [Martínez Ruiz, 2007] F. J. Martínez Ruiz. *A Development Method for User Interfaces of Rich Internet Applications*. A Thesis for the degree of Diploma of Extended Studies in Management Science. Catholic University of Leuven. 2007.
- [Mbaki, 2008] E. Mbaki, J. Vanderdonckt, J. Guerrero, M. Winckler. *Multi-level Dialog Modeling in Highly Interactive Web Interfaces*. In: Proceedings of the 8th Intl. Conf. on Web Engineering: pp. 44?49, IEEE Computer Society, Los Alamitos CA, USA. 2008.
- [Mefteh, 2015] M. Mefteh, N. Bouassida, H. Ben-Abdallah, *Implementation and Evaluation of an Approach for Extracting Feature Models from Documented Use Case Diagrams*, In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, ACM, pp. 1602-1609. 2015.
- [Mendonca, 2008] M. Mendonca, A. Wasowski, K. Czarnecki, D. Cowan. *Efficient compilation techniques for large scale feature models*. Proceedings of the 7th International Conference on Generative Programming and Component Engineering p13-22, GPCE '08, 2008.
- [Miles, 2007] R. Miles, K. Hamilton. *Learning UML 2.0*. O'Reilly. 2007.
- [Nare, 2008] A. Nare. *Introduction to Model Introduction to Model-Driven Development (MDD)*. The Virtual Enterprise (VE). 2008.
- [Nerome, 2014] T. Nerome, M. Numao. *A Product Domain Model based Software Product Line Engineering for Web Application*. Second International Symposium on Computing and Networking. 2014.
- [Norian, 2012] M. Norian, E. Bagheri, W. Du. *Non-functional Properties in Software Product Lines: A Taxonomy for classification*. In *proc. of 24th International Conference on Software Engineering and Knowledge Engineering (SEKE 2012)*. 2012.
- [OMG, 2009] Object Management Group, Inc. (OMG). *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.2*. [Online]. <http://www.omg.org/spec/UML/2.2/Superstructure/PDF/>. February 2009.
- [Ouali, 2015] S. Ouali, N. Kraïem, Z. Al-Khanjari, Y. Baghdadi. *Model Driven Software Product Line Process for Service/Component-Based Applications*, published in JSW2015, 2015.
- [Paternò, 2009] F. Paternò, C. Santoro, L. D. Spano. *MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments*. ACM Trans. Comput.-Hum. Interact., 16(4), November, pp 1-30, 2009.
- [Paternò, 2013] F. Paternò. *End User Development: Survey of an Emerging Field for Empowering People*. ISRN Software Engineering, Vol. 2013, Article ID 532659. 2013.
- [Peterson, 2014] C. Peterson. *Learning Responsive Web Design*. O'Reilly Media, Inc. ISBN: 9781449362942. 2014.
- [Pleuss, 2010] A. Pleuss, G. Botterweck, D. Deepak. *Integrating Automated Product Derivation and Individual User Interface Design*. Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria. 2010.

- [Pohl, 2005] K. Pohl, G. Böckle, F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, Berlin, Heidelberg. 2005.
- [Plechawska-Wojcik, 2012] M. Plechawska-Wojcik, S. Lujan Mora, L. Wojcik. *Assessment of User Experience with Responsive Web Applications using Expert Method and Cognitive Walkthrough*. In Proceedings of the 15th International Conference on Enterprise Information Systems (ICEIS-2013), pages 111-118. 2013.
- [Qasim, 2017] M. Qasim, K. Ullah, W. Ashraf. *Responsive web design complication and viable solutions for design implementation operation*. International Journal of Computer Science and Information Security (IJCSIS), Vol. 15, No. 1. 2017.
- [Razavian, 2008] M. Razavian, R. Khosravi. *Modeling Variability in Business Process Models Using UML*. In: Proceedings of the 5th Intl. Conf. on Inf. Technology: New Generations, IEEE, pp 82-87. 2008.
- [Riebisch, 2000] M. Riebisch, K. Böllert, D. Streitferdt, B. Franczyk. *Extending the UML to Model System Families*. In IDPT 2000, Integrated Design and Process Technology. Society for Design and Process Science. 2000.
- [Riebisch, 2002] M. Riebisch, K. Böllert, D. Streitferdt, I. Philippow. *Extending Feature Diagrams with UML Multiplicities*. 6th World Conference on Integrated Design and Process Technology (IDPT2002), Pasadena, CA, USA; June 23 - 27, 2002.
- [Ripon, 2013] S. Ripon, Sk. Jahir Hossain, T. Bhuiyan. *Managing and Analysing Software Product Line Requirements*. International Journal of Software Engineering & Applications (IJSEA), Vol.4, No.5, September 2013.
- [Robak, 2002] S. Robak, B. Franczyk, K. Politowicz. *Extending the UML for Modelling Variability for System Families*. In Intl. Journal of Appl. Math. Comput. Science, Vol.12, No.2, pp 285 - 298. 2002.
- [Rosado da Cruz, 2010] Rosado da Cruz, A. *Automatic Generation of User Interfaces from Rigorous Domain and Use Case Models*. Ph-D Thesis, Departamento de Engenharia Informática, Faculdade de Engenharia da Universidade do Porto. 2010.
- [Saravanakumar.org, 2019] saravanakumar.org CSS Frameworks compare. <https://saravanakumargn.github.io/css-frameworks-compare/>. 2019.
- [Schnieders, 2007] A. Schnieders, M. Weske. *Activity Diagram Based Process Family Architectures for Enterprise Application Families*. In: Part II of Enterprise Interoperability, 1st edn., Springer, pp 67-76. 2007.
- [Schobbens, 2007] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, Y. Bontemps. *Generic semantics of feature diagrams*. Computer Networks, vol. 51, no. 2, pp. 456-479. 2007
- [Seidl, 2016] C. Seidl, T. Winkelmann, I. Schaefer. *A Software Product Line of Feature Modeling Notations and Cross-Tree Constraint Languages* Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn. 2016.
- [Seixas, 2019] J. Seixas, A. Ribeiro, A. Rodrigues da Silva. *A Model-Driven Approach for Developing Responsive Web Apps*. In Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019). SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 257?264. DOI:<https://doi.org/10.5220/0007678302570264>. 2019.
- [Semmak, 2008] F. Semmak, C. Gnaho, R. Laleau. *Extended KAOS to Support Variability for Goal Oriented Requirements Reuse*. In MoDISE-EUS'08, Vol. 341 of CEUR Workshop Proceedings, pp 22-33. 2008.
- [Sidi, 2017] M. Sidi, O. Moulaye Abdellahi, M. SENE, M. Kimour. *Feature-oriented engineering of web applications*. Journal of Scientific and Engineering Research, 4(7):309-316. 2017.

- [Sottet, 2015] J-S. Sottet, A. Vagner, A. Frey. *Variability management supporting the model-driven design of user interfaces* 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD). 2015.
- [Sree-Kumar, 2018] A. Sree-Kumar, E. Planas, R. Clarisó. *Extracting software product line feature models from natural language specifications*. In Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1 (SPLC '18). Association for Computing Machinery, New York, NY, USA, 437-53. DOI:<https://doi.org/10.1145/3233027.3233029>. 2018.
- [Sridhar, 2020] C. Sridhar, N. Kesav V. *A family of software product lines in educational technologies* Computing (2020). <https://doi.org/10.1007/s00607-019-00772-x>. 2020.
- [Tan, 2013] L. Tan, Y. Lin, H. Ye. *An Aspect-Oriented Framework for Software Product Line Engineering*. 2013.
- [Texier, 2012] J. Texier, M. De Giusti, N. Oviedo, G. Villarreal, A. Lira. *The Benefits of Model-Driven Development in Institutional Repositories -Los Beneficios del Desarrollo Dirigido por Modelos en los Repositorios Institucionales*. Conference: II Conferencia Internacional Acceso Abierto, Comunicación Científica y Preservación Digital (BIREDIAL), At Colombia. 2012.
- [Tizzei, 2012] L. P. Tizzei, C. M. Rubira, J. A. Lee. *Feature-oriented Solution with Aspects for Component-based Software Product Line Architecting*. 38th Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Izmir, 2012, pp. 85-92. 2012.
- [Vallecillo, 2018] A. Vallecillo. *Model Driven Development*. Material de Master en Ingeniería del Software e Inteligencia Artificial. Universidad de Málaga. Dpto. Lenguajes y Ciencias de la Computación. 2018.
- [Valverde Giromé, 2010] F. Valverde Giromé. *OOWS 2.0: Un Método De Ingeniería Web Dirigido Por Modelos Para La Producción De Aplicaciones WEB 2.0*. Pastor López, O. dir., 2010.
- [Varela, 2011] P. Varela, J. Araújo, I. Brito, A. Moreira. *Aspect-Oriented Analysis for Software Product Lines Requirements Engineering*. In Proceedings of the 2011 ACM Symposium on Applied Computing, ACM, pp. 667- 674. 2011.
- [Vranić, 2016] V. Vranić, R. Táboršký, *Features as Transformations: A Generative Approach to Software Development*. Computer Science and Information Systems 13(3). pp 759-778, 2016.
- [Wang, 2009] B. Wang, W. Zhang, H. Zhao, Z. Jin, H. Mei, *A Use Case Based Approach to Feature Models' Construction*. In: Proceedings of 17th Intl. Conf. on Requirements Engineering (RE'09), IEEE, pp 121-130. 2009.
- [Weston, 2009] N. Weston, R. Chitchyan, A. Rashid, *A framework for constructing semantically composable feature models from natural language requirements*. In: Proceedings of the 13th International Software Product Line Conference (SPLC'09), pp.211-220. 2009.
- [White, 2009] J. White, B. Dougherty, D. Schmidt, D. Benavides. *Automated Reasoning for Multi-step Feature Model Configuration Problems*. Proceedings of the 13th International Software Product Line Conference, 2009.
- [Wolfart, 2019] D. Wolfart, W. K. Guez Assunção, J. Martinez. *Open Source Software on the Research of Extractive Adoption of Software Product Lines*. 16 Congresso Latino-americano de Software Livre e Tecnologias Abertas. Latinoware, 2019.
- [Zaki, 2017] S. M. Zaki, R. Mohamad, S. A. Halim, N. B. Ahmad. *Variability Management in Software Product Lines Online Learning Applications*. Journal of Telecommunication, Electronic and Computer Engineering, 2017.
- [Zhang, 2011] G. Zhang, H. Ye, Y. Lin. *Modelling Quality Attributes in Feature Models in Software Product Line Engineering*. In ICSOFT (2), pp. 249- 254. 2011.

-
- [Zhang, 2012] G. Zhang, X. Peng, Z. Xing, W. Zhao. *Cloning practices: Why developers clone and what can be changed*. In Software Maintenance (ICSM), 2012 28th IEEE International Conference on, pages 285-294. IEEE, 2012.
- [Ziadi, 2012] T. Ziadi, L. Frias, M. A. Almeida da Silva, M. Ziane. *Feature identification from the source code of product variants*. In Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering, CSMR'12, pages 417-422, Washington, DC, USA, IEEE Computer Society. 2012.