Títol: *Avaluació de la qualitat dels models navegacionals d'aplicacions web*

Volum: *1 de 1*

Alumne: *David García Giral*

Director/Ponent: *Cristina Gómez Seoane*

Departament: *LSI*

Data: *27/01/2010*

DADES DEL PROJECTE

Títol del Projecte: Avaluació de la qualitat dels models navegacionals d'aplicacions web

Nom de l'estudiant: David García Giral
Titulació: Enginyeria Tècnica en Informàtica de Gestió
Crèdits: 22,5
Director/Ponent: Cristina Gómez Seoane
Departament: LSI

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Enrique Mayol Sarroca

Vocal: Lluis Ametller Congost

Secretari: Cristina Gómez Seoane

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

Tabla de contenidos

Capítulo 1. Introducción 1

1.1 Motivación y objetivos 1
1.2 Situación actual 4
1.3 Descripción del proyecto 5
1.4 Metodología a utilizar 6
1.5 Análisis DAFO 8
1.5.1 Análisis interno 8
1.5.2 Análisis externo 9
1.6 Planificación 10
1.6.1 Planificación inicial 10
1.6.2 Tiempo utilizado para la realización del proyecto 12
1.6.3 Comparación entre la planificación y el tiempo utilizado 13
1.7 Coste del proyecto 14
1.7.1 Recursos humanos 14
1.7.2 Recursos materiales 14
1.7.3 Coste total 16

Capítulo 2. Análisis de requisitos 17

2.1 Introducción 17
2.2 Requisitos funcionales 18
2.2.1 Evaluación de la calidad del modelo navegacional 18
2.2.2 Visualización de las rutas navegacionales. 23
2.2.3 Visualización de los resultados de la evaluación cualitativa. 24
2.2.4 Importación de datos 24
2.2.5 Gestión de los proyectos 26
2.3 Requisitos no funcionales 27

Capítulo 3. Especificación 29

3.1 Introducción 29
3.2 Modelo de casos de uso 30
3.2.1 Diagrama de casos de uso 30
3.2.2 Especificación de los casos de uso 31
3.3 Modelo conceptual 42
3.4 Modelo de comportamiento 43
3.4.1 Caso de uso *Importar Modelos* 43
3.4.2 Caso de uso *Visualizar las rutas navegacionales* 43

3.4.3 Caso de uso *Evaluación Cualitativa del Modelo Navegacional* 44
3.4.4 Caso de uso *Nueva Regla de Dependencia* 44
3.4.5 Caso de uso *Editar Regla de Dependencia* 45
3.4.6 Caso de uso *Eliminar Regla de Dependencia* 46
3.4.7 Caso de uso *Nuevo Proyecto* 46
3.4.8 Caso de uso *Abrir Proyecto* 46
3.4.9 Caso de uso *Guardar Proyecto Actual* 47
3.4.10 Caso de uso *Cerrar Proyecto Actual* 47

Capítulo 4. Diseño 49

4.1 Introducción 49
4.2 Plataforma física 50
4.3 Arquitectura del sistema software 51
4.3.1 Modelo arquitectónico 51
4.3.2 Gestión de la persistencia 52
4.3.3 Tratamiento de errores 53
4.4 Diseño externo 55
4.4.1 Diseño de la pantalla principal 55
4.4.2 Diseño de las pantallas de los casos de uso 57
4.4.3 Diseño de los cuadros de diálogos genéricos 61
4.5 Diseño interno 62
4.5.1 Diseño de la capa de presentación 62
4.5.2 Diseño de la capa de dominio 65
4.5.3 Diseño de la capa de gestión de datos 72
4.5.4 Diagramas de secuencia 73

Capítulo 5. Implementación 91

5.1 Introducción 91
5.2 Lenguajes utilizados 91
5.2.1 Java 91
5.3 Librerías 92
5.3.1 Swing 92
5.3.2 JGraph 92
5.3.3 Lectura de XML 93
5.4 Ejemplos 93

Capítulo 6. Pruebas 99

Capítulo 7. Manual del usuario 101

7.1 Manual de instalación 102
7.1.1 Introducción 102
7.1.2 Requisitos del sistema 102
7.1.3 Contenido del CD-ROM 102
7.1.4 Instalación 103
7.1.5 Eliminar la instalación 106

7.2 Manual de uso 108
7.2.1 Introducción 108
7.2.2 Acceso al sistema 108
7.2.3 Cierre del sistema 109
7.2.4 Gestión de proyectos 109
7.2.5 Gestión de los modelos 113

Capítulo 8. Conclusiones 121

Capítulo 9. Trabajo futuro 123

Capítulo 10. Bibliografía 125

Capítulo 1. Introducción

1.1 Motivación y objetivos

Las aplicaciones web han adoptado un papel de gran importancia en los sistemas de la información de muchos sectores de negocio. Cada vez son más las empresas y organismos que utilizan estas herramientas para gestionar la información.

Como cualquier otro tipo de aplicaciones, en las webs se hace indispensable el uso de métodos a la hora de desarrollarlas, para aumentar la productividad, calidad e integridad, entre otros principios básicos de la ingeniería del software.

Inicialmente estos métodos estaban orientados a la construcción de aplicaciones que se limitaban a leer la información almacenada en las bases de datos. Éstos han ido evolucionando hasta cubrir los requerimientos de las aplicaciones web actuales, donde se incluye el procesado de los datos y las funcionalidades de manipulación de estos datos.

Como consecuencia, surge una necesidad de evaluar si el diseño realizado a partir de los modelos especificados (modelo de datos y modelo navegacional) cubre correctamente todas las necesidades o no, mediante los principios cualitativos que más adelante se explicarán.

Para entender las propiedades cualitativas de los modelos navegacionales se debe entender el concepto de modelo de datos y modelo navegacional.

Modelo de datos

El modelo de datos permite describir un esquema conceptual de los objetos que intervienen en el dominio, como son las entidades y las relaciones entre ellas.

En este modelo se ha incluido la definición de operaciones que permiten modificar el estado de la información. Estas operaciones básicas se encargan de insertar y eliminar instancias de entidades, así como modificar cualquier atributo de las mismas. Además también cubren la creación y eliminación de relaciones entre entidades.

Modelo navegacional

En el modelo navegacional se definen todas las posibles rutas que el usuario puede seguir dentro de la aplicación web. Estas rutas se especifican mediante nodos y enlaces inducidos del modelo de datos: los nodos representan ventanas lógicas (páginas) sobre las entidades, y los enlaces (links) derivan de las operaciones definidas en cada entidad.

La evolución en este modelo es evidente ya que es necesario definir nuevos caminos que permitan la modificación de los datos a petición del usuario. La inclusión de las operaciones de modificación en las entidades del modelo de datos es consecuencia de esta necesidad.

Calidad del modelo navegacional

La ampliación del modelo navegacional ha provocado que las propiedades cualitativas de este modelo y basadas en un análisis sintáctico de la estructura del modelo (por ejemplo, la comprobación de que todos los links tengan destino) no sean suficientes para asegurar un diseño correcto.

Según un reciente estudio (Cabot, Ceballos, Gómez, 2007), para evaluar la calidad de estos modelos surgen dos nuevas propiedades: la *completitud (completeness)* y la *correctitud (correctness)*

Completitud

La primera propiedad que se propone es la completitud del modelo navegacional respecto el modelo de datos. Se dice que el modelo de datos es completo si los usuarios de la web pueden todas las operaciones básicas de los elementos modificables correspondientes al modelo de datos. Un modelo navegacional es incompleto en las aplicaciones web que no permiten al usuario modificar alguna parte del contenido definido en el modelo de datos.

Correctitud

La segunda propiedad es la correctitud del modelo navegacional. Como se ha descrito anteriormente, en este modelo se definen todas las ejecuciones de las rutas posibles por parte del usuario que interactúa con la aplicación.

Durante la ejecución de cualquier ruta, se pueden ejecutar varias operaciones de modificación de datos. Es posible que al modificar los datos, las entidades afectadas tengan dependencias con otras definidas en el modelo de datos. Estas dependencias están preestablecidas para las operaciones más básicas como son

insertar o eliminar, pero también es posible que existan reglas de integridad que exijan otras dependencias. En el caso de que se incumpla alguna de las dependencias el usuario dejará la información inconsistente durante la ejecución de la ruta que incumple la dependencia.

Un modelo navegacional es correcto cuando todas las posibles rutas de navegación admiten por lo menos una ejecución que deja la información en un estado consistente, es decir, que no incumple las dependencias establecidas en el modelo de datos.

¿Cómo evaluar la calidad del modelo navegacional?

Acabamos de explicar qué propiedades deben tener los modelos navegacionales para que sean cualitativamente óptimos. En las aplicaciones con pocas entidades es relativamente fácil la comprobación del cumplimiento de estas propiedades sin ayuda de ninguna herramienta informatizada. Pero a medida que las aplicaciones crecen en número de entidades, se incrementa el tiempo invertido en la comprobación, además de correr el riesgo de omitir errores en la especificación.

La existencia de alguna herramienta informatizada capaz de evaluar las propiedades cualitativas eliminaría los errores cualitativos y reduciría notablemente el tiempo dedicado a la especificación. El actual proyecto responde a esta necesidad de mejora de esta fase. Por lo tanto el objetivo marcado es desarrollar una aplicación capaz de evaluar las propiedades descritas, y por consiguiente validar y detectar posibles errores en la especificación del modelo navegacional de cualquier aplicación web a partir de su modelo de datos.

¿Cómo definir el modelo navegacional y el modelo de datos?

Para poder evaluar el modelo navegacional, tendremos que definir el diagrama o esquema previamente en alguna aplicación que permita el modelado de este tipo de aplicaciones. La misma situación se presenta con el modelo de datos. Existe una herramienta orientada al diseño de aplicaciones web capaz de modelar estos esquemas, necesarios para la evaluación de la completitud y correctitud del modelo navegacional. La herramienta a la que estamos haciendo referencia es WebRatio, es una herramienta pensada para la generación de los diagramas basados en el lenguaje de modelado WebML (Web Modelling Language).

1.2 Situación actual

WebML está basado en el modelo impulsado por la ingeniería del software web. Éste está enfocado hacia el diseño de las aplicaciones a partir de un conjunto de herramientas y métodos que se abordan por separado utilizando diferentes modelos (conceptual, navegacional, presentación).

No obstante, existen más metodologías que se basan en el mismo modelo que WebML, como son UWE, HDM, OOHDM, OO-H o RMM. Cualquiera de estas metodologías define los diagramas de datos y navegacional, con lo cual en cualquiera de éstas se podrían aplicar las propiedades cualitativas del éste.

En cambio, el hecho de que existan diferentes metodologías no implica que exista tanta variedad de herramientas que permiten modelar los diagramas. Algunas de las pocas herramientas que existen para la definición de los modelos son:

VisualWADE, orientado al modelo OO-H (http://www.visualwade.com)

ArgoUWE y MagicUWE, plugins para ArgoUML y MagicDraw respectivamente, orientado al modelo UWE (http://www.pst.ifi.lmu.de/projekte/uwe/index.html)

WebRatio, orientado al modelo WebML (http://www.webratio.com)

Aun existiendo herramientas que permiten definir los esquemas conceptuales y la navegación, éstas no permiten evaluar las propiedades de completitud y correctitud de la navegación descritas en el documento (Cabot, Ceballos, Gómez, 2007).

En realidad, las propiedades definidas en el estudio son conceptos desconocidos hasta el momento de la publicación y por lo tanto es complicado que pueda existir una herramienta que evalúe estas propiedades.

1.3 Descripción del proyecto

Se va a desarrollar una aplicación que será capaz de evaluar las propiedades cualitativas de *correctitud* y *completitud* del modelo navegacional en las aplicaciones web que permiten al usuario interactuar con el sistema para modificar el contenido definido en el modelo de datos.

La propiedad de *correctitud* se evaluará a partir de una serie de reglas que deben cumplir los elementos implicados en los modelos. Además de las reglas que se definen en el estudio, existen las reglas de integridad propias de cada operación y que también pueden requerir dependencias con otras operaciones. La idea de evaluar la *correctitud* mediante reglas, impulsa la posibilidad de que el usuario pueda definir sus propias reglas para definir algunas reglas de integridad definidas en el modelo de datos.

Por otro lado, la herramienta a desarrollar inicialmente está orientada a la integración con los modelos definidos en la herramienta de modelado WebRatio, ya que además de tener una notación grafica bastante intuitiva, permite la exportación de todos los esquemas a ficheros XML, que serán los que se utilizarán como entrada de datos de la aplicación a desarrollar.

De cara a una posible ampliación, se evaluará la posibilidad de integrar esta aplicación con otras herramientas de modelado, como las citadas en el punto anterior. Esta integración se podría realizar de una manera explícita, donde el usuario podría definir esquemas que contengan los objetos característicos de los diagramas (clases y relaciones en el modelo conceptual; páginas, links, índices en el modelo de navegación) junto a la notación (XML) asignada en la herramienta de modelado. Por lo tanto es conveniente diseñar la aplicación pensando en esta ampliación.

1.4 Metodología a utilizar

Definir la metodología a utilizar en el desarrollo del software propuesto es un punto muy importante porque será la estrategia a seguir para alcanzar los objetivos marcados del proyecto.

Se ha decidido seguir el modelo tradicional de desarrollo de software, también conocido como modelo en cascada o ciclo de vida clásico, ya que la planificación es sencilla y permite trabajar individualmente.

Siguiendo este modelo, el desarrollo se realiza secuencialmente en diferentes fases muy bien diferenciadas entre sí, de forma que el inicio de cada una de las etapas no se produce hasta que no finaliza la anterior. Las **fases** que se siguen son las siguientes:

1. **Análisis de requisitos**. Se analizarán las necesidades del sistema, y se definirán los requisitos que debe tener, tanto los funcionales, propios del software (entradas/salidas de datos y procesos), como los no funcionales (aspectos sociales, conocimientos de los usuarios, etc.).

2. **Especificación**. Se detallará todo el comportamiento que debe tener el sistema definido en la fase anterior. Se especificarán las funciones y procesos requeridos con cierto detalle, independientemente de la tecnología que se utilizará para desarrollarlo.

3. **Diseño**. En esta fase se definirá con total detalle cómo se desarrollará el sistema para que realice las funcionalidades descritas en la fase de especificación

4. **Implementación**. A partir de la fase de diseño se ha de codificar el software. Previamente a la codificación es óptimo y aconsejable definir unas reglas o normas de codificación que se han de seguir para tener el código mejor estructurado y más comprensible.

5. **Pruebas**. Una vez finalizada la codificación se procederá a la realización de test que permitan detectar errores, para así proceder a la corrección de éstos. Se verificará que el sistema cumple los requisitos correctamente antes de ponerlo en producción. Si se detecta algún error se debe volver a la fase responsable del error y corregirlo en esa fase y en las posteriores si se ven afectadas por el cambio.

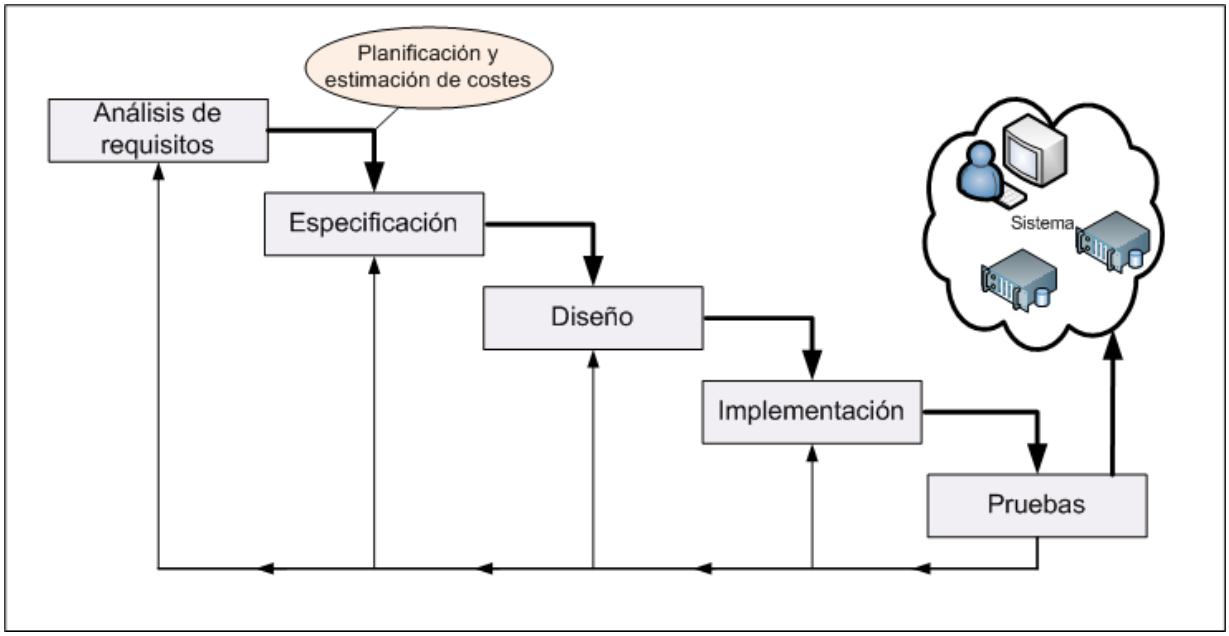


Fig. Ciclo de vida en cascada.

Además de las fases indicadas, una vez realizado el análisis de requisitos se procederá a realizar una planificación temporal y un estudio económico del proyecto.

El presente documento seguirá el orden establecido por la metodología, donde cada una de las fases originará un capítulo. No obstante, la planificación y el estudio económico estarán incluidos antes que el análisis de requisitos, en los puntos 1.6 y 1.7 respectivamente. Esta circunstancia no es significativa ya que no altera el orden cronológico de las fases. El estudio económico y la planificación se realizarán después de evaluar el alcance del sistema en el estudio del análisis de requisitos.

1.5 Análisis DAFO

El objetivo principal del análisis DAFO es determinar la situación real del sistema que se va a desarrollar en el proyecto actual, así como los riesgos y oportunidades que ofrece el mercado.

1.5.1 Análisis interno

El análisis interno estudia las fortalezas y debilidades del proyecto, es decir, todos los aspectos controlables tanto positivos como negativos para la elaboración del proyecto.

1.5.1.1 Fortalezas

Las fortalezas son aspectos (funcionalidades, recursos…) alcanzados por el proyecto y que potencian la competitividad, y por lo tanto pueden generar nuevas oportunidades.

- Integración de los modelos navegacionales y de datos de aplicaciones web definidos en la herramienta WebRatio. Se permiten importar los modelos al sistema para el posterior análisis.

- Capacidad de validar la correctitud y completitud del modelo navegacional.

- Posibilidad de visualizar los modelos. La visualización gráfica de los modelos facilita la visión del problema.

- Posibilidad de modificar las dependencias que definen la propiedad cualitativa de correctitud, mediante reglas configurables por el usuario.

- Bajo coste en el desarrollo. Se utiliza un entorno de desarrollo gratuito como Eclipse y un lenguaje de programación que no requiere licenciamiento. Además se utilizan librerías gratuitas como JGraph.

- Seguridad y fiabilidad en la ejecución. El desarrollo del programa se realiza en un entorno Java, y por lo tanto la ejecución del mismo la gestiona una máquina virtual.

- Es un software libre, donde cualquier usuario podrá ampliar y mejorar la aplicación.

1.5.1.2 Debilidades

Las debilidades hacen referencia a aquellos aspectos que pueden ser más vulnerables para la realización del proyecto. Son elementos o características del proyecto que limitan o reducen la capacidad de desarrollo. Deben ser controladas y superadas.

- Integración parcial de los componentes del modelo navegacional de la aplicación de WebRatio. La herramienta permite importar íntegramente el modelo de datos. En cambio, en el modelo navegacional tan sólo se importarán los elementos propios que intervienen en este modelo, es decir, páginas, links y operaciones.

- Integración exclusiva de WebRatio. Solamente se integran los modelos definidos en WebRatio, ignorando todas las demás herramientas de modelado de aplicaciones web.

1.5.2 Análisis externo

El análisis externo se compone de dos factores no controlables: oportunidades y amenazas.

1.5.2.1 Oportunidades

Las situaciones que se producen en el entorno, ventajosas para el proyecto en el ámbito competitivo y rentable son las siguientes:

- La herramienta que se construirá es nueva en el mercado, no existe otra que comparta características con ésta, por lo tanto no tiene competencia.

- El elevado crecimiento de aplicaciones web, especificadas siguiendo metodologías basadas en modelos navegacionales y de datos, por lo tanto puede aumentar la demanda de la herramienta.

1.5.2.2 Amenazas

A continuación se detallan las características externas y negativas que pueden impedir la realización del proyecto, reducir la efectividad, incrementar los riesgos o los recursos requeridos para el desarrollo del proyecto.

- Posible aparición de herramientas similares, y por consiguiente un posible aumento de la competencia y reducción de la demanda.

1.6 Planificación

1.6.1 Planificación inicial

El proyecto se realizará siguiendo la metodología indicada en el punto 1.4. Cada una de las fases depende de la anterior, es decir, hasta que no finalice una fase no se podrá pasar a la siguiente, exceptuando la fase de pruebas, que se realizará simultáneamente junto a la fase de implementación ya que se irán realizando pruebas a medida que se vaya implementando las diferentes funcionalidades definidas en la fase de especificación.

Además de las fases indicadas en la metodología, se tienen que tener en cuenta los trámites administrativos previos a la lectura. Se entregará un informe resumen de la descripción del proyecto 2 meses antes de la lectura. Alrededor de un mes antes de la lectura, se reservará el aula donde se realizará el acto. Asimismo, la memoria, junto con la fecha acordada por los miembros del tribunal, se debe presentar en secretaría como mínimo una semana antes de la lectura, donde sellarán las memorias y posteriormente se repartirán a los miembros del tribunal.

Se ha de tener en cuenta que la memoria se debe imprimir y encuadernar siguiendo la normativa marcada por la FIB. Esta tarea se asignará al CPED (*Centre de publicacions del Campus Nord*). El tiempo de producción está estimado entre 3 días y una semana y se debe entregar a los miembros del tribunal como mínimo una semana antes de realizar la lectura.

Por último, se procederá a la lectura la última semana permitida en el calendario establecido por la FIB.

A continuación se muestra la planificación de las tareas textualmente, indicando la duración en días (1 día = 8 horas) y las fechas de inicio y final de la tarea. Debajo, se muestra el diagrama de Gantt correspondiente a la descripción textual.

	Nombre de tarea	Duración	Comienzo	Fin	Prede
1	**- Analisis de requisitos**	**13,38 días?**	**sáb 25/10/08**	**sáb 15/11/08**	
2	Requisitos funcionales	12,38 días?	sáb 25/10/08	vie 14/11/08	
3	Requisitos no funcionales	0,5 días?	sáb 15/11/08	sáb 15/11/08	
4	**- Especificacion**	**8,63 días?**	**dom 16/11/08**	**dom 30/11/08**	**1**
5	Modelo de casos de uso	4,63 días?	dom 16/11/08	dom 23/11/08	
6	Modelo conceptual	4 días?	lun 24/11/08	dom 30/11/08	5
7	Modelo de comportamiento	4 días?	lun 24/11/08	dom 30/11/08	5
8	**- Diseño**	**13,13 días?**	**lun 01/12/08**	**dom 21/12/08**	**4**
9	Definir arquitectura sistema	1 día?	lun 01/12/08	mar 02/12/08	
10	Diseño externo	0,88 días?	mar 02/12/08	mié 03/12/08	
11	Diseño interno	11,75 días?	jue 04/12/08	dom 21/12/08	10
12	**- Implementación**	**13,13 días?**	**lun 22/12/08**	**dom 11/01/09**	**8**
13	Definir Reglas	0,5 días?	lun 22/12/08	lun 22/12/08	
14	Programacion	12,63 días?	mar 23/12/08	dom 11/01/09	13
15	Realización manual de usuario	2,63 días?	sáb 03/01/09	mié 07/01/09	
16	**- Pruebas**	**8,75 días?**	**mié 31/12/08**	**mar 13/01/09**	
17	Realización pruebas	8,75 días?	mié 31/12/08	mar 13/01/09	
18	Entrega informe previo tribunal	0,5 días?	lun 01/12/08	lun 01/12/08	
19	Reserva aula	0,38 días?	vie 02/01/09	vie 02/01/09	18
20	Encargo memoria CPED	0,5 días?	jue 08/01/09	jue 08/01/09	19;15
21	Recoger memoria CPED	0,38 días?	vie 16/01/09	vie 16/01/09	20
22	Entrega memoria y fecha lectura	0,5 días?	lun 19/01/09	lun 19/01/09	21;16
23	Lectura del PFC	0,5 días?	vie 30/01/09	vie 30/01/09	22

Planificación inicial de las tareas.

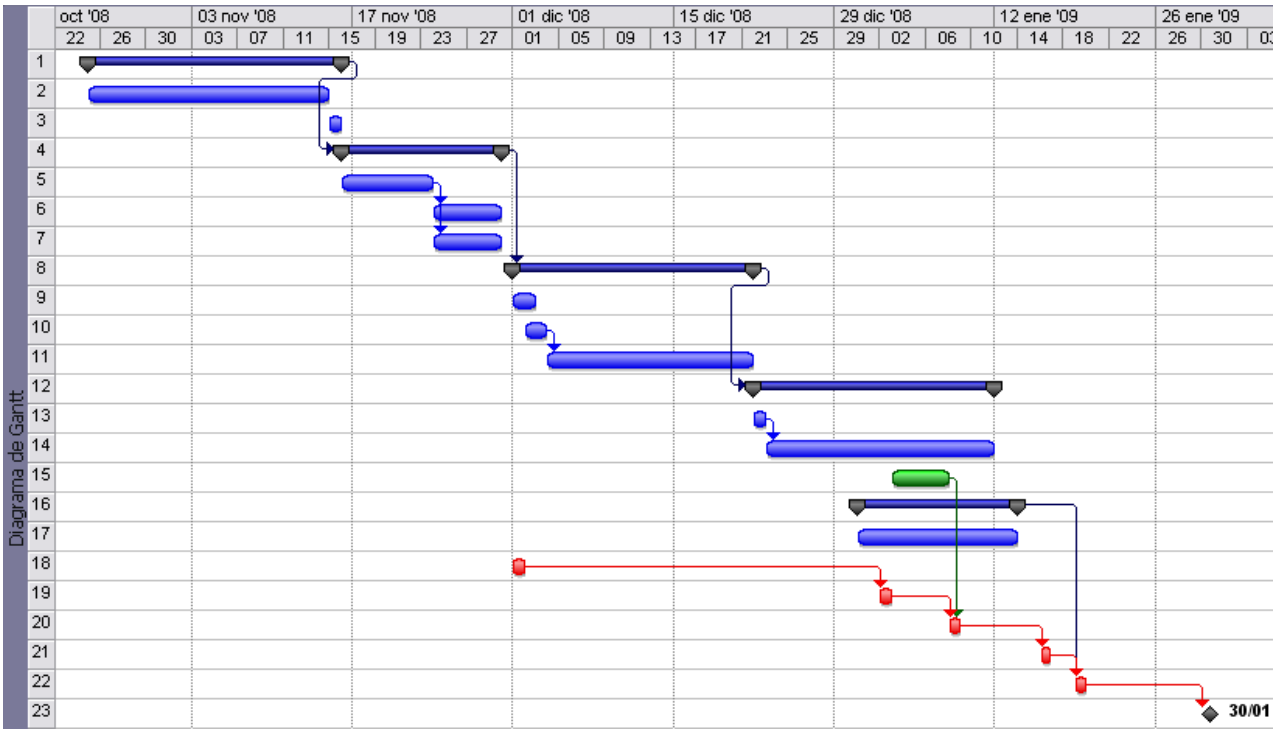


Diagrama de Gantt correspondiente a la planificación inicial

1.6.2 Tiempo utilizado para la realización del proyecto

	Nombre de tarea	Duración	Comienzo	Fin	Prede
1	**Analisis de requisitos**	**14 días**	**sáb 15/11/08**	**lun 08/12/08**	
2	Requisitos funcionales	13,5 días	sáb 15/11/08	dom 07/12/08	
3	Requisitos no funcionales	0,5 días	lun 08/12/08	lun 08/12/08	
4	**Especificacion**	**18,88 días**	**lun 23/02/09**	**vie 27/03/09**	**1**
5	Modelo de casos de uso	8,75 días	lun 23/02/09	lun 09/03/09	
6	Modelo conceptual	3,63 días	mar 10/03/09	dom 15/03/09	5
7	Modelo de comportamiento	6,5 días	lun 16/03/09	vie 27/03/09	5
8	**Diseño**	**21,75 días**	**lun 18/05/09**	**mié 24/06/09**	**4**
9	Definir arquitectura sistema	1 día	lun 18/05/09	mar 19/05/09	
10	Diseño externo	4,13 días	lun 25/05/09	dom 31/05/09	
11	Diseño interno	13 días	mar 02/06/09	mié 24/06/09	10
12	**Implementación**	**48,25 días**	**mar 13/10/09**	**vie 18/12/09**	**8**
13	Programacion	48,25 días	mar 13/10/09	vie 18/12/09	
14	**Pruebas**	**1,88 días**	**lun 21/12/09**	**mar 22/12/09**	**12**
15	Realización pruebas	1,88 días	lun 21/12/09	mar 22/12/09	
16	Realización del manual	2,13 días	lun 28/12/09	mié 30/12/09	14
17	Entrega informe previo tribunal	0,5 días	lun 22/06/09	lun 22/06/09	
18	Reserva aula	0,25 días	mié 23/12/09	mié 23/12/09	17
19	Encargo impresión memoria	0,25 días	lun 11/01/10	lun 11/01/10	16
20	Recoger memoria	0,15 días	vie 15/01/10	vie 15/01/10	19
21	Entrega memoria y fecha lectura	0,1 días	vie 15/01/10	vie 15/01/10	20;18
22	Lectura del PFC	0,25 días	mié 27/01/10	mié 27/01/10	21

Diagrama de Gantt

Planificación inicial de las tareas.

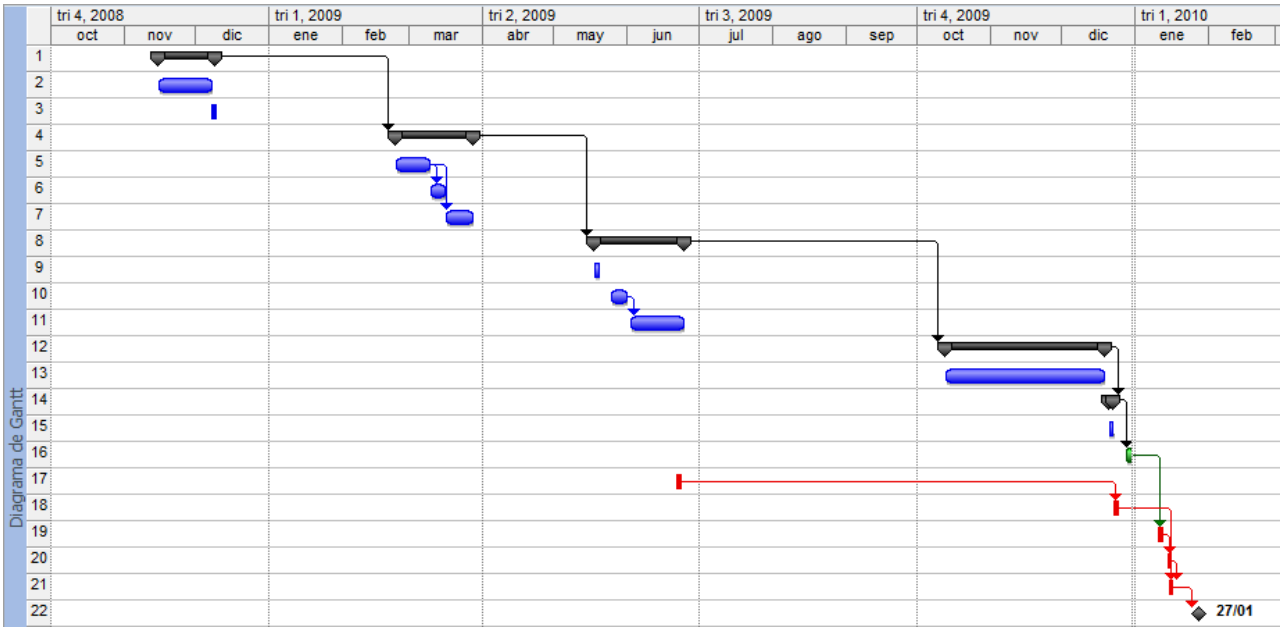


Diagrama de Gantt correspondiente a la planificación inicial

1.6.3 Comparación entre la planificación y el tiempo utilizado

Inicialmente se planificó la realización del proyecto en un periodo corto de tiempo para finalizarlo el siguiente cuatrimestre después de haber sido elegido, marcando unos tiempos muy optimistas. Por diferentes motivos externos al proyecto no se han podido cumplir los plazos y por lo tanto no se ha llegado a la planificación inicial.

Se han alargado los tiempos de cada fase, destacando el tiempo invertido en la fase de implementación y provocado por diferentes factores:

- Inicialmente no se conocía la librería JGraph, se ha requerido un aprendizaje que ha producido un retraso.

- La importación de los datos de la herramienta de WebRatio ha sido más costosa de lo inicialmente planificado, concretamente han surgido dificultades en la lectura de los ficheros XML. Por un lado se ha tenido que estudiar la estructura del XML, y por otro lado el desarrollo de la funcionalidad se ha aumentado debido a que se han probado diferentes librerías, hasta que se ha decidido utilizar la propia de J2SE 5.0 ya que se ajusta mejor a las necesidades.

- En general, la falta de práctica en el desarrollo de aplicaciones en lenguaje Java ha causado que el ritmo no haya sido el previsto inicialmente.

1.7 Coste del proyecto

A continuación se realizará la valoración económica del consumo de los recursos utilizados en el desarrollo del proyecto, los cuales se desglosan en recursos humanos y recursos materiales.

1.7.1 Recursos humanos

El desarrollo del proyecto actual será realizado por dos perfiles de profesionales principalmente:

Analista / Diseñador. Llevará a cabo la fase de análisis, especificación y diseño. El precio de cada hora de trabajo de este perfil profesional es de 70€.

Programador. Llevará a cabo la fase de programación y las pruebas. Realizará el manual de usuario. El precio es de 40€ / h.

A continuación se detalla el coste relativo a recursos humanos. Se ha de tener en cuenta que un día laborable comprende una jornada de 8 horas:

Roles		
Descripción Rol	**ID**	**Coste por hora**
Analista / Diseñador	AD	70,00 €
Programador	P	40,00 €

Coste RRHH				
Fase	**Rol**	**Días**	**Horas**	**Coste**
Analisis de requisitos	AD	14,00	112	7.840,00 €
Especificacion	AD	26,63	213	14.912,80 €
Diseño	AD	14,63	117	8.192,80 €
Implementación	P	20,63	165	6.601,60 €
Pruebas	P	1,88	15	601,60 €
Realización del manual	P	2,13	17	681,60 €
			Coste Total	**38.830,40 €**

1.7.2 Recursos materiales

Los recursos materiales utilizados para llevar a cabo el proyecto, y por lo tanto imputados en el coste del proyecto, son recursos de hardware y software.

1.7.2.1 Hardware

Únicamente se necesita un ordenador portátil donde se llevará a cabo la documentación, el desarrollo y las pruebas. Se elige un portátil debido al bajo coste, muy similar a un PC convencional, además de adquirir movilidad en caso necesario (viajes, muestra de prototipo de aplicación a clientes, etc.).

Costes de Hardware			
Producto	**Precio**	**#**	**Total**
PC portátil Compaq Presario serie CQ60-200 + Windows XP	449,00 €	1	449,00 €
		Coste Total	**449,00 €**

1.7.2.2 Software

La licencia correspondiente al sistema operativo Windows XP no se contempla explícitamente, ya que viene incluida en la compra del ordenador del PC, por lo tanto, el precio del PC ya contempla la licencia del SO.

Para realizar la documentación se utilizará el editor de textos Microsoft Word 2007, incluido en la suite Microsoft Office Standard 2007.

Para la implementación de los diagramas UML se utilizará la herramienta Office Visio Standard 2007. Se aprovechará esta herramienta para diseñar el prototipo de interfaz gráfica de usuario y algunos gráficos. Por otro lado, la planificación del proyecto y el seguimiento se llevará a cabo con la herramienta Office Project 2007.

Por último, no se debe olvidar que la herramienta está orientada al análisis de los datos generados por la herramienta WebRatio. Por lo tanto, se deberá instalar la herramienta, la cual permitirá el análisis de los datos generados y la realización de pruebas. Una licencia de WebRatio está valorada en 750€, aunque en el proyecto se utilizará una versión trial para el desarrollo de la herramienta, por lo tanto no tendrá ningún coste.

El coste del software será el siguiente:

Costes de Software				
Producto	**Precio**	**#**	**Total**	**Observaciones**
Microsoft Windows XP	0,00 €	1	0,00 €	Incluido en el PC
Microsoft Office Standard 2007	569,00 €	1	569,00 €	
Microsoft Visio Standard 2007	329,95 €	1	329,95 €	
Microsoft Project 2007	779,00 €	1	779,00 €	
WebRatio 5.0 trial	0,00 €	1	0,00 €	Trial de 3 meses
Eclipse	0,00 €	1	0,00 €	
		Coste Total	**1.108,95 €**	

1.7.2.3 Otros costes

La impresión y encuadernación de la documentación que se debe presentar al tribunal se debe realizar en el CPED (*Centre de publicacions del Campus Nord*). Los precios son los siguientes:

Costes de maquetación			
Producto	**Precio**	**#**	**Total**
Impresión B/N	0,04 €	324	12,96 €
Impresión Color	0,37 €	244	90,28 €
Encuadernación	3,10 €	4	12,40 €
		Coste Total	**115,64 €**

1.7.3 Coste total

Tipo de Recurso	Recurso	Coste
Recursos Humanos	Costes personal	38.830,40 €
Recursos Materiales	Costes Hardware	449,00 €
	Costes Software	1.108,95 €
	Otros costes materiales	115,64 €
	Total	**40.503,99 €**

Capítulo 2. Análisis de requisitos

2.1 Introducción

En esta fase se realizará un análisis de objetivos y necesidades del cliente. Se determinará un conjunto de características propias del sistema software que se va a desarrollar y que satisfaga estos requerimientos.

Se distinguen dos tipos de requisitos, los **funcionales** y los **no funcionales**.

Los requisitos funcionales describen las características o funcionalidades propias del software, las entradas y salidas de datos y procesos que debe haber.

Los requisitos no funcionales definen las cualidades generales que debe tener el sistema al realizar su función. No están relacionados con funciones del sistema y se refieren a varios factores, como la interfaz gráfica de usuario, documentación, consideraciones de hardware, características de calidad, económicas o políticas.

2.2 Requisitos funcionales

2.2.1 Evaluación de la calidad del modelo navegacional

El objetivo principal del sistema es determinar si el modelo navegacional de una aplicación web dada cumple las propiedades cualitativas de completitud y correctitud.

2.2.1.1 Evaluación de la completitud (Completeness)

Un modelo navegacional es completo cuando los usuarios de una aplicación web pueden realizar todas las operaciones básicas de los elementos modificables correspondientes al modelo de datos, interactuando con el conjunto de páginas. Por otro lado, es incompleto en las aplicaciones web que no permiten modificar parte del contenido modificable.

Es decir, el conjunto de operaciones que el modelo navegacional debe contener depende de las entidades definidas en el modelo de datos. Existen las siguientes operaciones básicas propias de las entidades:

- **InsertET**, permite insertar una nueva instancia de la entidad ET.
- **DeleteET**, permite eliminar una instancia existente de la entidad ET.
- **UpdateET**, actualiza un atributo de una instancia existente de la entidad ET.

Además de las entidades, en el modelo de datos se definen relaciones binarias entre entidades. Cada relación RT tiene un nombre y dos participantes, donde cada uno está formado por una entidad, una cardinalidad mínima y una cardinalidad máxima. Las operaciones básicas de las relaciones son:

- **InsertRT**, permite crear una relación RT entre dos entidades indicadas.
- **DeleteRT**, permite eliminar una relación RT existente entre dos entidades.

Un modelo navegacional es **completo** cuando todas las operaciones de modificación definidas en el modelo de datos se pueden llegar a ejecutar en el modelo navegacional N.

Adicionalmente, se dice que el modelo navegacional es **mínimo** cuando es completo y además cada una de las operaciones de modificación definidas en el modelo de datos está incluida una sola vez en el modelo navegacional, y por lo tanto, únicamente se puede ejecutar en un link.

La herramienta será capaz de evaluar si el modelo navegacional es completo. En el caso que no sea completo, el sistema informará al usuario de las operaciones que faltan para llegar a serlo. Si el modelo navegacional es completo, el sistema tiene que determinar si también es mínimo e informar al usuario del resultado de esta evaluación.

2.2.1.2 Evaluación de la correctitud (*Correctness*)

Un modelo navegacional describe las posibles rutas de navegación permitidas en una aplicación web, es decir, diferentes ejecuciones o caminos a seguir. Cada una de estas ejecuciones representa una situación de interacción entre el usuario y la aplicación. Durante una ejecución, varias operaciones de modificación pueden ser aplicadas a objetos o asociaciones definidas en el modelo de datos.

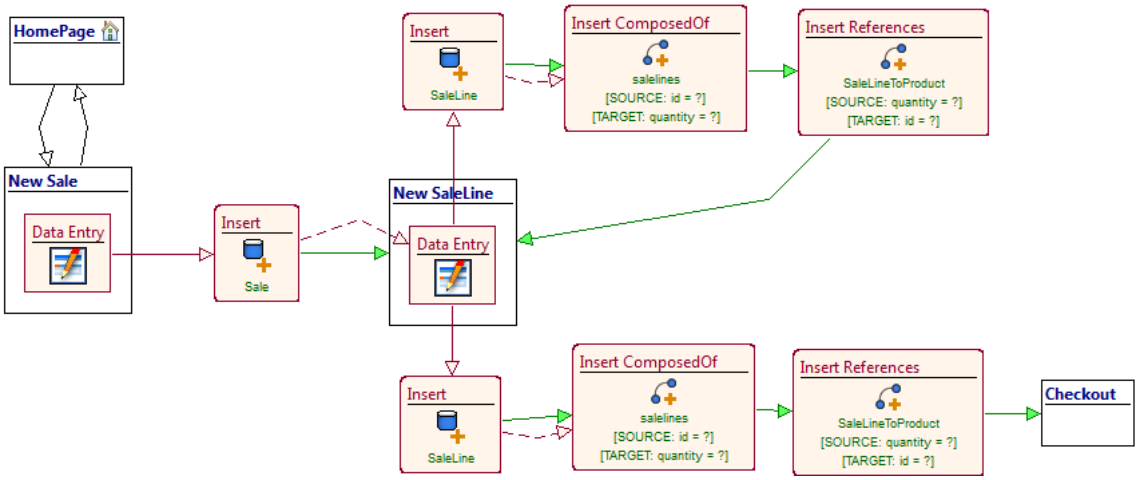


Fig. Parte del modelo navegacional de un e-commerce

Estas operaciones pueden dejar la información en un estado inconsistente cuando un usuario no introduce los valores apropiados en los formularios de las páginas visitadas durante la navegación. En este caso todos los cambios realizados durante la ejecución deben ser descartados. Es posible que todas las posibles ejecuciones de una navegación fallen.

Los modelos navegacionales correctos son aquellos que no contienen rutas de navegación que dejan la información en un estado inconsistente, independientemente de los valores que los usuarios suministran durante la interacción con las páginas. Por lo tanto, se puede decir que un modelo navegacional es correcto si todas las posibles rutas de navegación son correctas.

Para determinar la correctitud, el sistema tiene que resolver los siguientes puntos:

a) Definición de las rutas navegacionales de un modelo navegacional.

El sistema tiene que determinar el conjunto de posibles rutas navegacionales en del modelo navegacional. Suponiendo que el modelo navegacional se representa mediante un grafo, el conjunto de rutas navegacionales corresponde al conjunto de rutas del grafo que no incluyen arcos repetidos.

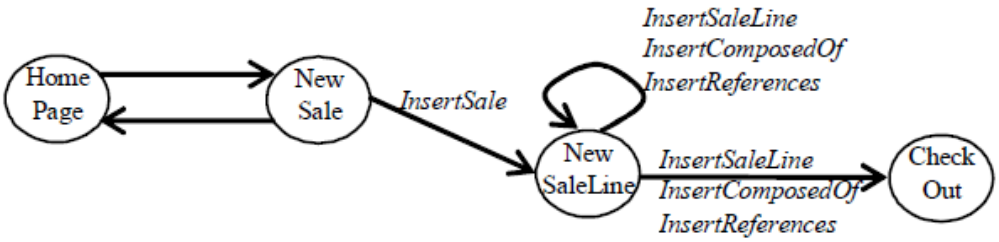


Fig. Representación del modelo navegacional mediante un grafo.

El grafo está compuesto por un conjunto de vértices y aristas. Los vértices o nodos representan las páginas, y las aristas representan los links que permiten la navegación entre las páginas.

Además, en el modelo navegacional se ejecutan las operaciones que modifican el estado del modelo de datos durante la ejecución de un link. Cada una de estas operaciones está definida en la etiqueta del arco y ordenada por la secuencia de ejecución.

b) Correctitud de una ruta navegacional.

La correctitud de una ruta navegacional depende de las operaciones asociadas a los arcos contenidos en la ruta. Algunas operaciones requieren la presencia de otras en un arco precedente o posterior dentro de la ruta actual, para dejar la información en un estado consistente. Decimos que una operación op_1 depende de op_2 cuando op_1 requiere la presencia de op_2 en la misma ruta.

Las dependencias de una operación dependen del tipo de operación (insert, update, delete) y de las reglas de integridad definidas en el modelo de datos. Una ruta navegacional deberá satisfacer todas las dependencias de todas las operaciones incluidas en ella.

Una ruta navegacional es correcta cuando para cada operación del conjunto de operaciones que se ejecutan en la ruta, se satisfacen correctamente todas las dependencias de ésta.

Para definir las dependencias de una operación se utilizan los siguientes criterios:

- Operación requerida de la entidad o relación dependiente.
- Número de ejecuciones de la operación requerida.
- Dirección. Indica si la operación requerida debe ser ejecutada antes de la operación que se está evaluando (←), después (→) o si el orden es independiente (↑)

Normalmente una operación está compuesta por varias reglas atómicas, definidas con los criterios que se acaban de exponer. En este caso, la dependencia quedará definida mediante una secuencia anidada de dependencias atómicas interconectadas con operadores AND y/o OR.

Para el tipo de operaciones básicas tanto de entidades (InsertET, DeleteET) como de relaciones (InsertRT, DeleteRT) existen las siguientes dependencias predefinidas:

Dependencias de las entidades

Dada una entidad *ET* y una operación *op* definida en *ET, Dep_{op}* se calcula de la siguiente manera:

- Si *op=InsertET*, existen las dependencias:
  - dep_{RT} = < →, InsertRT, min(ET, RT) > para cada RT donde min(ET,RT)≥1.
  - En el caso que ET sea un supertipo de la generalización Gens(ET, ET_1,…ET_n) existe la dependencia $dep_{GensSup}$= < →, $InsertET_i$, 1 > para cada entidad ET_i.
  - En el caso que sea un subtipo de una generalización disjoint y completa Gens(ET', ET, …), es necesaria la dependencia $dep_{GensSub}$= < ←, InsertET', 1 >

    Dep_{op} es la unión de las dependencias dep_{RT},$dep_{GensSup}$ y $dep_{GensSub}$

- Si *op=DeleteET*, existen las dependencias:

- dep_{RT} = < ←, DeleteRT, min(ET, RT) > para cada RT donde min(ET,RT)≥1.

- En el caso que ET sea un supertipo de la generalización Gens(ET, ET_1,…ET_n) existe la dependencia $dep_{GensSup}$= < ←, $DeleteET_i$, 1 > para cada entidad ET_i.

- En el caso que sea un subtipo de una generalización disjoint y completa Gens(ET', ET, …), es necesaria la dependencia $dep_{GensSub}$= < →, DeleteET', 1 >

Dep_{op} es la unión de las dependencias dep_{RT} , $dep_{GensSup}$ y $dep_{GensSub}$. **{Dep_{op} = dep_{RT} AND $dep_{GensSup}$ AND $dep_{GensSub}$}**

Dependencias de asociaciones

Dada una asociación RT con dos participantes ET_1 y ET_2, y una operación *op* definida en *RT*, Dep_{op} se calcula de la siguiente manera:

- Si *op=InsertRT*, existen las dependencias:

  - dep_{RT} = < ↑, DeleteRT, 1) > para un solo participante ET_i si min(ET_i,RT) = max(ET_i,RT).

  - dep_{ET} = < ←, $InsertET_i$, 1) > para cada uno de los participantes ET_i en el caso que min(ET_i,RT) = max(ET_i,RT)≥1

    Dep_{op} es la conjunción de las dependencias dep_{RT} y dep_{ET}.

    { Dep_{op} = dep_{RT} OR dep_{ET} }

- Si *op=DeleteRT*, existen las dependencias:

  - dep_{RT} = < ↑, InsertRT, 1) > para un solo participante ET_i si min(ET_i,RT) = max(ET_i,RT).

  - dep_{ET} = < ←, $DeleteET_i$, 1) > para cada uno de los participantes ET_i en el caso que min(ET_i,RT) = max(ET_i,RT)≥1

    Dep_{op} es la conjunción de las dependencias dep_{RT} y dep_{ET}.

    { Dep_{op} = dep_{RT} OR dep_{ET} }

Además de las dependencias de las operaciones básicas que se acaban de indicar, pueden existir otras dependencias que surgen sin basarse en ninguna regla genérica, por ejemplo, debido a restricciones textuales de integridad del modelo de datos. Estas reglas también se deberán cumplir para que el modelo navegacional llegue a ser correcto.

2.2.1.3 Gestión de las reglas de dependencia

El sistema permitirá la gestión de reglas de dependencia para una correcta validación de la correctitud del modelo navegacional.

Al contrario que las dependencias genéricas establecidas por la definición de correctitud y que se tienen que cumplir en todos los modelos navegacionales, las dependencias definidas por el usuario serán reglas que afectarán a los modelos individualmente. Es decir, las reglas que el usuario defina sobre un modelo de datos, y más concretamente sobre una operación de él, únicamente se podrán aplicar sobre el modelo navegacional correspondiente.

Las reglas de dependencia se definirán bajo los mismos principios que dependencias derivadas de la correctitud. El usuario tiene que seleccionar la operación sobre la que se aplica la dependencia, además de indicar los elementos que forman la regla, es decir, la operación de la que depende, el número de repeticiones y el orden de ejecución (antes, después o indiferente). El diseñador podrá definir reglas más complejas mediante operadores AND y OR entre las reglas atómicas.

Es decir, el usuario podrá definir nuevas reglas en el sistema, y además podrá editar y eliminar las reglas existentes no típicas de la correctitud.

2.2.2 Visualización de las rutas navegacionales.

La simplicidad de los datos cargados en el sistema y al mismo tiempo el ofrecer una visión enfocada al problema es una característica importante para facilitar al diseñador su comprensión. Esto ayudará a detectar con más precisión el origen de los errores y como consecuencia se tomarán decisiones más apropiadas que alterarán la definición de los modelos a partir de los resultados.

Para ayudar a la comprensión de la información, se utilizará una representación gráfica de los datos, de una manera similar a como lo hacen las herramientas de modelado, aunque más simplificada ya que el sistema únicamente necesitará una parte muy reducida de los elementos que se pueden definir en estas herramientas.

Pero esta visualización estará orientada a la representación de las rutas navegacionales que componen el modelo navegacional, mostrando las páginas y operaciones ejecutadas en los links entre éstas. Esto facilitará la comprensión de la evaluación de la correctitud.

2.2.3 Visualización de los resultados de la evaluación cualitativa.

Como se indica en el punto anterior, la representación gráfica de los modelos cargados en el sistema ayudará a comprender los datos que manipula la herramienta. Siguiendo este criterio, el sistema mostrará los resultados de la evaluación cualitativa basándose en la representación de los modelos cargados.

El sistema mostrará un mensaje textual informando si el modelo N es completo, mínimo y correcto:

En el caso de no ser completo, informará de las operaciones definidas en el modelo de datos no incluidas en el modelo navegacional.

Si el modelo navegacional no cumple la propiedad de correctitud, el sistema informará de esta situación y de las rutas navegacionales que no son correctas.

Para facilitar la comprensión de las reglas que fallan en la correctitud, el sistema cargará las rutas navegacionales y añadirá un indicador en cada operación que falle, el cual indicará la regla que ha fallado.

2.2.4 Importación de datos

En las aplicaciones web el diseñador define el modelo de datos y el modelo navegacional mediante herramientas de modelado web, como *WebRatio*, que utilizan el lenguaje WebML y son almacenados en ficheros con sintaxis XML. Serán estos modelos los que utilizará el sistema para evaluar las propiedades cualitativas, por lo tanto, el sistema utilizará los ficheros XML generados como entrada de datos.

El diseñador podrá llevar a cabo esta tarea, mediante una importación que adapte los datos generados por el sistema de modelado externo en el sistema. Es decir, el sistema será capaz de traer los datos de otras aplicaciones, en este caso WebRatio, y adaptarlos para el posterior análisis. Estos datos son el ***modelo de datos*** y el ***modelo navegacional***. Dentro del ***modelo de datos*** se trabajará con **entidades**, **relaciones** y **generalizaciones**. En el ***modelo navegacional*** lo que es relevante para llevar a cabo el análisis de las propiedades cualitativas que se describirán más adelante son las **páginas**, los **links** y las **operaciones** ejecutadas en cada link.

Los diagramas definidos en WebRatio están ubicados en la carpeta del proyecto correspondiente, y normalmente están almacenados en el fichero **Model.wr** en lenguaje XML. Este fichero será una entrada de datos en la aplicación y tiene una estructura definida, en la cual existen elementos y a su vez atributos dentro de estos elementos relevantes para el sistema. De todos modos, el sistema no debe ser flexible y permitir seleccionar cualquier fichero que contenga un esquema WebML correcto.

El sistema tendrá que leer los elementos correspondientes al modelo de datos y al modelo navegacional, definidos en XML:

- Modelo de datos (*DataModel*): El modelo de datos contiene entidades y relaciones:
    - Entidades (*Entity*): Contiene atributos, y puede estar relacionado con una *SuperEntity.*
    - Relaciones (*Relationship*)**:** Define una relación entre dos entidades. La entidad donde se inicia la relación está definida en el atributo *SourceEntity*, y la entidad en la que finaliza está en el atributo *targetEntity*. Además, está formado por un elemento *RelationshipRole1* y un elemento *RelationshipRole2.* Estos dos elementos tienen la misma definición, donde el atributo *maxCard* contiene el valor de la cardinalidad máxima del rol.
- Modelo navegacional (*WebModel*): El modelo navegacional definido en WebRatio está compuesto por diferentes vistas *(SiteView).* Estas vistas implementan diferentes rutas navegacionales en un contexto determinado. *SiteView* contiene la información que necesitamos para realizar el análisis: páginas, links, y operaciones
    - Páginas (*Page*). Elementos de la interfaz del usuario, quién a través de links puede recorrerlos en la secuencia deseada. Una página típica está compuesta de unidades que agrupadas dan un significado informativo. Una página contiene elementos atómicos que son utilizados para especificar la composición de ésta, aunque no intervienen en las propiedades cualitativas que se van a evaluar.
    - Hipervínculo (*Link*). Los links permiten conectar o enlazar elementos entre sí, como páginas, y operaciones. Es decir, permiten especificar los posibles caminos que puede seguir un sitio web.

      KOLink y *OKLink* son un tipo de links exclusivos de las operaciones, que se diferenciarán de un link normal en que su ejecución depende de

la correcta ejecución de la operación. Si una operación se ejecuta sin errores, se seguirá *OKLink*. En cambio, si existe algún error en la ejecución, se seguirá *KOLink*

- Operaciones. Las operaciones son elementos distribuidos fuera de las páginas y que por lo tanto se ejecutan fuera de éstas. Cada operación puede mantener una vinculación entre otra operación o una página. Las operaciones que nos interesan son *CreateUnit*, *DeleteUnit* y *ModifyUnit,* que modifican entidades, y los elementos que modifican relaciones *ConnectUnit* y *DisconnecUnit.*

2.2.5 Gestión de los proyectos

La importación de los modelos, la gestión de plantillas, las reglas de dependencia y la propia evaluación del modelo navegacional son funcionalidades que cambian el estado de los datos cargados en el sistema. Es importante que estos datos, relacionados entre sí, se puedan guardar en cualquier momento para evitar perdida del trabajo o repeticiones de los mismos procesos con los mismos datos. Por lo tanto, el sistema deberá poder guardar los datos en el estado en el que se encuentran en el momento de guardarlos.

El conjunto de los datos que maneja la aplicación se denominará **proyecto de evaluación cualitativa del modelo navegacional**, y en el momento de guardarlo el usuario podrá definir el nombre y la ubicación en el disco del proyecto.

Guardar los datos no serviría de nada si no se pudiese cargar la información de nuevo en el sistema para volver a utilizarla. Es por eso que el sistema tiene que permitir **abrir proyectos de evaluación cualitativa del modelo navegacional**, es decir, el sistema podrá cargar los datos en el estado que fueron guardados para su manejo.

Además de abrir y guardar proyectos, el sistema deberá tener un cierre controlado del proyecto, controlando si existen cambios no guardados antes del cierre, y ofreciendo la posibilidad de guardar o descartar los cambios realizados antes de liberar los datos cargados en ese momento, momento en el que dejarán de ser accesibles y manejables.

Por último, el usuario podrá **crear proyectos nuevos**, donde se manejarán los datos aunque inicialmente no habrá ningún dato cargado.

2.3 Requisitos no funcionales

- La herramienta a desarrollar está orientada a analistas/diseñadores de aplicaciones web, es decir, personas que realizan el diseño de estos sistemas.

- Los orígenes de datos del sistema, es decir, los modelos que se van a analizar, se deberán definir utilizando la herramienta WebRatio, de la versión 4 hasta la versión 5.1.0.

- La herramienta será funcional en un entorno Windows (98, Me, 2000, XP, Vista, 7) de 32bits, sin garantizar el funcionamiento en sistemas x64.

- La aplicación se desarrollará en un entorno Java. Por lo tanto, para que funcione se requiere la instalación de la máquina virtual Java, incluida en la plataforma J2SE 5.0 (o JDK1.5) o superiores.

- La interfaz del software estará implementada en **inglés**, es decir, todos los textos de menús, ventanas y demás elementos visibles en la interfaz gráfica de usuario estarán expresados en esta lengua.

- En cuanto a requerimientos de hardware, la aplicación funcionará en cualquier máquina que funcione con los sistemas operativos anteriormente citados, sin ser necesaria una conexión a internet.

- Pantalla con un mínimo de resolución de 1024 x 768 pixeles y 256 colores.

- 3MB de espacio en disco requerido.

- 32MB de memoria.

- Además del software, se facilitará un manual de usuario. El manual de usuario contendrá un manual de uso, donde se definirán con detalle todas las funcionalidades del sistema, y un manual de instalación, que detallará el proceso de instalación del sistema.

Capítulo 3. Especificación

3.1 Introducción

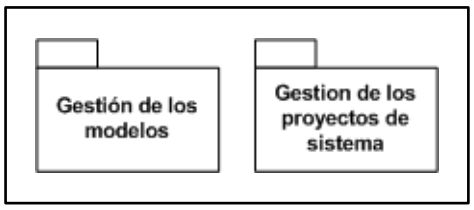
En esta fase se detallará el comportamiento del sistema definido en el análisis de requisitos. Se especificarán las funciones y procesos requeridos con cierto detalle, independientemente de la tecnología que se utilizará para desarrollarlo. La especificación se lleva a cabo desarrollando los siguientes modelos:

- Modelo de casos de uso. Recoge las funcionalidades del sistema y la interacción del usuario con el sistema. Está formado por las siguientes partes:
    - Diagrama de casos de uso. Recoge gráficamente las funcionalidades del sistema y los actores participantes en ellas.
    - Especificación de casos de uso. Definición del comportamiento del sistema en cada funcionalidad e interacción con el usuario. Se definirá la interacción básica y las posibles alternativas.
- Modelo conceptual. Representa las entidades que forman parte del dominio y la asociación existente entre ellas mediante relaciones.
- Modelo de comportamiento. Recoge con mayor rigor cada uno de los escenarios de interacción actor-sistema para cada caso de uso, indicando las entidades participantes. El modelo de comportamiento está formado por:
    - Diagramas de secuencia del sistema. Definen el modelo de comportamiento mediante una notación gráfica. Para cada caso de uso, se definen tantos diagramas como escenarios posibles tenga.
    - Contratos de las operaciones del sistema. Describen la entrada y salida que tiene la operación, así como el cambio de estado que sufren las entidades. Para cada operación de sistema que participe en el diagrama de secuencia se definirá esto.

3.2 Modelo de casos de uso

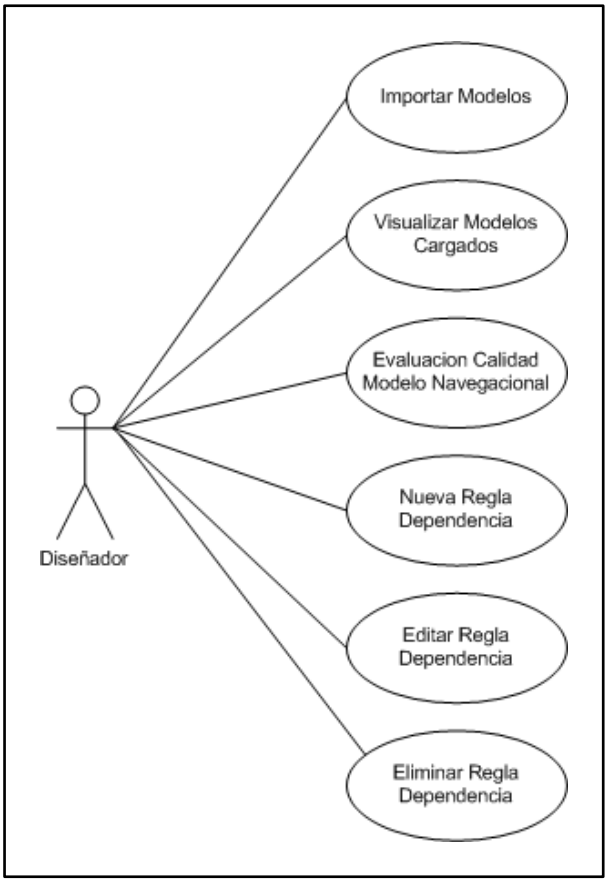
3.2.1 Diagrama de casos de uso

El sistema que se construirá se divide en 2 subsistemas principales:



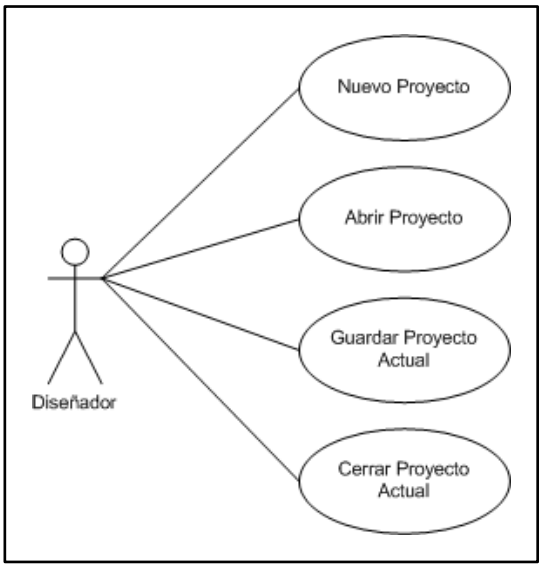
Gestión de los modelos

Los casos de uso que aparecen en este subsistema son los relacionados a las funcionalidades que se realizan en torno a los modelos cargados en el sistema, como son la importación de éstos al sistema, la representación gráfica de las rutas navegacionales y la evaluación del modelo navegacional. Además, también se incluye la gestión (Alta, Modificación, Baja) de las dependencias de las operaciones del modelo de datos necesarias para la correcta evaluación del navegacional.



Gestión de proyectos del sistema.

Los casos de uso incluidos en la gestión de proyectos del sistema son los referentes a la gestión de la persistencia de los datos que utiliza la herramienta.



3.2.2 Especificación de los casos de uso

Para cada caso de uso que aparece en los diagramas, se especificará la siguiente información:

- Nombre
- Descripción
- Actores
- Precondiciones
- Postcondiciones
- Curso principal
- Cursos alternativos

3.2.2.1 Caso de uso *Importar Modelos*

Descripción: Se importan los modelos de datos y navegacional definidos en sistemas de modelado externos.

Actores: Diseñador

Pre-condiciones:

1. Se ha abierto un proyecto de sistema.

Post-condiciones:

Se han importado los modelos definidos en sistema de modelado externos al proyecto abierto del sistema

Curso principal:

1. El usuario inicia la importación.
2. El sistema solicita la selección de un origen del modelo de datos
3. El diseñador selecciona el origen del modelo de datos
4. El sistema solicita la selección del origen del modelo navegacional.
5. El diseñador selecciona el origen del modelo navegacional
6. El sistema importa los modelos seleccionados en el proyecto actual e informa del éxito al diseñador

Cursos alternativos:

***) Si el diseñador cancela la importación**

1. Finaliza el caso de uso

***) Si el sistema no reconoce ningún sistema de ficheros definido en el SO, y por lo tanto no puede seleccionar ningún origen de datos**

1. El sistema informa del error y finaliza el caso de uso

3,5) Si el diseñador no selecciona ningún origen

1. El sistema informa que no se importará el modelo que se está solicitando y vuelve al paso anterior (2 o 4)

3,5) Si el diseñador selecciona un origen inconsistente, vacío o que no contiene el modelo que se ha de importar

1. El sistema informa que no se importará el modelo que se está solicitando y vuelve al paso anterior (2 o 4)

3,5) Si el diseñador selecciona un origen correcto pero ya existe un modelo de datos cargado actualmente en el sistema.

1. El sistema informa al diseñador y solicita si desea o no reemplazarlo o volver a

seleccionar un origen distinto 2. El diseñador selecciona una opción 1) Si el diseñador acepta el reemplazo: 1. El sistema continúa en el siguiente paso del curso principal (4 o 6) manteniendo el origen seleccionado en el paso actual 2) Si el diseñador anula el reemplazo: 1. El sistema continúa en el siguiente paso del curso principal (4 o 6) sin ningún origen seleccionado, por lo tanto mantendrá el modelo cargado en el sistema 3) El diseñador solicita volver a seleccionar el origen del modelo. 1. El sistema salta al paso anterior del escenario principal (2 o 4)

3.2.2.2 Caso de uso *Visualizar las rutas navegacionales*

Descripción: Se representan gráficamente las rutas navegacionales del modelo navegacional cargado en el sistema.
Actores: Diseñador
Pre-condiciones: Se ha cargado un proyecto del sistema que contiene el modelo de datos y el modelo navegacional.
Post-condiciones: El sistema reproduce gráficamente las rutas navegacionales del modelo navegacional cargado en el proyecto.
Curso principal: 1. El diseñador solicita visualizar las rutas navegacionales. 2. El sistema reproduce el diagrama seleccionado.

3.2.2.3 Caso de uso *Evaluación Cualitativa del Modelo Navegacional*

Descripción: El sistema evalúa la calidad del modelo navegacional en base a las propiedades de completitud y correctitud. Por lo tanto, el caso de uso se divide en dos fases: la evaluación de la completitud y la evaluación de la correctitud. Primero evalúa la completitud, comprobando que en el modelo navegacional se

pueden ejecutar todas las operaciones básicas de modificación de cada una de las entidades definidas en el modelo de datos. Adicionalmente en esta fase, y en el caso que el modelo navegacional sea correcto, el sistema comprueba si es mínimo, es decir, que cada una de las operaciones del modelo de datos se ejecuta una única vez. Una vez ha finalizado la evaluación de la completitud satisfactoriamente el sistema procede a evaluar la correctitud, determinando las posibles rutas navegacionales del modelo navegacional y evaluando individualmente, mediante las reglas de dependencia, cada una de las operaciones que componen la secuencia de operaciones de cada una de las rutas navegacionales determinadas por el sistema.
Actores: Diseñador
Pre-condiciones: Se ha cargado un proyecto del sistema que contiene el modelo de datos y el modelo navegacional.
Post-condiciones: El sistema determina si el modelo navegacional es *completo*, *mínimo* y *correcto*.
Curso principal: 1. El diseñador solicita evaluar la calidad del modelo navegacional cargado en el sistema. 2. El sistema evalúa si el modelo navegacional es completo, mínimo y correcto e informa al usuario del resultado de la evaluación.

3.2.2.4 Caso de uso *Nueva Regla de Dependencia*

Descripción: El sistema crea una nueva regla de dependencia asociada a una operación del modelo de datos cargado en el sistema.
Actores: Diseñador
Pre-condiciones: Se ha cargado un proyecto del sistema que contiene el modelo de datos y el modelo navegacional.
Post-condiciones: El sistema crea una regla ***r*** asociada a la operación **op** y que depende de la operación **reqop**, que se deberá ejecutar **reqnum** veces y en el momento indicado por **reqdir**

para considerar la operación **op** correcta. Esta regla se relaciona con la regla **ruleRel** mediante el operador **operator** (AND/OR), formando una regla lógica entre dos reglas de **op**.

Curso principal:

1. El usuario inicia la creación de la nueva regla de dependencia
2. El sistema carga los constructores del modelo de datos (entidades y relaciones) y solicita seleccionar uno de ellos.
3. El usuario selecciona la entidad.
4. El sistema carga las operaciones del constructor seleccionado y solicita la operación a la cual se le asignará la regla.
5. El usuario selecciona la operación **op** y continúa.
6. El sistema carga los constructores del modelo de datos (entidades y relaciones) y solicita seleccionar uno de ellos que formará la dependencia de la regla.
7. El usuario selecciona el constructor.
8. El sistema carga las operaciones del constructor seleccionado y solicita la operación que formará la dependencia (**reqop**).
9. El usuario selecciona la operación **reqop**.
10. El sistema solicita introducir el número de ejecuciones de la operación requeridas por la regla (**reqnum**).
11. El usuario introduce un número de ejecuciones **reqnum**.
12. El sistema solicita el orden de la ejecución de las operaciones (**reqdir***: {BEFORE, UNDEFINED, AFTER}*)
13. El usuario introduce el orden de la ejecución **reqdir**.
14. El sistema carga todas las reglas asociadas a la operación **op** y solicita al usuario la selección de una ellas para relacionar la nueva regla con ésta (**ruleRel**).
15. El usuario selecciona la regla existente **ruleRel**.
16. El sistema carga los operadores lógicos AND y OR y solicita la selección de uno para poder relacionar la nueva regla creada y la seleccionada en el punto 15.
17. El usuario selecciona un operador (**operator)** y confirma el alta de la regla.
18. El sistema crea una regla ***r*** asociada a **op** y que depende de la operación **reqop**, que se deberá ejecutar **reqnum** veces y en el momento indicado por **reqdir** para considerar la operación **op** correcta. Esta regla se relaciona con la regla **ruleRel** mediante el operador **operator** (AND/OR), formando una regla lógica entra dos reglas de **op**.

Cursos alternativos:

17) El usuario no ha introducido algún valor de los solicitados por el sistema en los puntos 3, 5, 7, 9, 11, 13, 15, 17.

1. El sistema informa del error y finaliza el caso de uso.

17) El usuario ha seleccionado una operación en la dependencia que se refiere a la propia operación de la regla.

1. El sistema informa del error y finaliza el caso de uso.

***) Si el usuario cancela el caso de uso**

1. Finaliza el caso de uso.

3.2.2.5 Caso de uso *Editar Regla Dependencia*

Descripción: Se modifica una regla existente asociada a una operación del modelo de datos cargado en el sistema.

Actores: Diseñador

Pre-condiciones:

Se ha cargado un proyecto del sistema que contiene el modelo de datos y el modelo navegacional.

Post-condiciones:

El sistema edita la regla existente **rule** asociada a la operación **op** y que depende de la operación **reqop**, que se deberá ejecutar **reqnum** veces y en el momento indicado por **reqdir** para considerar la operación **op** correcta. Esta regla se relaciona con la regla **ruleRel** mediante el operador **operator** (AND/OR), formando una regla lógica entra dos reglas de **op**.

Curso principal:

1. El usuario inicia el caso de uso
2. El sistema carga los constructores del modelo de datos (entidades y relaciones) y solicita seleccionar uno de ellos.
3. El usuario selecciona la entidad.
4. El sistema carga las operaciones del constructor seleccionado y solicita la operación a la cual se le asignará la regla.
5. El usuario selecciona la operación **op** y continúa.
6. El sistema carga las reglas asociadas al constructor seleccionado y solicita la regla **rule** que el usuario va a editar.
7. El usuario selecciona la regla **rule** y continúa.

8. El sistema carga los datos de **rule**:

- Los constructores del modelo de datos (entidades y relaciones) y deja seleccionado por defecto el constructor **constr** de la operación dependiente de **rule**.
- Las operaciones de **constr** y se deja seleccionado por defecto la operación **reqop**.
- El número de ejecuciones de la operación requeridas por la regla (**reqnum**) y el orden **reqdir** de la ejecución de **reqop.**
- Las reglas asociadas a **op** y deja seleccionada por defecto **ruleRel** relacionada con **rule**, de igual modo que el operador (AND/OR) **operator** que relaciona **ruleRel** y **rule**

9. El usuario modifica cualquier campo relacionado con la regla **rule** o los deja igual y confirma la edición.

10. El sistema edita la regla **rule** asociada a **op** y que depende de la operación **reqop**, que se deberá ejecutar **reqnum** veces y en el momento indicado por **reqdir** para considerar la operación **op** correcta. Esta regla se relaciona con la regla **ruleRel** mediante el operador **operator** (AND/OR), formando una regla lógica entra dos reglas de **op**.

Cursos alternativos:

9) El usuario no ha introducido algún valor de los solicitados por el sistema en los puntos 3, 5, 7, 9.

1. El sistema informa del error y finaliza el caso de uso.

9) El usuario ha seleccionado una operación en la dependencia que se refiere a la propia operación de la regla.

1. El sistema informa del error y finaliza el caso de uso.

***) Si el usuario cancela el caso de uso**

1. Finaliza el caso de uso.

3.2.2.6 Caso de uso *Eliminar Regla Dependencia*

Descripción: Se elimina una regla asociada a una operación del modelo de datos cargado en el sistema.

Actores: Diseñador

Pre-condiciones:

Se ha cargado un proyecto del sistema que contiene el modelo de datos y el modelo navegacional.
Post-condiciones: El sistema elimina la regla existente **rule** asociada a la operación **op**.
Curso principal: 1. El usuario inicia el caso de uso 2. El sistema carga los constructores del modelo de datos (entidades y relaciones) y solicita seleccionar uno de ellos. 3. El usuario selecciona la entidad. 4. El sistema carga las operaciones del constructor seleccionado y solicita la operación a la cual se le asignará la regla. 5. El usuario selecciona la operación **op** y continúa. 6. El sistema carga las reglas asociadas al constructor seleccionado y solicita la regla **rule** que el usuario va a editar. 7. El usuario selecciona la regla **rule** y continúa. 8. El sistema carga los datos correspondientes a **rule**. 9. El usuario confirma la eliminación de la regla. 10. El sistema elimina la regla **rule** asociada a **op**
Cursos alternativos: **9) El usuario no ha introducido algún valor de los solicitados por el sistema en los puntos 3, 5, 7.** 1. El sistema informa del error y finaliza el caso de uso. ***) Si el usuario cancela el caso de uso** 1. Finaliza el caso de uso.

3.2.2.7 Caso de uso *Nuevo Proyecto*

Descripción: Se crea un nuevo proyecto del sistema.
Actores: Diseñador
Pre-condiciones: No hay ningún proyecto cargado actualmente en el sistema
Post-condiciones: Se ha creado y cargado un nuevo proyecto con el nombre y la ubicación del sistema

de almacenaje definido por el usuario.
Curso principal: 1. El sistema solicita al usuario la introducción del nombre del proyecto y la ubicación en el disco donde se almacenarán los ficheros. 2. El usuario introduce el nombre y la ubicación del disco. 3. El sistema muestra los datos y solicita confirmación para almacenarlos 4. El usuario confirma la creación del nuevo proyecto 5. El sistema crea un nuevo proyecto con el nombre y la ubicación indicada por el diseñador y lo carga.
Cursos alternativos: ***) Si el usuario cancela la creación del proyecto** 1. El sistema regresa al paso 2 del curso principal del caso de uso sin crear ningún proyecto nuevo **2) Si el usuario no introduce nombre o ubicación del proyecto en el disco.** 1. El sistema alerta del error y regresa al paso 1 del curso principal

3.2.2.8 Caso de uso *Abrir Proyecto*

Descripción: El sistema carga los datos del proyecto seleccionado por el usuario.
Actores: Diseñador
Pre-condiciones: No hay ningún proyecto cargado actualmente en el sistema
Post-condiciones: Se ha cargado en el sistema todos los datos del proyecto seleccionado por el usuario.
Curso principal: 1. El sistema solicita al usuario la ubicación en disco del proyecto que desea abrir 2. El diseñador indica la ubicación del proyecto (path) 3. El sistema carga el proyecto seleccionado del disco
Cursos alternativos: ***) Si el usuario cancela la selección del proyecto** 1. El sistema regresa al paso 2 del curso principal del caso de uso sin abrir ningún proyecto del disco

2) Si el usuario selecciona una ubicación que no contiene ningún proyecto del sistema. 1. El sistema alerta del error y regresa al paso 1

3.2.2.9 Caso de uso *Guardar Proyecto Actual*

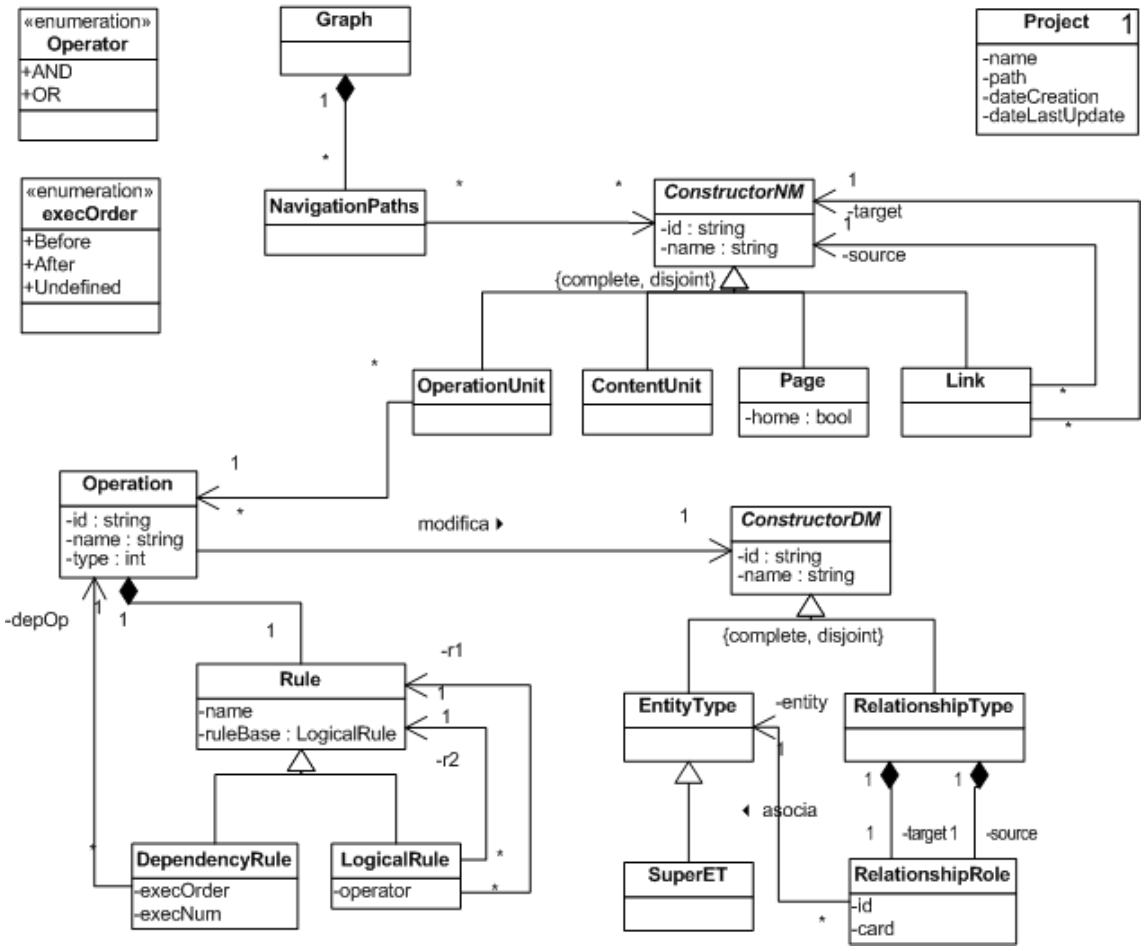
Descripción: El usuario de la herramienta guarda el proyecto actual, salvando los datos cargados en el sistema en el estado actual.
Actores: Diseñador
Pre-condiciones: Existe un proyecto cargado en el sistema
Post-condiciones: Los datos del proyecto actualmente cargado en el sistema están guardados en el estado actual.
Curso principal: 1. El sistema muestra la ubicación en el disco donde se guardará el proyecto y solicita al usuario la confirmación para guardar el proyecto. 2. Si el usuario confirma la acción de guardar 3. El sistema guarda el proyecto en la ubicación indicada
Cursos alternativos: **1) Si el sistema no encuentra la ruta indicada o no hay suficiente espacio** 1. El sistema alerta del error y permite al usuario seleccionar otra ubicación de guardado. 2. Si el usuario selecciona otra ubicación de guardado, la nueva ruta se le asigna al proyecto y vuelve al paso 1

3.2.2.10 Caso de uso *Cerrar Proyecto Actual*

Descripción: El usuario cierra el proyecto actual, dejando inaccesibles los datos
Actores: Diseñador
Pre-condiciones:

Existe un proyecto cargado en el sistema
Post-condiciones: No existe ningún proyecto cargado en el sistema.
Curso principal: 1. El sistema comprueba que el proyecto esté guardado y pregunta al diseñador si realmente quiere cerrar el proyecto. 2. Si el diseñador confirma el cierre. 3. El sistema cierra el proyecto dejando el sistema abierto sin ningún proyecto cargado.
Cursos alternativos: ***) Si el proyecto no está guardado:** 1. El sistema envía un mensaje advirtiendo de la situación al diseñador. Además ofrece al diseñador saltar a la opción de guardar el proyecto, para lo cual solicita al usuario la confirmación de la acción del guardado. 2. Si el diseñador confirma la acción de guardado, el sistema salta al paso 1 del curso principal

3.3 Modelo conceptual



Restricciones textuales

1. Claves clases no asociativas: (ConstructorDM, id), (EntityType, id), (RelationshipType, id), (RelationshipRole, id), (ConstructorNM, id), (Link, id), (Page, id), (ContentUnit, id), (OperationUnit, id), (Operation, id), (Rule, name) , (DependencyRule, name), (LogicalRule, name)

2. RelationshipType.target ≠ RelationshipType.source

3. LogicalRule.r1 ≠ LogicalRule.r2 o LogicalRule.r1 y LogicalRule.r2 son vacios.

4. DependencyRule.execNum >= 0

5. Link.source ≠ Link.target y Type(Link.source) ≠ Type(Link) y Type(Link.target) ≠ Type(Link)

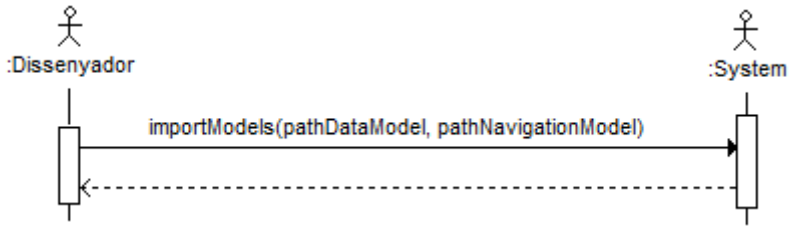
6. Para todas las instancias de DependencyRule asociadas a Dependency, Dependency.opReq ≠ DependencyRule.depOP

3.4 Modelo de comportamiento

En el modelo de comportamiento se definen con mayor detalle los escenarios de interacción actor-sistema más relevantes derivados de la especificación de los casos de uso definidos previamente.

Para cada escenario de los casos de uso, se definirá el diagrama de secuencia correspondiente y los contratos de las operaciones del sistema.

3.4.1 Caso de uso *Importar Modelos*



context importModels (pathDataModel: string, pathNavigationModel: string)

Se importa el modelo de datos situado en *pathDataModel* y el modelo navegacional situado en *pathNavigationModel* al sistema

post.2.1 → Se ha realizado la importación del contenido de los modelos al sistema, traduciendo la información contenida en las ubicaciones de disco pathDataModel y pathNavigationModel y creando las instancias de los constructores de los modelos de datos y navegacional.

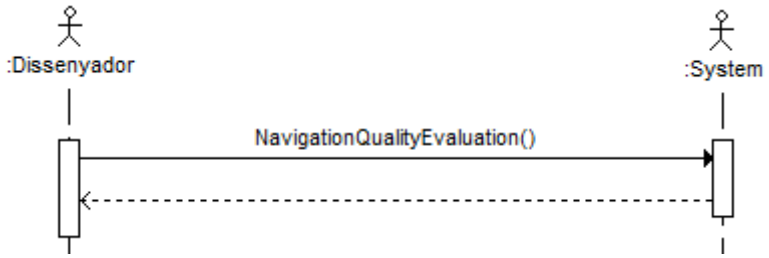
3.4.2 Caso de uso *Visualizar las rutas navegacionales*

context DrawModel ()

Se visualizan los paths del modelo navegacional

post.2.1 → El sistema muestra las rutas navegacionales del modelo N.

3.4.3 Caso de uso *Evaluación Cualitativa del Modelo Navegacional*

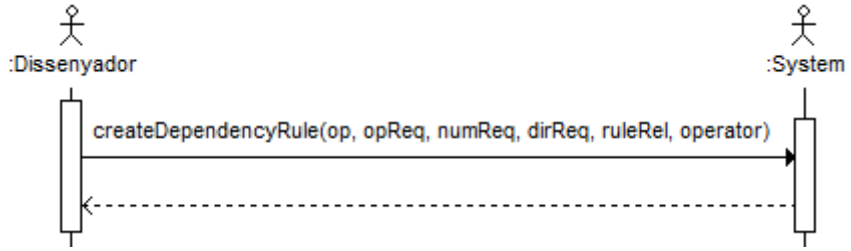


context NavigationQualityEvaluation ()

El sistema evalúa la calidad del modelo navegacional en base a las propiedades de completitud y correctitud

post.2.1 → Se han evaluado las propiedades cualitativas correctitud y completitud del modelo navegacional.

3.4.4 Caso de uso *Nueva Regla de Dependencia*



context createDependencyRule (op: Operation, opReq: Operation, numReq: Integer, dirReq: String, ruleRel: Rule, operator: String)

Se crea una nueva regla que afecta a la operación op con los datos indicados.

pre.1.1 → *op* es una operación definida en el modelo de datos.

pre.1.2 → *opReq* existe.

pre.1.3 → *numReq* contiene un número >=0

pre.1.4 → *dirReq* = {BEFORE, AFTER, UNDEFINED}

pre.1.5 → *ruleRel* es una regla existente en *op*, o es vacío si *op* no tiene reglas asociadas.

pre.1.6 → *operator* = {AND, OR}

post.2.1 → Se ha creado una nueva instancia r de RuleDependency, donde:

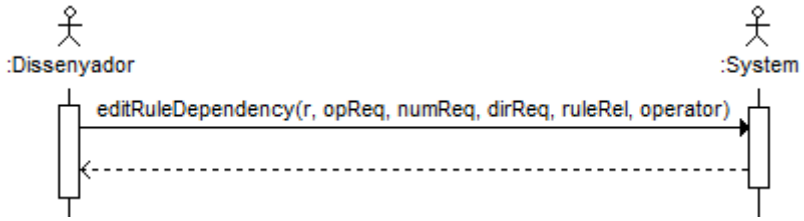
- r.opReq = opReq
- r.numExec = numReq
- r.dirExec = dirReq

post.2.2.a →Si ruleRel == null y op.Rule == null, entonces op.Rule = r

post.2.2.b →Si ruleRel <> null y op.Rule <> null, entonces se ha creado una nueva instancia rlog e RuleLogical, donde:

- rLog.r1 = r
- rLog.r2 = ruleRel
- rLog.operator = operator
- rLog.ruleBase = ruleRel.ruleBase
- rLog.r1.ruleBase = rLog
- rLog.r2.ruleBase = rLog

3.4.5 Caso de uso *Editar Regla de Dependencia*



context editDependencyRule (r: RuleDependency, opReq: Operation, numReq: Integer, dirReq: String, ruleRel: Rule, operator: String)

Se crea una nueva regla que afecta a la operación op con los datos indicados.

pre.1.1 → *r* existe.

pre.1.2 → *opReq* existe.

pre.1.3 → *numReq* contiene un número >=0

pre.1.4 → *dirReq* = {BEFORE, AFTER, UNDEFINED}

pre.1.5 → *ruleRel* es una regla existente en *op*, o es vacío si *op* no tiene reglas asociadas.

pre.1.6 → *operator* = {AND, OR}

post.2.1 → Se ha editado la regla de dependencia r, donde:

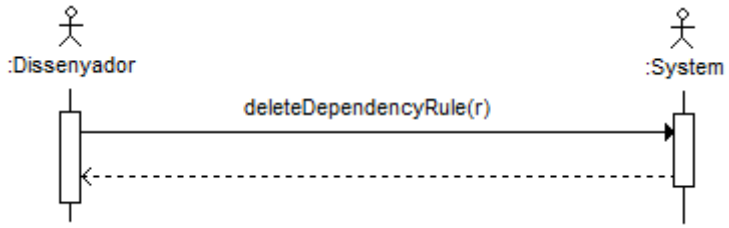
- r.opReq = opReq
- r.numExec = numReq
- r.dirExec = dirReq

post.2.2.a →Si ruleRel == null y op.Rule == null, entonces op.Rule = r

post.2.2.b →Si ruleRel <> null y op.Rule <> null, entonces se ha creado una nueva instancia rlog e RuleLogical, donde:

- rLog.r1 = r
- rLog.r2 = ruleRel
- rLog.operator = operator
- rLog.ruleBase = ruleRel.ruleBase
- rLog.r1.ruleBase = rLog
- rLog.r2.ruleBase = rLog

3.4.6 Caso de uso *Eliminar Regla de Dependencia*



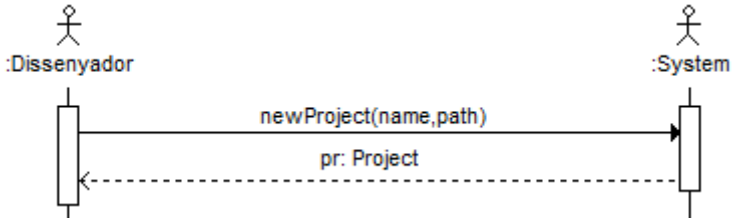
context deleteDependencyRule (r: RuleDependency)

Se elimina la regla r

pre.1.1 → La regla r existe

post.2.1 → Se elimina la regla r de la operación.

3.4.7 Caso de uso *Nuevo Proyecto*



context newProject (name: String, path: String): Project

Se crea un nuevo proyecto con el nombre y el path indicados donde se guardarán persistentemente los modelos manejados por la aplicación.

pre.1.1 → *name* contiene un valor no vacío

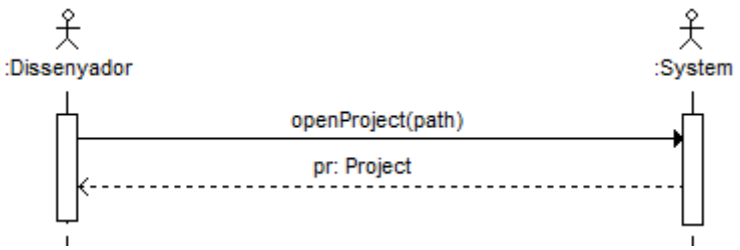
pre.1.2 → *Path* es una ruta accesible para la escritura

post.2.1 → Se define una nueva instancia de Project *pr*

post.2.2 → pr.name = *name*

post.2.3 → pr.path = *path*

3.4.8 Caso de uso *Abrir Proyecto*



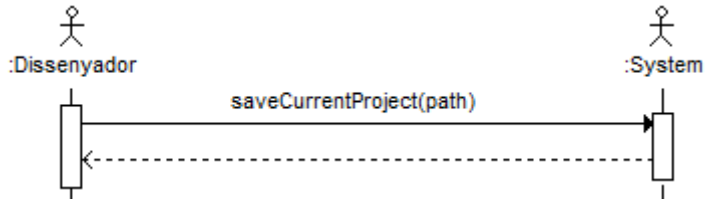
context openProject (path: String): Project

El sistema abre el proyecto ubicado en ***path*** y carga los modelos guardados en él.

pre.1.1 → Path es una ruta accesible para el sistema operativo y contiene un proyecto perteneciente a la herramienta

post.2.1 → Se ha cargado el proyecto *pr* ubicado en *path*

3.4.9 Caso de uso *Guardar Proyecto Actual*



context saveCurrentProject (path: String)

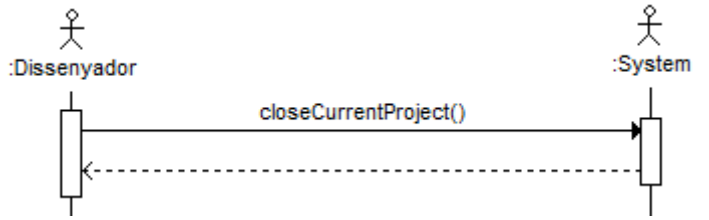
El sistema guarda el proyecto actualmente cargado con los datos de los modelos cargados actualmente en el sistema.

pre.1.1 → Si *path* no está vacío, *path* es una ruta accesible para la escritura.

post.2.1 → Si *path* no está vacío, el path del proyecto actualmente cargado = *path*

post.2.2 → El proyecto actualmente cargado se guarda en disco en la ruta indicada por el atributo path de la entidad project.

3.4.10 Caso de uso *Cerrar Proyecto Actual*



context closeCurrentProject()

El sistema cierra el proyecto actualmente cargado sin guardar los cambios realizados desde el último guardado.

post.2.1 → Se libera de memoria principal el proyecto cargado en el sistema

Capítulo 4. Diseño

4.1 Introducción

En la fase de diseño se define con total detalle cómo se desarrollará el sistema para que realice las funcionalidades descritas en la fase de especificación. Se detallarán los siguientes conceptos:

- Plataforma física. Se decidirá la estructura de hardware y comunicaciones que requiere el sistema.

- Arquitectura del sistema software. Se decidirá el modelo arquitectónico que debe seguir el sistema, obteniendo las diferentes capas y la distribución de éstas en la plataforma física definida.

- Diseño externo. Borrador de la interfaz gráfica de usuario, donde se incluirán las características visuales de todas las ventanas que aparecen en el sistema.

- Diseño interno. Se especificarán todas las capas definidas en el modelo arquitectónico, incluyendo las operaciones que las forman y aplicando patrones. Para poder llegar a este punto, se deben definir las responsabilidades de cada una de las capas previamente realizando una transición de la fase de especificación.

4.2 Plataforma física

En este apartado se define la configuración de la infraestructura de hardware y comunicación necesarios para el sistema.

El sistema se ejecutará en un PC simple, en el cual se almacenará la información generada por la aplicación y únicamente será utilizada por el propio sistema. Es decir, no se requiere compartir información con otro sistema.

No obstante, los datos que se importarán al sistema se obtienen de los ficheros XML generados por las herramientas de modelado. Es decir, el sistema se ejecutará sobre el mismo PC donde reside la herramienta de modelado, lo que garantiza el acceso a los datos que se importarán.

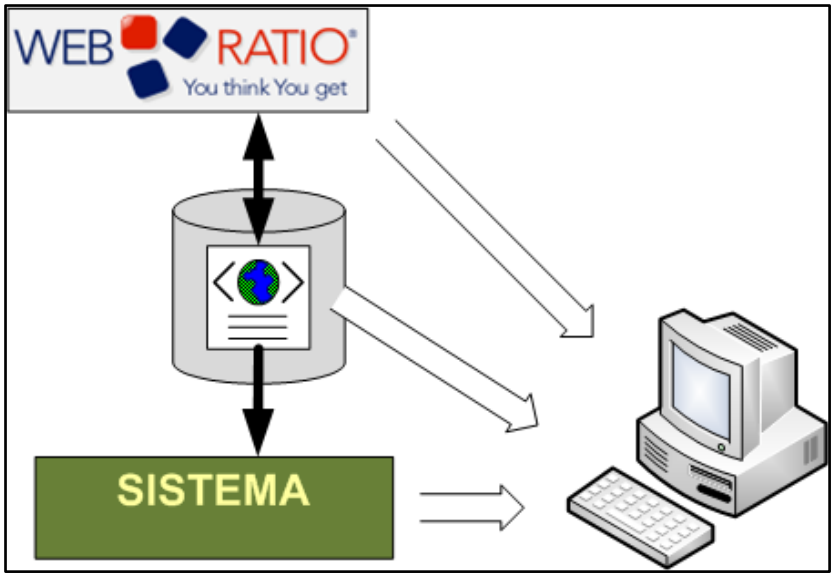


Fig. Plataforma física del sistema.

4.3 Arquitectura del sistema software

4.3.1 Modelo arquitectónico

El modelo arquitectónico que seguirá el diseño del sistema es el modelo de 3 capas. De este modo se podrá aislar las capas asignando responsabilidades a cada una de ellas y así poder trabajar sin depender de las demás capas. Esto facilitará que el sistema sea más reusable, portable y adaptable.

Las 3 capas que forman el modelo arquitectónico son las siguientes:

- Capa de presentación. Responsable de la interacción del sistema con el usuario. Por un lado recibe los eventos que le envía el usuario y los envía a la capa de dominio, y por el otro lado recibe desde el dominio el resultado de las consultas y presenta los datos al usuario.

- Capa de dominio. La capa de dominio recibe los eventos y consultas de la capa de presentación y procesa la información para solicitar los datos necesarios a la capa de gestión de datos. Una vez ha obtenido los datos solicitados, los vuelve a procesar para enviarlos a la capa de presentación.

- Capa de gestión de datos. La capa de datos es la encargada de devolver a la capa de dominio toda la información consultada y almacenada persistentemente en el sistema mediante ficheros XML.

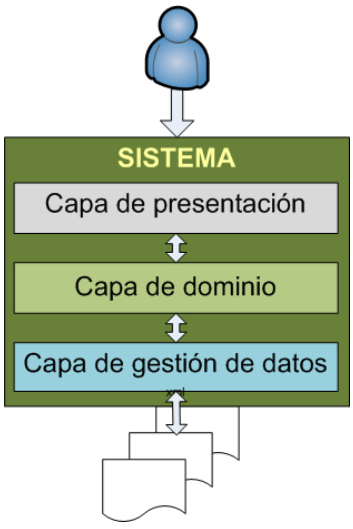


Fig. Distribución de las capas definidas en el sistema

Respecto a la ubicación física de cada una de las capas y en base a la definición de la arquitectura física, todas las capas que forman el modelo arquitectónico se procesan sobre el mismo PC.

4.3.2 Gestión de la persistencia

En este apartado se definirá la estrategia a seguir para la gestión de la persistencia de los datos manipulados por el sistema. Es decir, cómo los almacena, los carga y los manipula.

La arquitectura de 3 capas ofrece dos estrategias posibles para esta gestión:

a) Generación automática

Se utiliza un sistema que proporciona una traducción automática para almacenar la información en el disco, lo que permite dejar la información en un estado persistente al realizar actualizaciones en memoria principal.

Esta estrategia hace que la gestión de la persistencia recaiga sobre la capa de dominio, liberando responsabilidad a la capa de gestión de datos, la cual únicamente contendrá el objeto que permitirá la persistencia automática.

b) Diseño directo

La capa de gestión de datos será la responsable de la gestión de la información que recibimos del SGBD o de los ficheros XML. La capa de dominio pasa a ser un intermediario entre capas, pudiendo realizar manipulaciones simples sobre la información recibida de la capa de datos.

El diseño directo requiere conocer el SGBD que se utilizará para almacenar los datos ya que se aprovechan las potentes funcionalidades del SGBD (*triggers, procedures, views, cheks, etc.*). Esto afecta al diseño del sistema, ya que se tiene que normalizar el modelo conceptual dependiendo de las características del SGBD. La capa de datos se diseñará en base al esquema de la base de datos.

Una vez expuestas las dos estrategias sobre la gestión de la persistencia en el diseño del sistema, se ha decidido utilizar la estrategia de **generación automática**, ya que los datos que se harán persistentes no requieren una gestión compleja y el volumen de datos es mínimo para utilizar un diseño directo. La generación automática mediante Java se realizará utilizando la interface *Serializable* en los objetos persistentes del dominio. De este modo, los datos se guardarán en el disco mediante ficheros únicos para cada objeto *Serializable*.

4.3.3 Tratamiento de errores

Es importante decidir cómo se tratarán los errores en cada operación, ya que los contratos de éstas dependen de esta decisión. Existen dos formas de llevar a cabo el tratamiento de errores.

a) Un error no se producirá nunca al invocar una operación porque se controla en una capa superior. La ausencia del error se establece en la precondición de la operación, y por lo tanto no se realizarán controles redundantes.

```
operacion(param1: T2, param2: T2): T

pre: P
post: Q
```

b) En el caso que no se controle el error en la capa superior, la operación deberá controlarlo. Devolverá un booleano que indique su ausencia o presencia, mientras que el valor que devolvía se convierte en un parámetro de salida. Por lo tanto, la postcondición contemplará la detección o no del error, y en el caso que se produzca no se realizará ninguna acción. Además, este control se traslada a los diagramas de secuencia que tendrán que controlar los posibles errores antes de realizar cualquier cambio de estado.

```
operacion(param1: T2, param2: T2, out res: T): Bool

post:
   1. Si ¬P → Devuelve falso y no hace nada
   2. De otro modo, devuelve cierto y:
      2.1.  Q
```

Una vez expuestas las estrategias de control de errores, se ha decidido que, en la medida de lo posible, el sistema controlará los errores en la capa de presentación, porque un elevado porcentaje de posibles errores se originan en la interacción del usuario con el sistema, es decir, a través de la interfaz gráfica de usuario.

Esto puede suponer la aparición de nuevas operaciones que preservarán la propagación de errores en las demás capas. Por ejemplo, las operaciones del caso de uso **editar regla de dependencia** usan una regla existente como parámetro de entrada. Para garantizar la ausencia de error en capas inferiores, se define una nueva operación que devuelva las reglas existentes y las muestra en la interfaz gráfica dentro

de un cuadro de lista no editable, de manera que el usuario solamente podrá seleccionar una regla existente.

Aunque no todos los posibles errores originan nuevas operaciones. Es posible que haya controles que se realicen en la propia operación de la capa superior, como comprobar que no se introducen valores nulos o vacíos, que la longitud del texto sea de un número determinado de caracteres o que una fecha es inferior a la fecha actual, entre otros.

Por otro lado, pueden existir comprobaciones de errores que no se pueden realizar en la capa de presentación, como son las relacionadas con la persistencia. En este caso el sistema deberá controlarlo en la propia capa de datos siguiendo la política de control correspondiente.

4.4 Diseño externo

En este apartado se define un borrador de los elementos que forman parte de la interfaz gráfica del usuario, como son los contenedores (ventanas o diálogos), y los controles incluidos en éstos (botones, cuadros de texto, listas desplegables, menús).

El sistema presenta una serie de contenedores importantes: la ventana principal, los diálogos que corresponden a los casos de uso, y los diálogos típicos de error o de información que pueden aparecer en cualquier caso de uso.

4.4.1 Diseño de la pantalla principal

A continuación se muestra el diseño de la interfaz gráfica de la pantalla principal. Esta ventana se mantendrá siempre visible para el usuario y está estructurada por los siguientes controles:

A. En la parte superior de la ventana se sitúa la barra de menús que contiene dos menús que corresponden a los 2 subsistemas definidos en los casos de uso: *Project* (Gestión de proyectos) y *Models* (Gestión de los modelos).

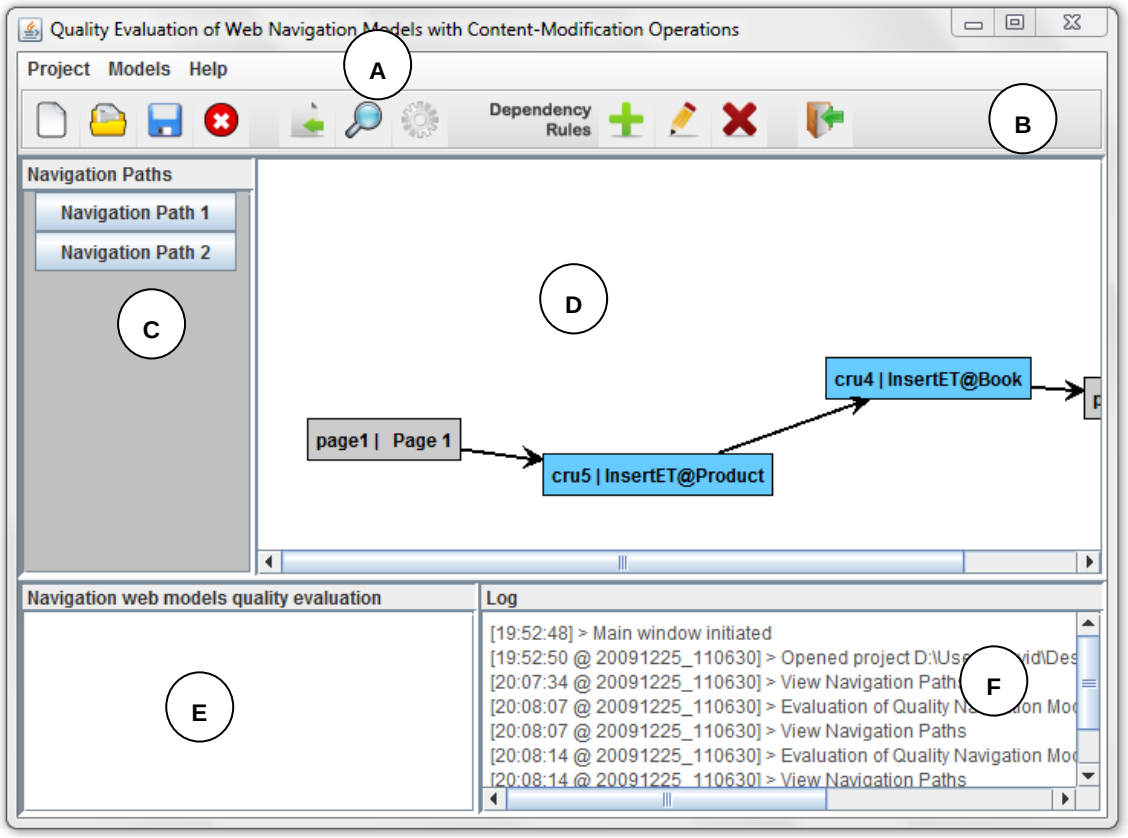
B. Debajo de la barra de menús, aparece una barra de iconos. Éstos permiten un acceso más directo a cada una de las funcionalidades que aparecen en el menú principal, sin tener que acceder al menú.

C. En la parte izquierda de la pantalla se sitúa un índice de las diferentes rutas del modelo navegacional cargado, formado por botones.

D. En la parte central de la pantalla se sitúa la visualización de las rutas navegacionales y el resultado de la evaluación de la correctitud.

E. En la parte inferior/izquierda se sitúa el resultado de la evaluación de la calidad del modelo navegacional.

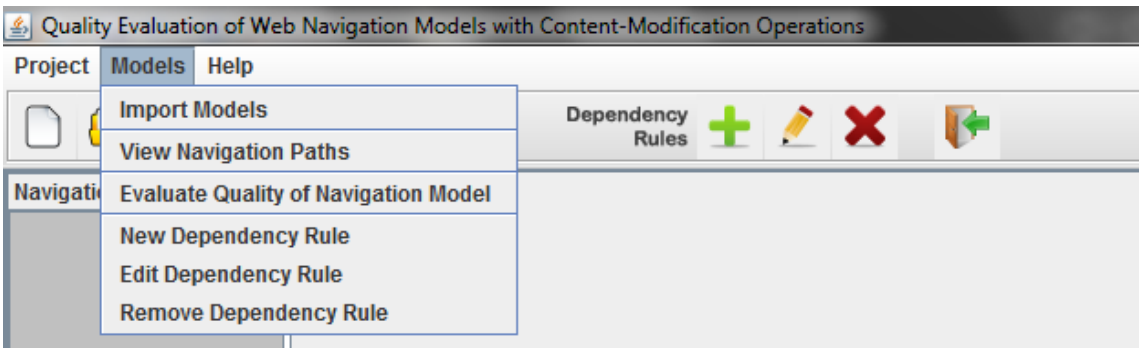
F. En la parte inferior/derecha de la pantalla se encuentra un cuadro de texto que registrará todos los eventos que se van produciendo.



Cabe destacar que los menús son los elementos que dispararán los diferentes casos de uso. A continuación se describe la estructura visual de cada menú:

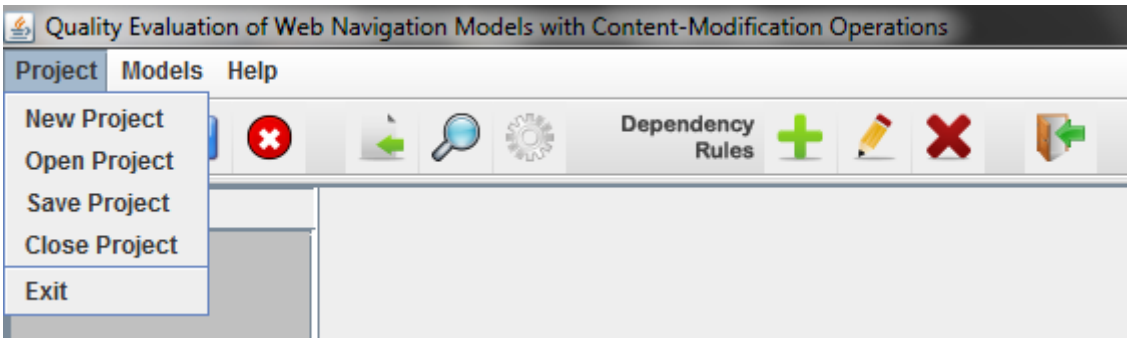
4.4.1.1 Menú gestión de los modelos

Todos los casos incluidos en el subsistema de gestión de importación son accesibles desde el menú principal, incluidos en el menú *Models*.



4.4.1.2 Menú gestión de proyectos

Todos los casos incluidos en el subsistema de gestión de importación son accesibles desde el menú principal, incluidos en el menú *Project*.

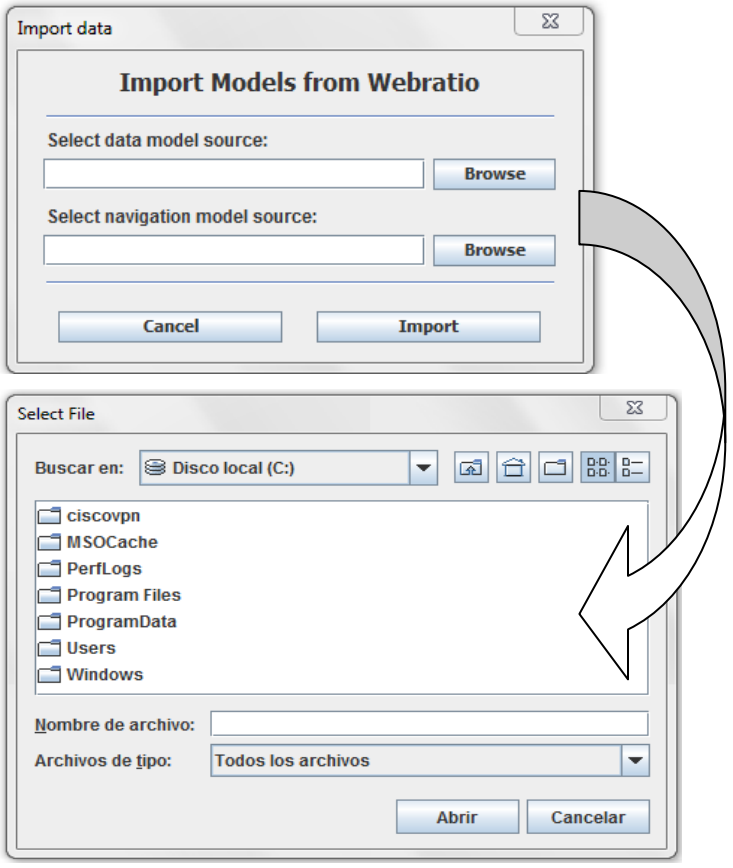


4.4.2 Diseño de las pantallas de los casos de uso

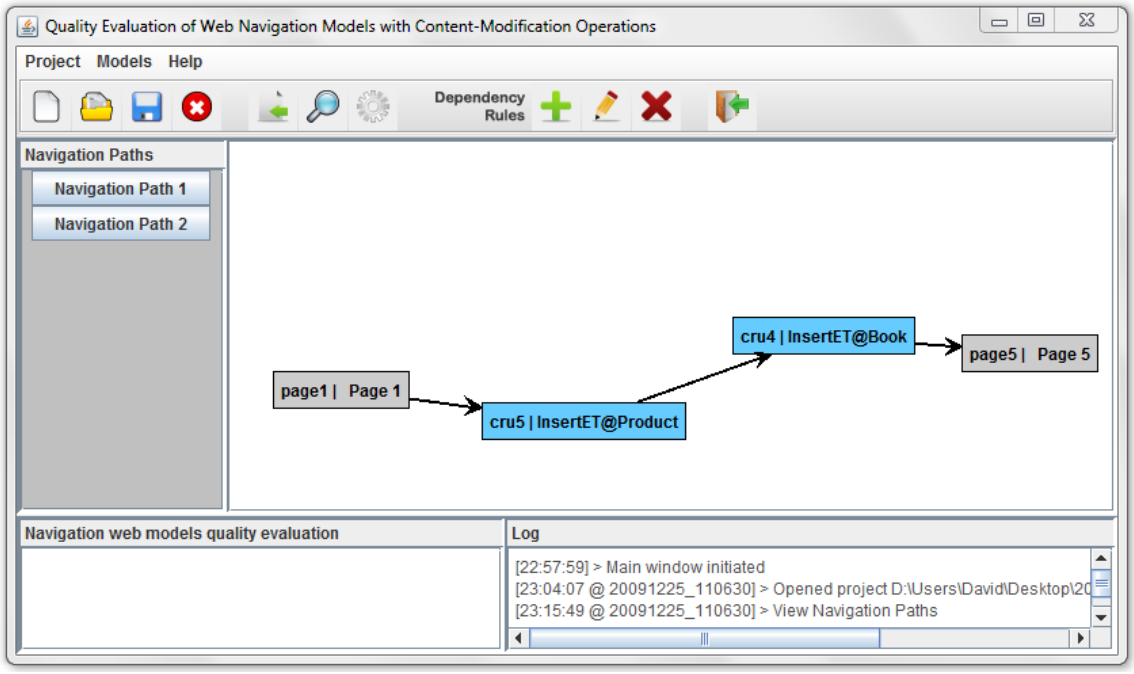
Existen varios subsistemas donde se agrupan una serie de casos de uso. Siguiendo esta agrupación se han diseñado los menús de la pantalla principal, donde cada menú es un subsistema que contiene los diferentes casos de uso accesibles a partir de las opciones incluidas en el menú.

A continuación, para cada subsistema se define la interfaz gráfica correspondiente a cada uno de los casos de uso definidos en la especificación y que se utilizarán en la definición de los casos de uso concretos.

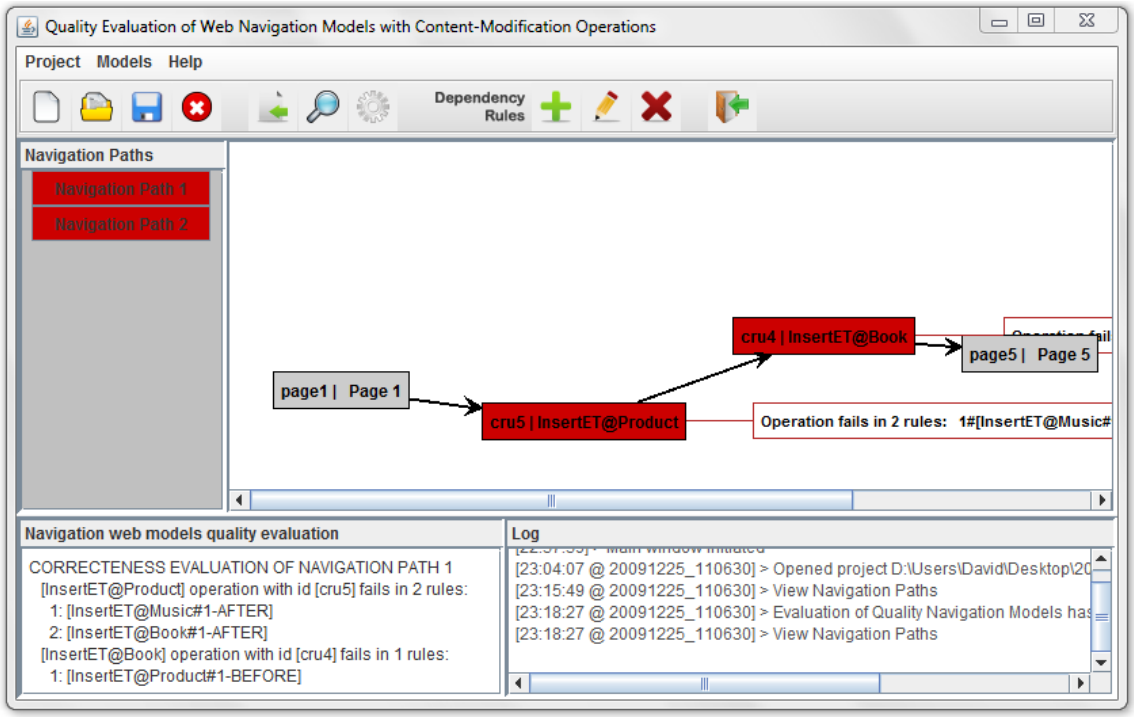
4.4.2.1 Importar Modelos



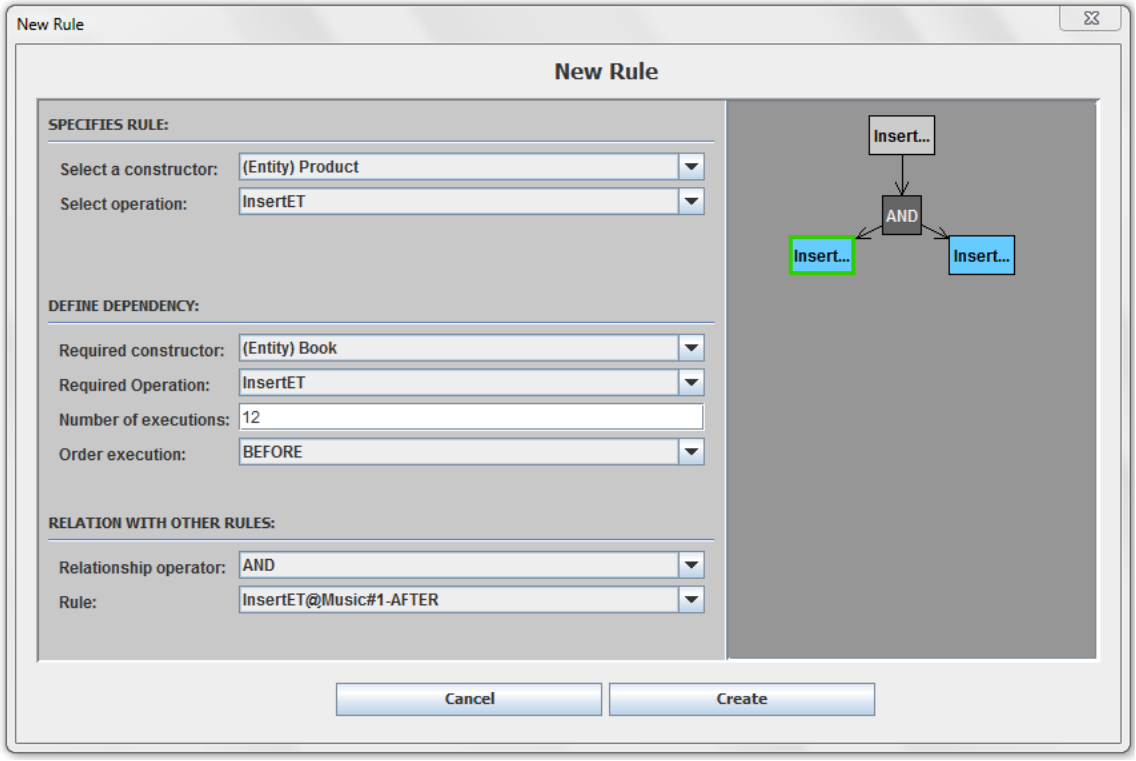
4.4.2.2 Visualizar las rutas del modelo navegacional cargado



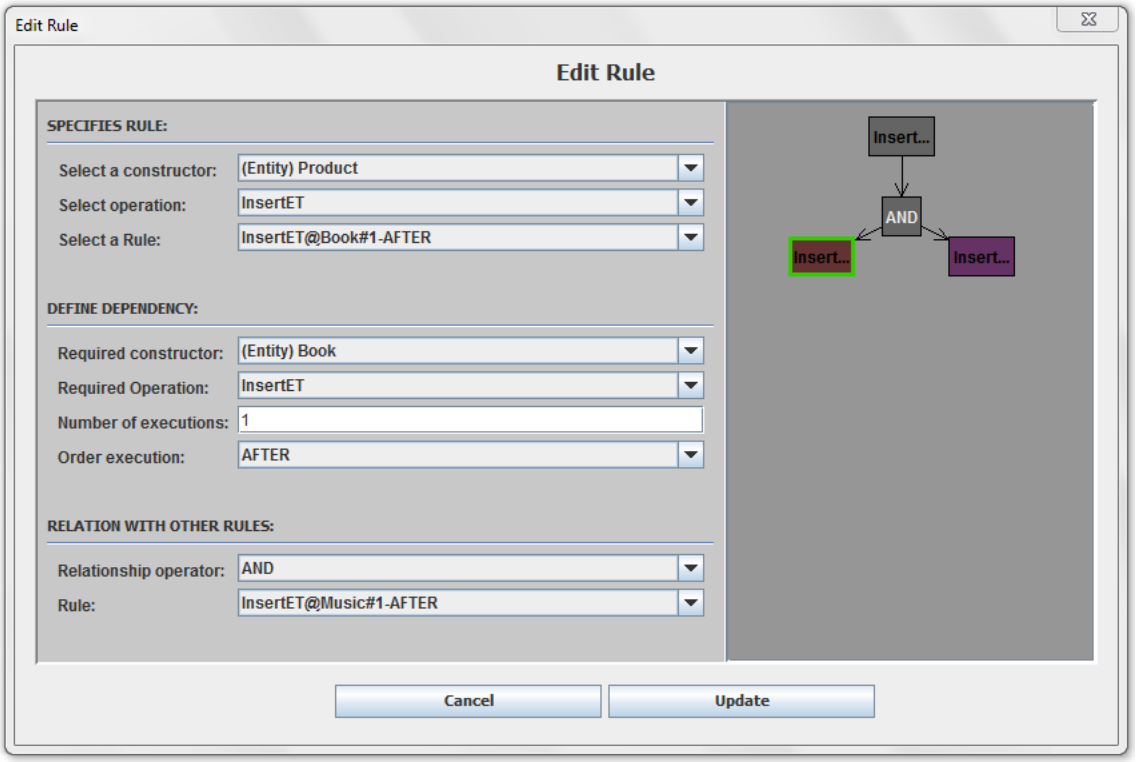
4.4.2.3 Evaluar la calidad del modelo navegacional



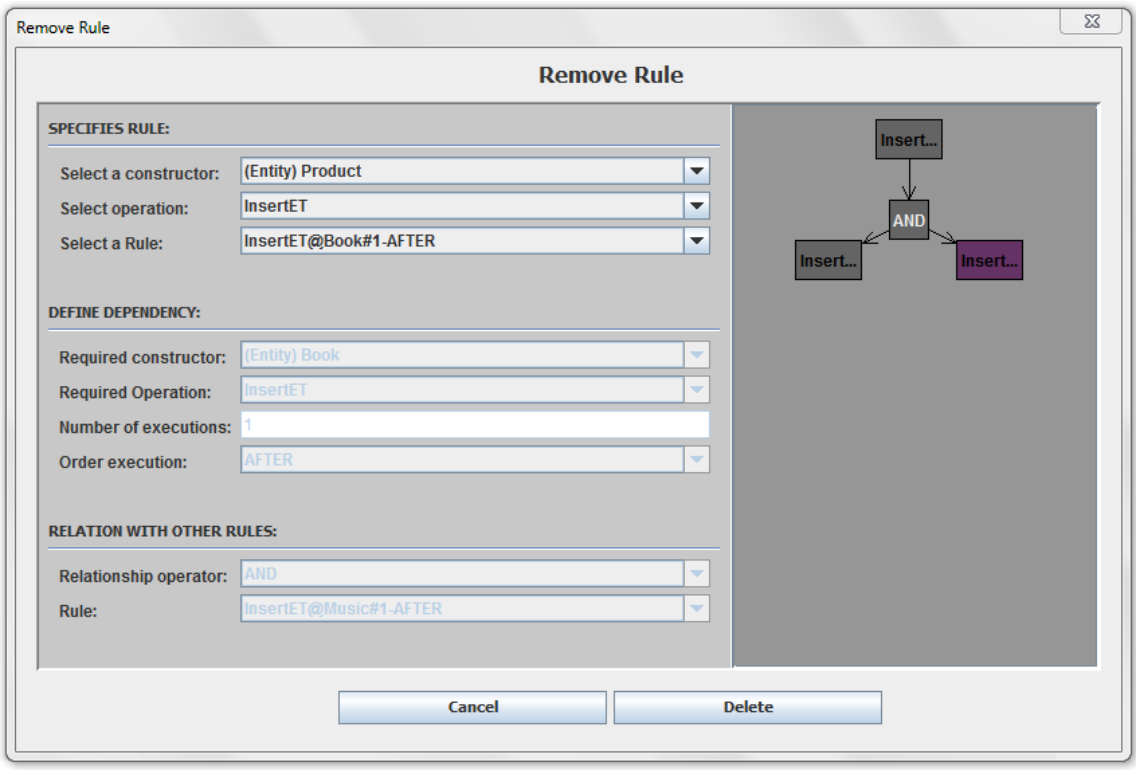
4.4.2.4 Nueva Regla de Dependencia



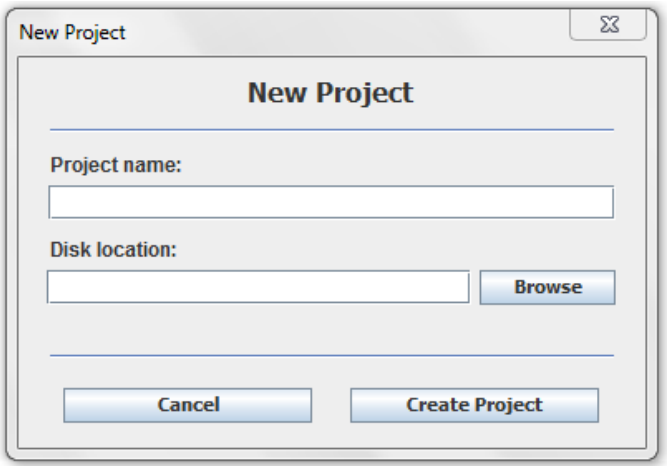
4.4.2.5 Editar Regla de Dependencia



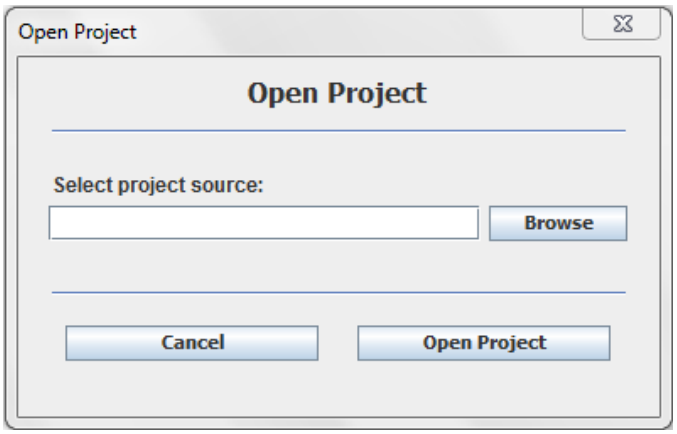
4.4.2.6 Eliminar Regla Dependencia



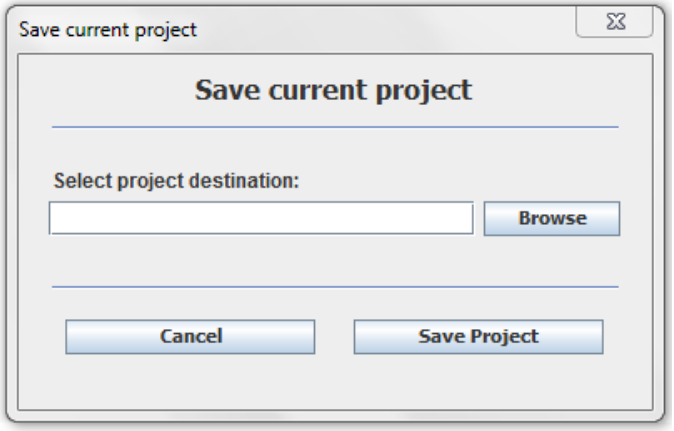
4.4.2.7 Nuevo Proyecto



4.4.2.8 Abrir Proyecto

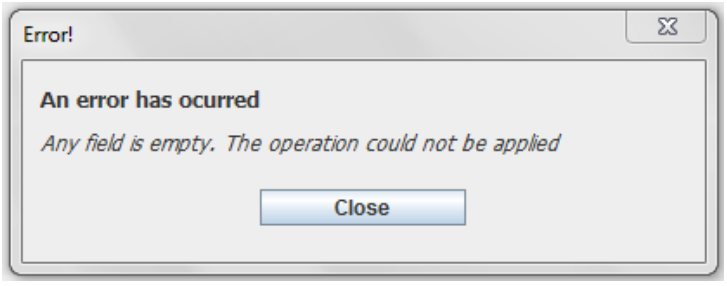


4.4.2.9 Guardar Proyecto Actual



4.4.3 Diseño de los cuadros de diálogos genéricos

4.4.3.1 Cuadro de diálogo de error

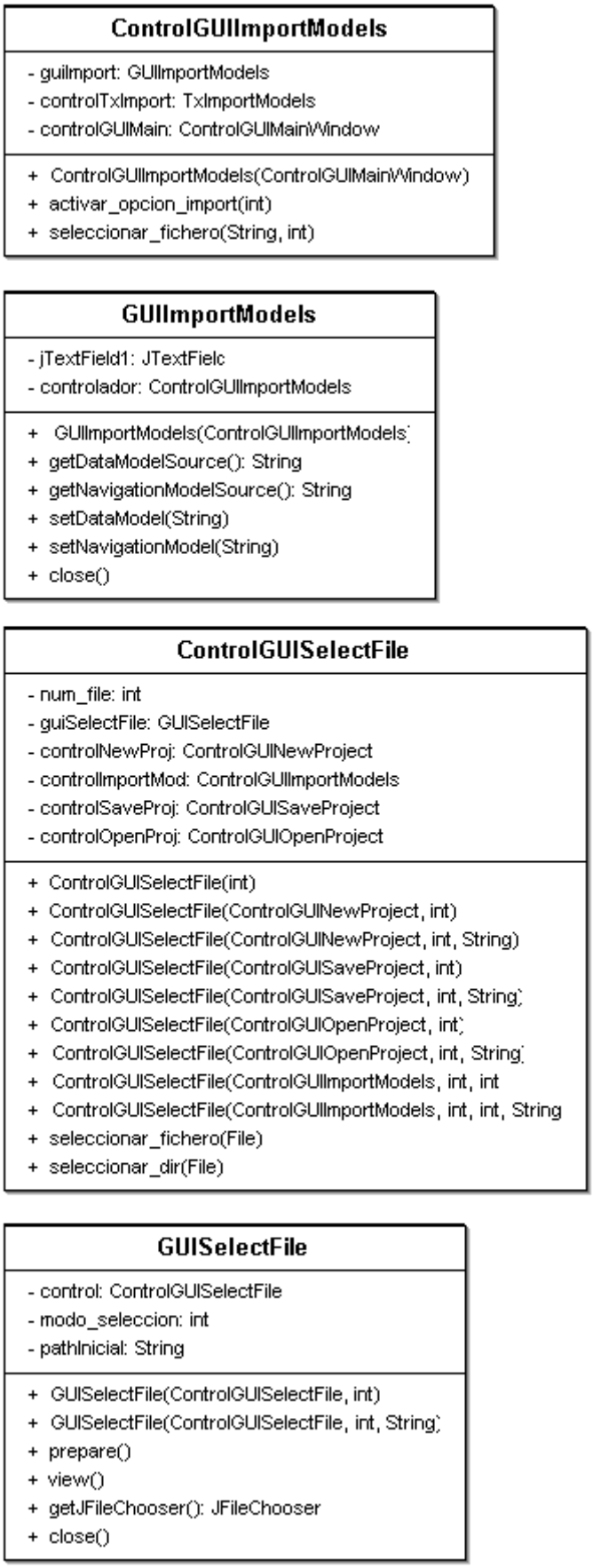


4.5 Diseño interno

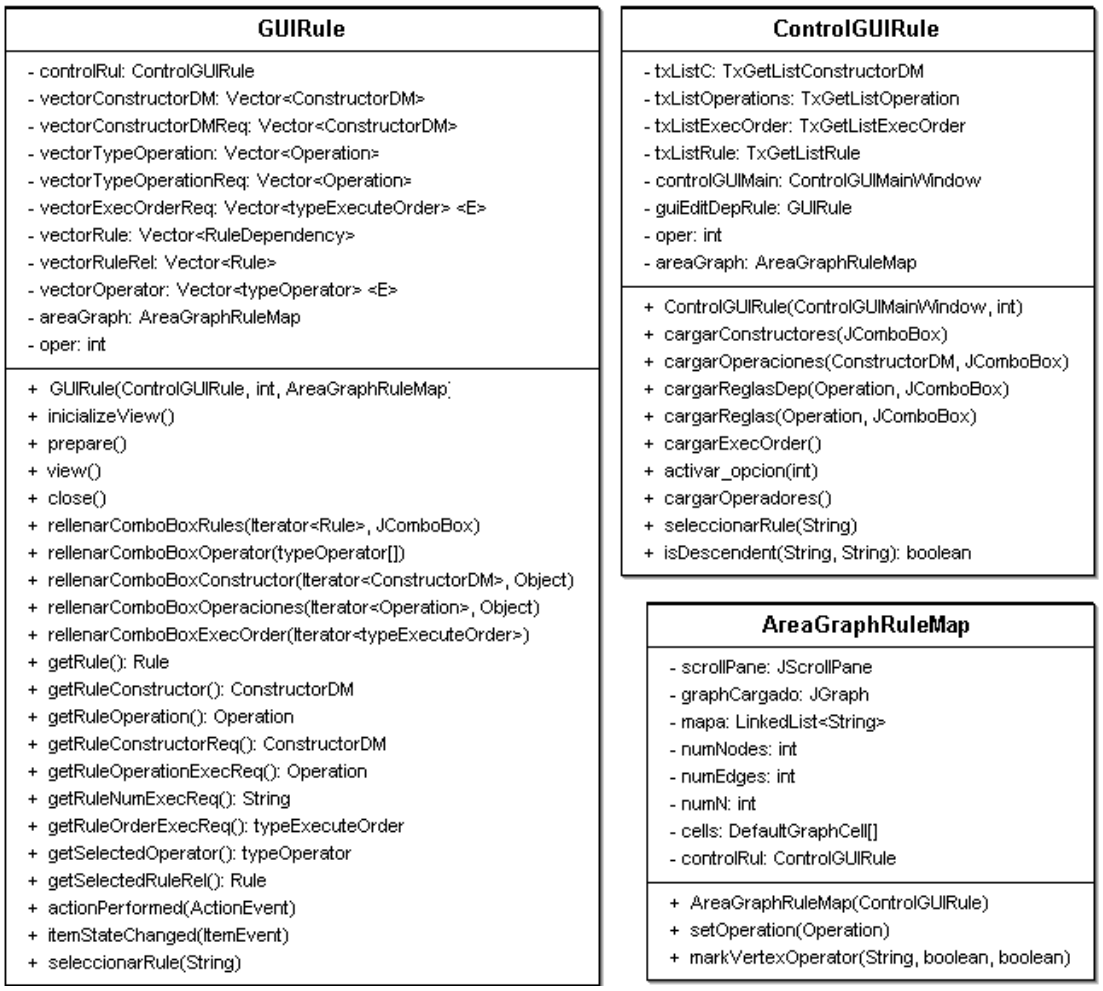
4.5.1 Diseño de la capa de presentación

La capa de presentación recibirá los eventos de los objetos que forman la interfaz gráfica de usuario. La comunicación con la capa de dominio se realizará a través de controladores. Se definirá un controlador para cada pantalla definida en el sistema.

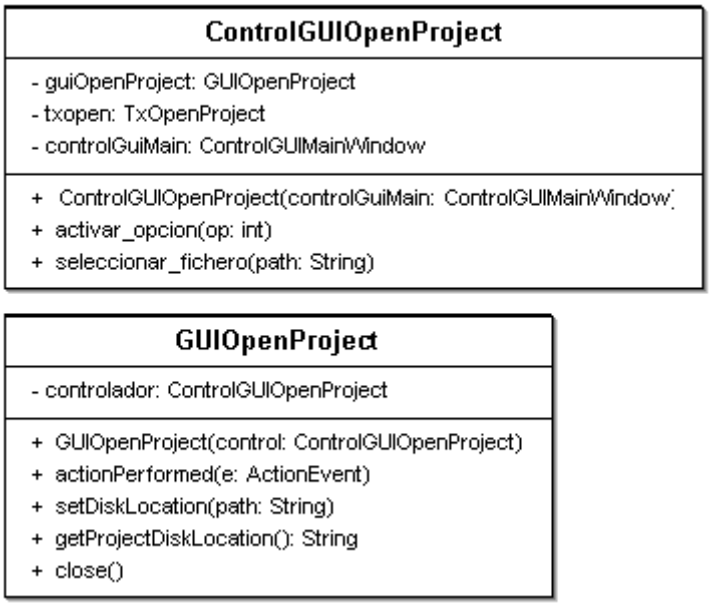
Clases correspondientes a la pantalla principal



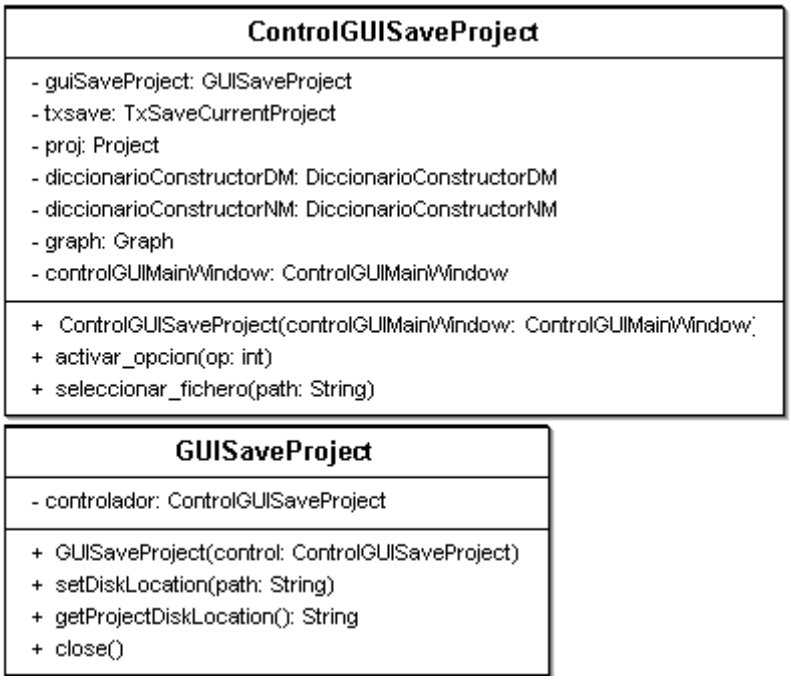
*Clases correspondientes a la interfaz del caso de uso **importación de modelos***



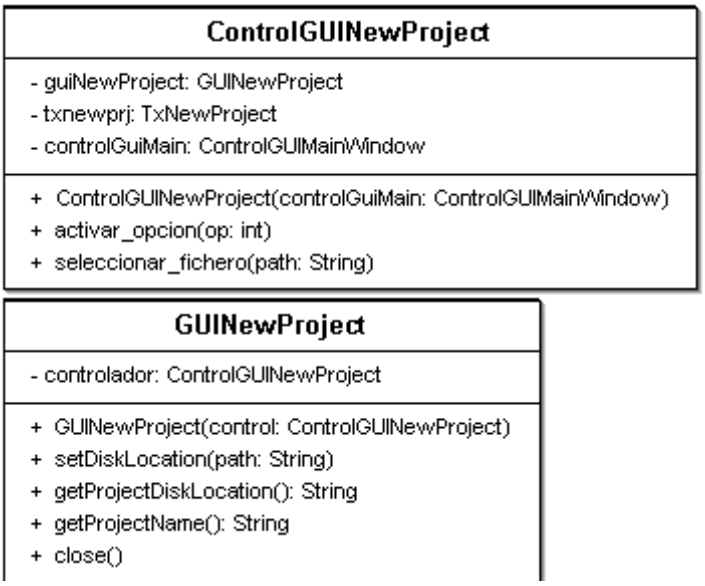
*Clases correspondientes a la interfaz de usuario de **gestión de reglas de dependencia.***



*Clases correspondientes a la interfaz de usuario **abrir proyecto***



*Clases correspondientes a la interfaz de usuario **guardar proyecto***



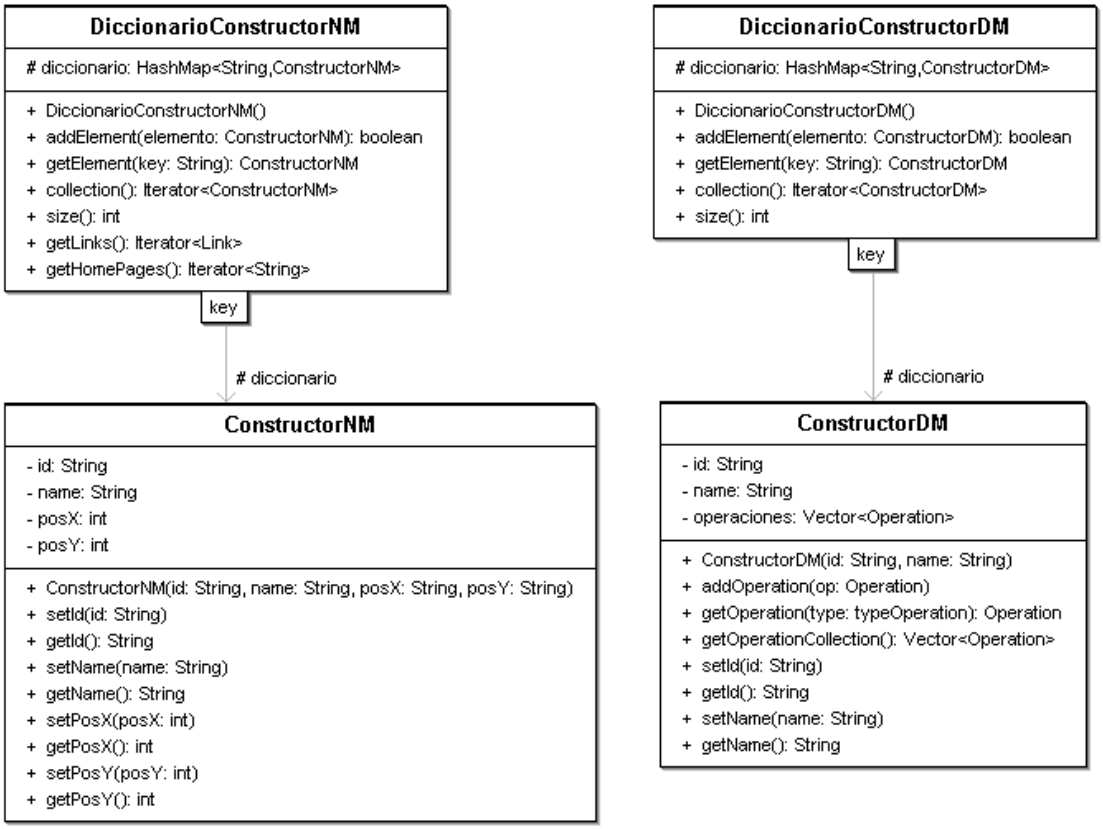
*Clases correspondientes a la interfaz de usuario **nuevo proyecto***

4.5.2 Diseño de la capa de dominio

La capa de dominio contiene la lógica del sistema y es la responsable de procesar los eventos y consultas de la capa de presentación. Estas responsabilidades se asignarán a controladores, que serán los objetos del dominio encargados de la comunicación entre las demás capas adyacentes. Al mismo tiempo, deben delegar el tratamiento del evento sobre los objetos del modelo conceptual. A continuación se muestra el modelo conceptual, donde se incluyen las operaciones propias de cada clase.

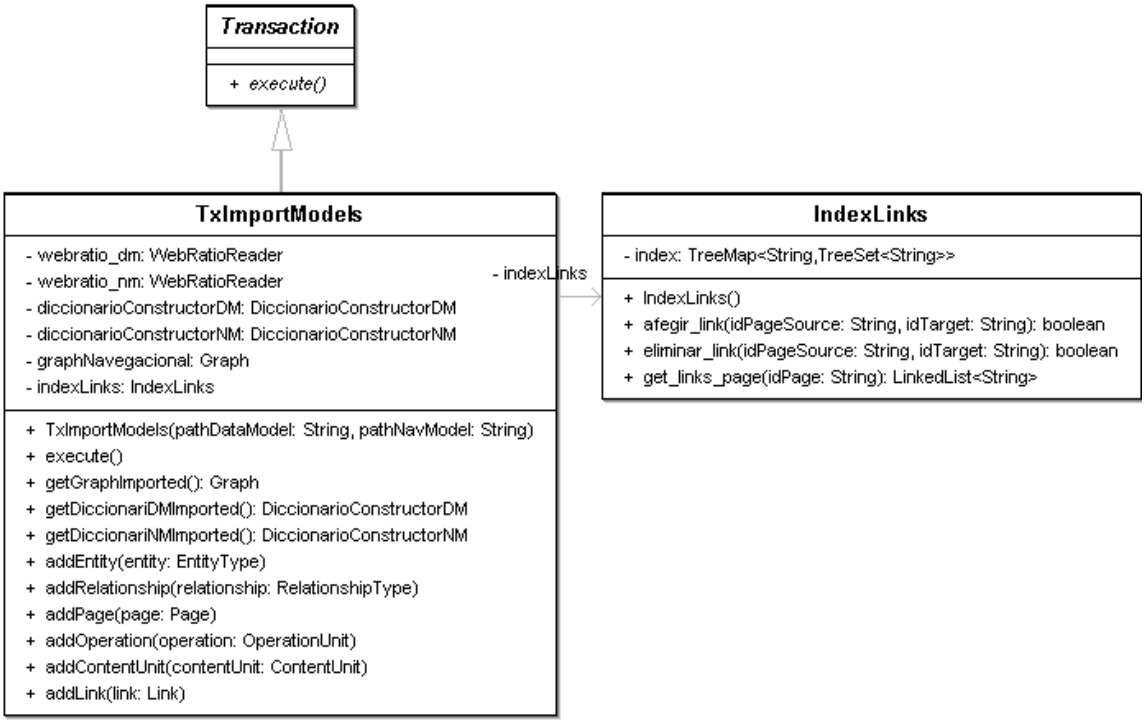

Project
- name: String
- path: String
- dateCreation: Date
- dateLastUpdate: Date
+ Project(String, String, Date, Date)
+ getName(): String
+ setName(String)
+ getPath(): String
+ setPath(String)
+ getDateCreation(): Date
+ setDateCreation(Date)
+ getDateLastUpdate(): Date
+ setDateLastUpdate(Date)
ConstructorNM
- id: String
- name: String
- posX: int
- posY: int
+ ConstructorNM(String, String, String, String)
+ setId(String)
+ getId(): String
+ setName(String)
+ getName(): String
+ setPosX(int)
+ getPosX(): int
+ setPosY(int)
+ getPosY(): int
Page
- isHomePage: boolean
+ Page(String, String, boolean, String, String)
+ isHomePage(): boolean
+ setHomePage(boolean)
ContentUnit
- pageID: String
+ ContentUnit(String, String, String, String, String)
+ getPageID(): String
+ setPageID(String)
Link
- source: String
- target: String
+ Link(String, String, String, String)
+ getSource(): String
+ setSource(String)
+ getTarget(): String
+ setTarget(String)
ConstructorDM
- id: String
- name: String
- operaciones: Vector<Operation>
+ ConstructorDM(String, String)
+ addOperation(Operation)
+ getOperation(typeOperation): Operation
+ getOperationCollection(): Vector<Operation>
+ setId(String)
+ getId(): String
+ setName(String)
+ getName(): String
EntityType
- relaciones: Vector<RelationshipType>
- subEntities: Vector<EntityType>
- idSuperEntity: String
+ EntityType(String, String, String)
+ getRelacionesCollection(): Vector<RelationshipType>
+ addRelacion(RelationshipType)
+ getSubEntities(): Vector<EntityType>
+ addSubEntity(EntityType)
+ getSuperType(): String
+ setSuperType(String)
RelationshipType
- target: RelationshipRole
- source: RelationshipRole
+ RelationshipType(String, String)
+ setTarget(RelationshipRole)
+ getTarget(): RelationshipRole
+ setSource(RelationshipRole)
+ getSource(): RelationshipRole
RelationshipRole
- id: String
- name: String
- card: String
- entity: EntityType
+ RelationshipRole(String, String, String, EntityType)
+ getEntity(): EntityType
«enumeration»
typeOperator
+ AND
+ OR
OperationUnit
- operation: Operation
- isCorrect: int
- failRules: Vector<Rule>
+ OperationUnit(String, String, Operation, String, String)
+ getOperation(): Operation
+ setOperation(Operation)
+ isCorrect(): int
+ setIsCorrect(int)
+ addFailRule(Rule)
+ getFailRules(): Vector<Rule>
Rule
- name: String
- opDep: Operation
- writeable: boolean
- ruleBase: RuleLogical
+ Rule(String, Operation, boolean)
+ setRuleBase(RuleLogical)
+ getRuleBase(): RuleLogical
+ setName(String)
+ getName(): String
+ setOpDep(Operation)
+ getOpDep(): Operation
+ setWriteable(boolean)
+ isWriteable(): boolean
RuleLogical
- optr: typeOperator
- r1: Rule
- r2: Rule
+ RuleLogical(String, Operation, boolean, typeOperator, Rule, Rule)
+ setOptr(typeOperator)
+ getOptr(): typeOperator
+ setR2(Rule)
+ getR2(): Rule
+ setR1(Rule)
+ getR1(): Rule
RuleDependency
- execOrder: typeExecuteOrder
- execNum: int
- execOp: Operation
+ RuleDependency(Operation, boolean, typeExecuteOrder, int, Operation)
+ setExecNum(int)
+ getExecNum(): int
+ setExecOrder(typeExecuteOrder)
+ getExecOrder(): typeExecuteOrder
+ setOpReq(Operation)
+ getOpReq(): Operation
«enumeration»
typeOperation
+ InsertET
+ UpdateET
+ DeleteET
+ ConnectRT
+ DisconnectRT
«enumeration»
typeExecuteOrder
+ BEFORE
+ AFTER
+ UNDEFINED
Operation
- idOperation: String
- constructorDM: ConstructorDM
- type: typeOperation
- rootRule: Rule
+ Operation(typeOperation, ConstructorDM)
+ removeRegla(Rule)
+ addRegla(Rule, typeOperator)
+ addRegla(Rule, Rule, typeOperator)
+ editRegla(RuleDependency, Rule, typeOperator)
+ getReglasCollection(): Vector<Rule>
+ getNumRules(): int
+ getReglaDep(): Rule
+ setReglaDep(Rule)
+ getIdOperation(): String
+ setType(typeOperation)
+ getType(): typeOperation
+ setElementAffected(ConstructorDM)
+ getElementAffected(): ConstructorDM
Graph
- map: Map<String,LinkedHashSet<ConstructorNM>>
- paths: Vector<NavigationPath>
+ addEdge(ConstructorNM, ConstructorNM)
+ buscarPaths(Iterator<ConstructorNM>)
+ getPaths(): Vector<NavigationPath>
NavigationPath
- path: LinkedList<ConstructorNM> [0..*]
+ getPath(): LinkedList<ConstructorNM>
+ pathIterator(): Iterator<ConstructorNM>
+ isPathEmpty(): boolean
+ containsPath(ConstructorNM): boolean
+ containsAllPath(Collection<ConstructorNM>): boolean
+ pathSize(): int
+ pathToArray(): ConstructorNM[]
+ NavigationPath()
+ add(ConstructorNM)
+ getLast(): ConstructorNM
+ contains(ConstructorNM): boolean
+ removeLast()
+ addLast(ConstructorNM)
+ size(): int
+ get(int): ConstructorNM

Para acceder rápidamente a la información de los objetos que forman el modelo de datos y navegacional, se usarán diccionarios que ayudarán a acceder a cada objeto a partir del identificador. Se definen dos diccionarios, uno para los constructores del modelo de datos (EntityType, RelationshipType) y otro para los constructores del modelo navegacional (Page, Link, OperationUnit, ContentUnit).

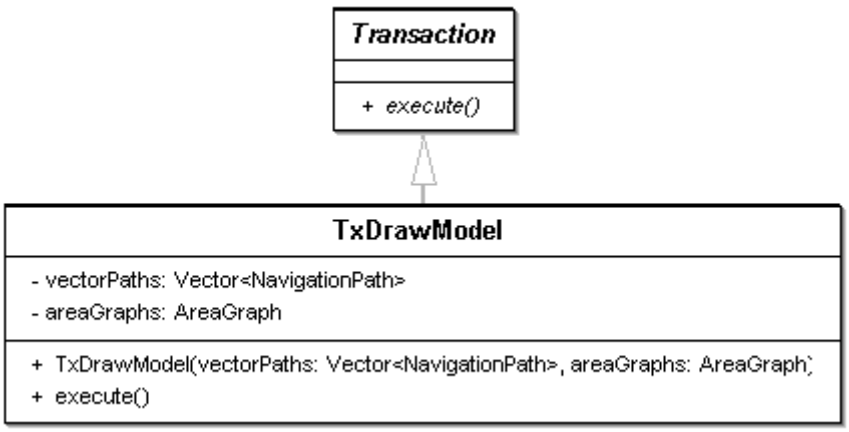


Se definirá un controlador para cada operación del sistema existente. Cada controlador de transacción heredará la clase abstracta ***Transacción***, que contendrá la operación abstracta ***execute()***. Los parámetros de las operaciones del sistema originarán los atributos de los controladores correspondientes. Además, si la operación devuelve algún resultado, aparece un nuevo atributo **result**, que se podrá consultar mediante la operación **getResult()**.

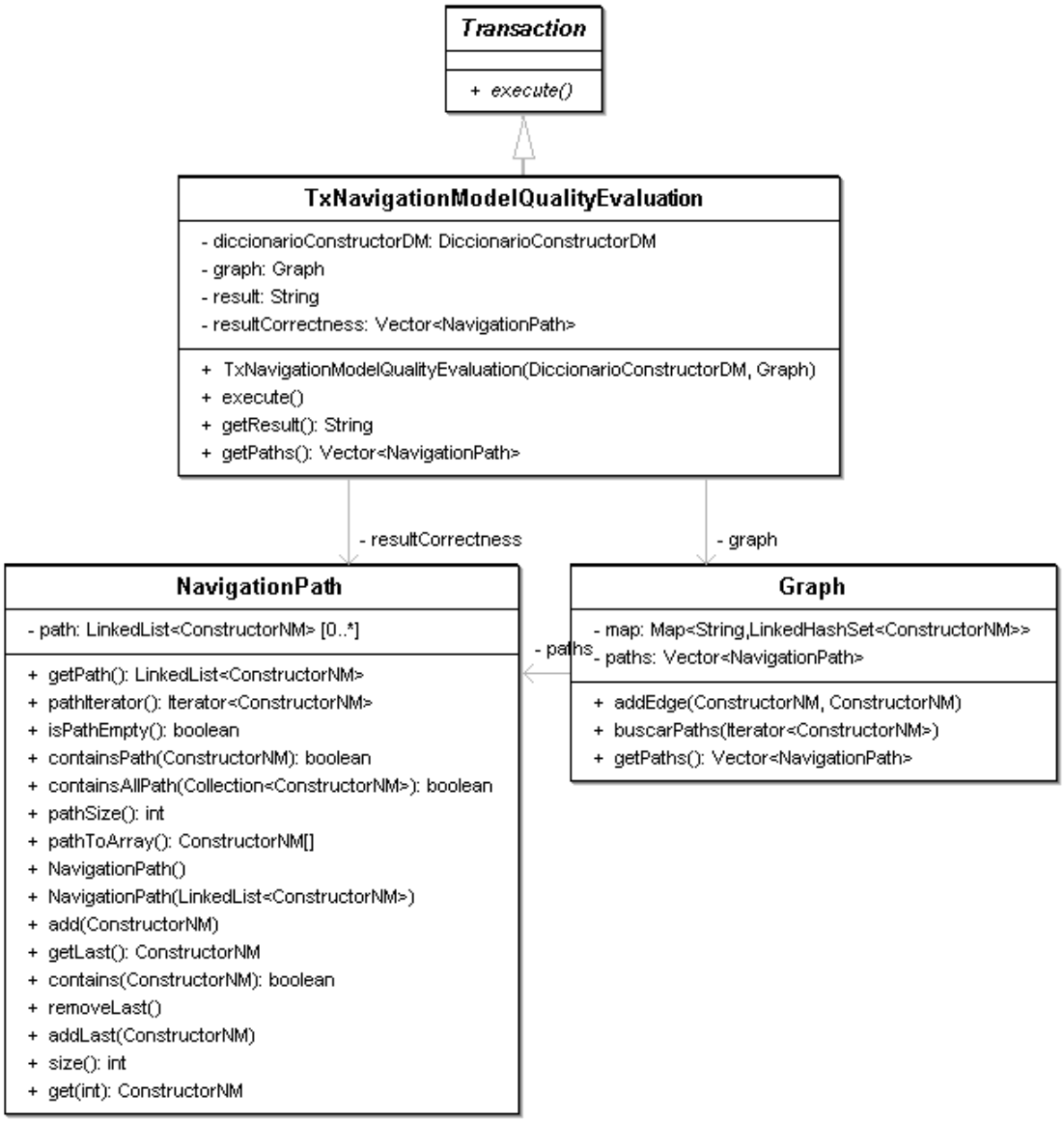
4.5.2.1 Importar Modelos



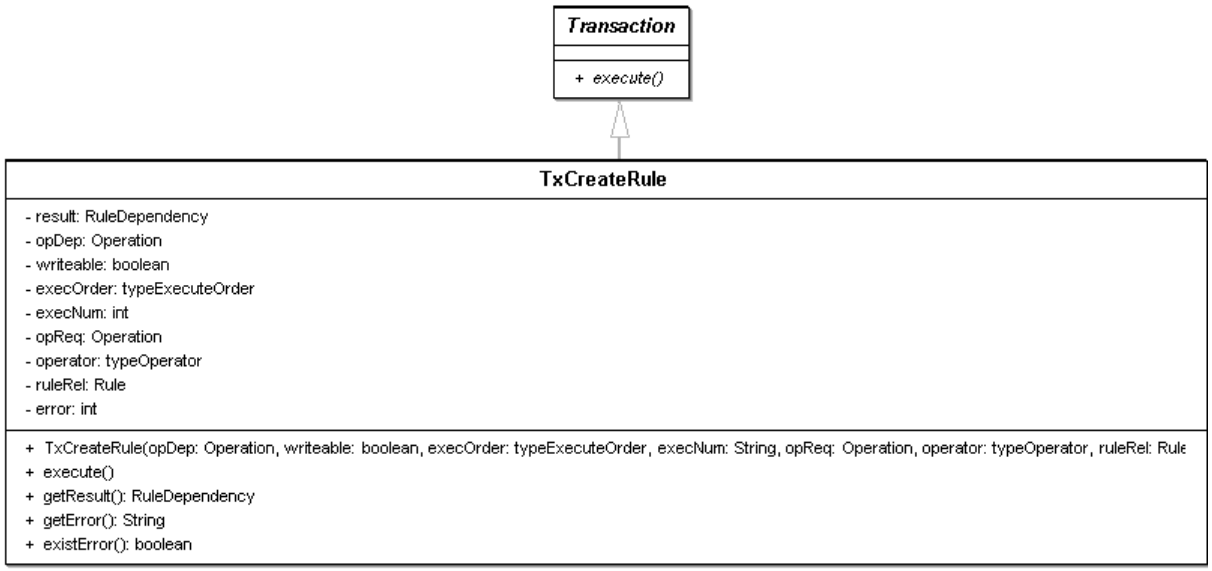
4.5.2.2 Visualizar las rutas navegacionales



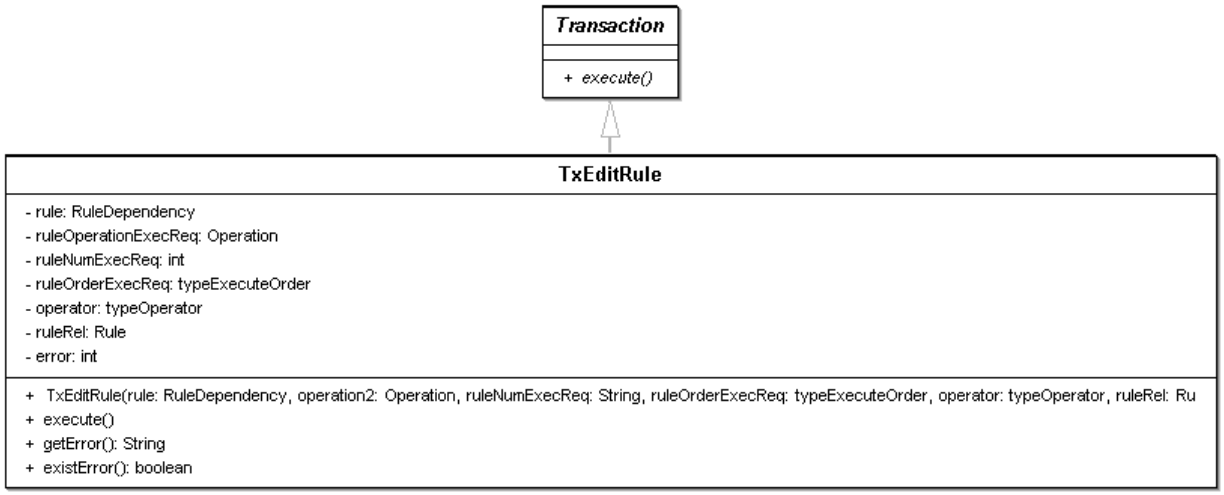
4.5.2.3 Evaluar la calidad del modelo navegacional



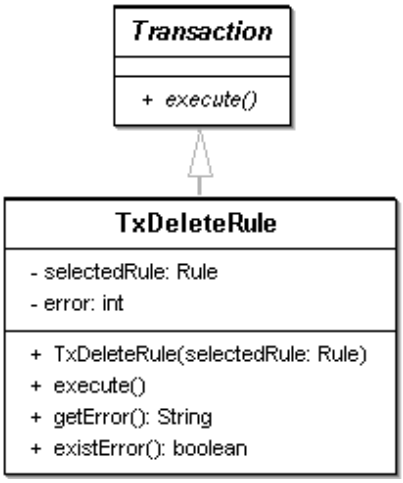
4.5.2.4 Nueva Regla de Dependencia



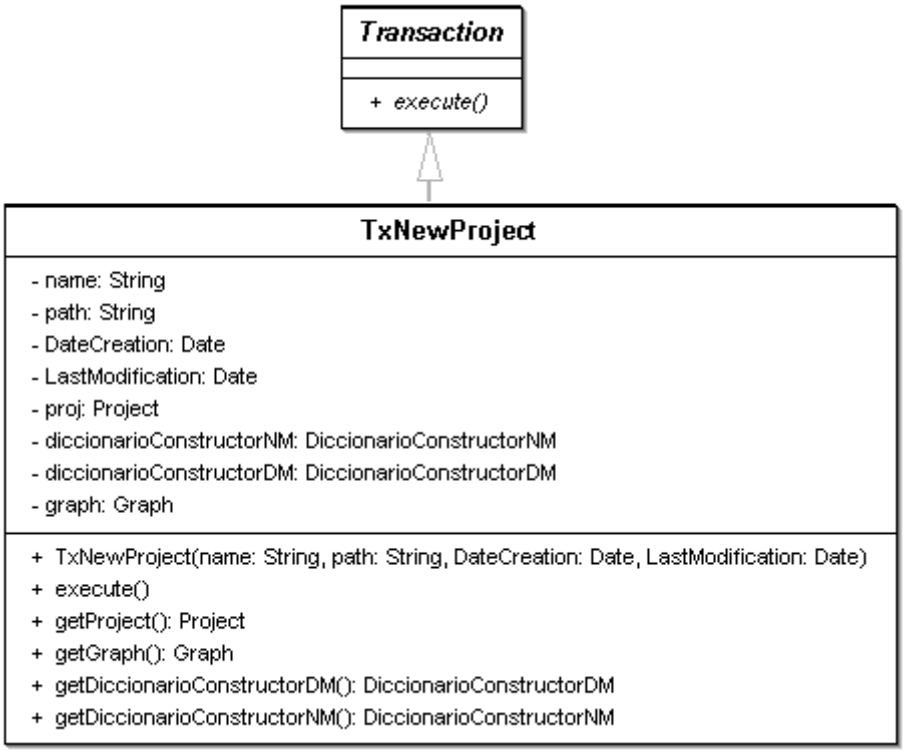
4.5.2.5 Editar Regla de Dependencia



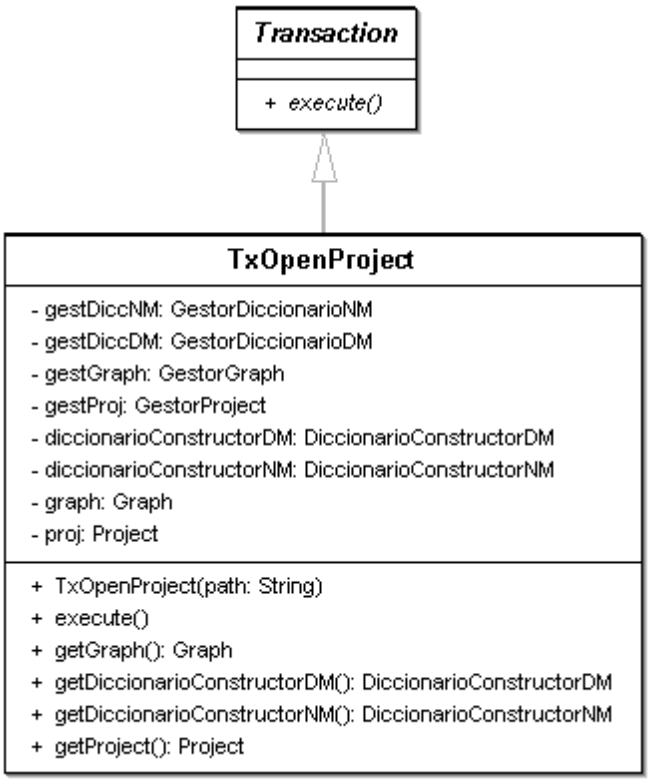
4.5.2.6 Eliminar Regla de Dependencia



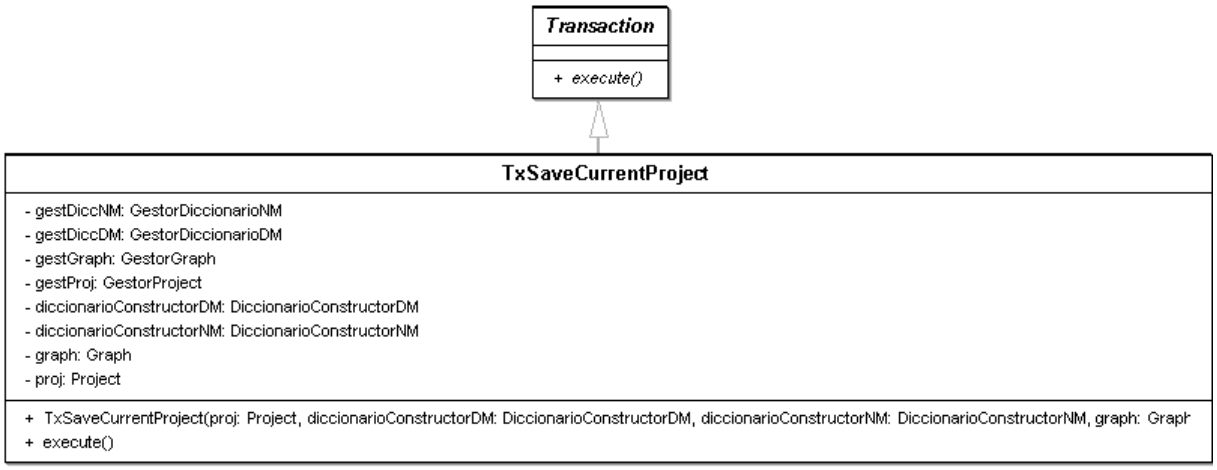
4.5.2.7 Nuevo Proyecto



4.5.2.8 Abrir Proyecto

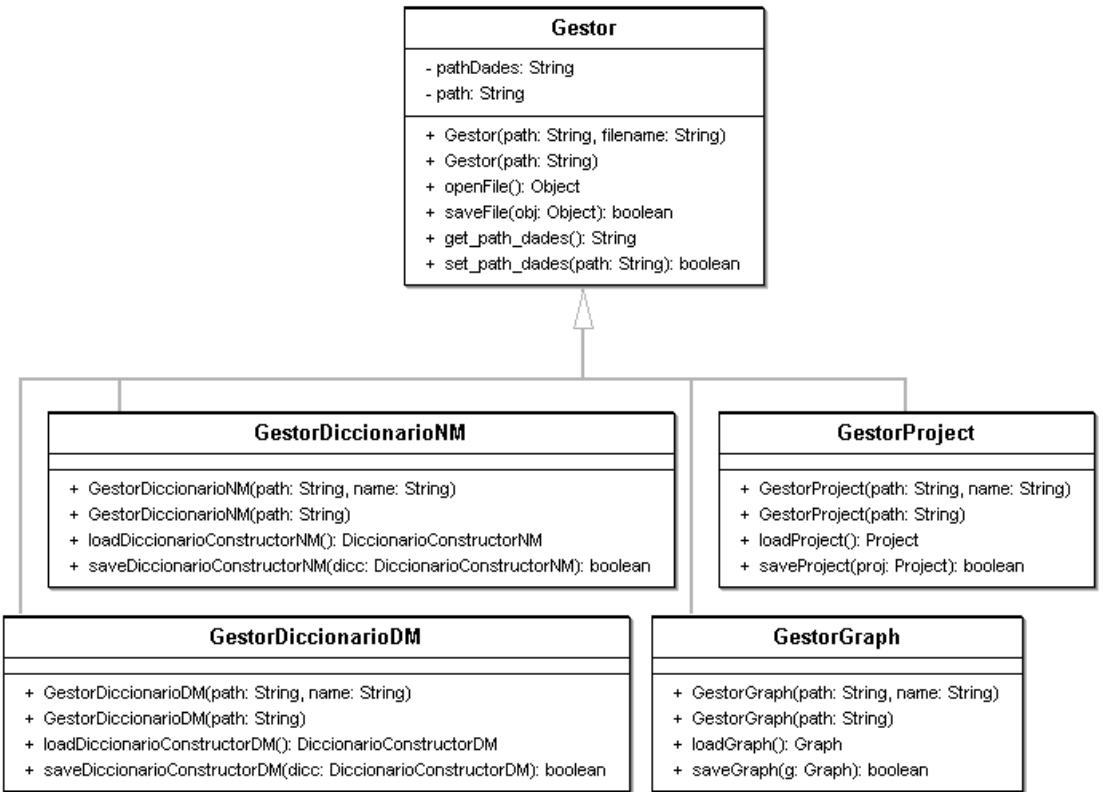


4.5.2.9 Guardar Proyecto Actual



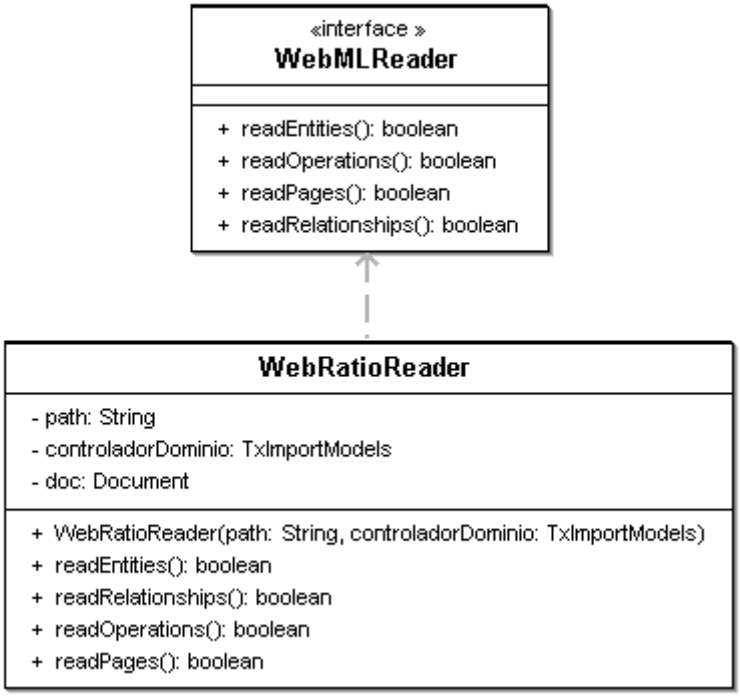
4.5.3 Diseño de la capa de gestión de datos

La capa de gestión de datos es la responsable de la persistencia de los datos de la aplicación. Al haber elegido una política de gestión automática de la persistencia, la capa de datos se limita a recibir solicitudes de la capa de dominio para almacenar o recuperar los datos de un dispositivo persistente. Para poder guardar en disco las clases persistentes de Java, éstas se deben implementar utilizando la interface *Serializable.* La capa de datos utilizará las librerías propias de Java para el tratamiento de la persistencia incluidas en el package java.io (*FileInputStream, FileOutputStream, ObjectInputStream, ObjectOutputStream*)



Por otro lado, el sistema tiene que importar los datos generados en XML con la herramienta de modelado externa (*WebRatio*) y adaptarlos para su posterior uso. Para conseguir esto se ha definido una interface para la lectura de los objetos definidos en el modelo conceptual. Esta interface esta implementada por la clase WebRatioReader.

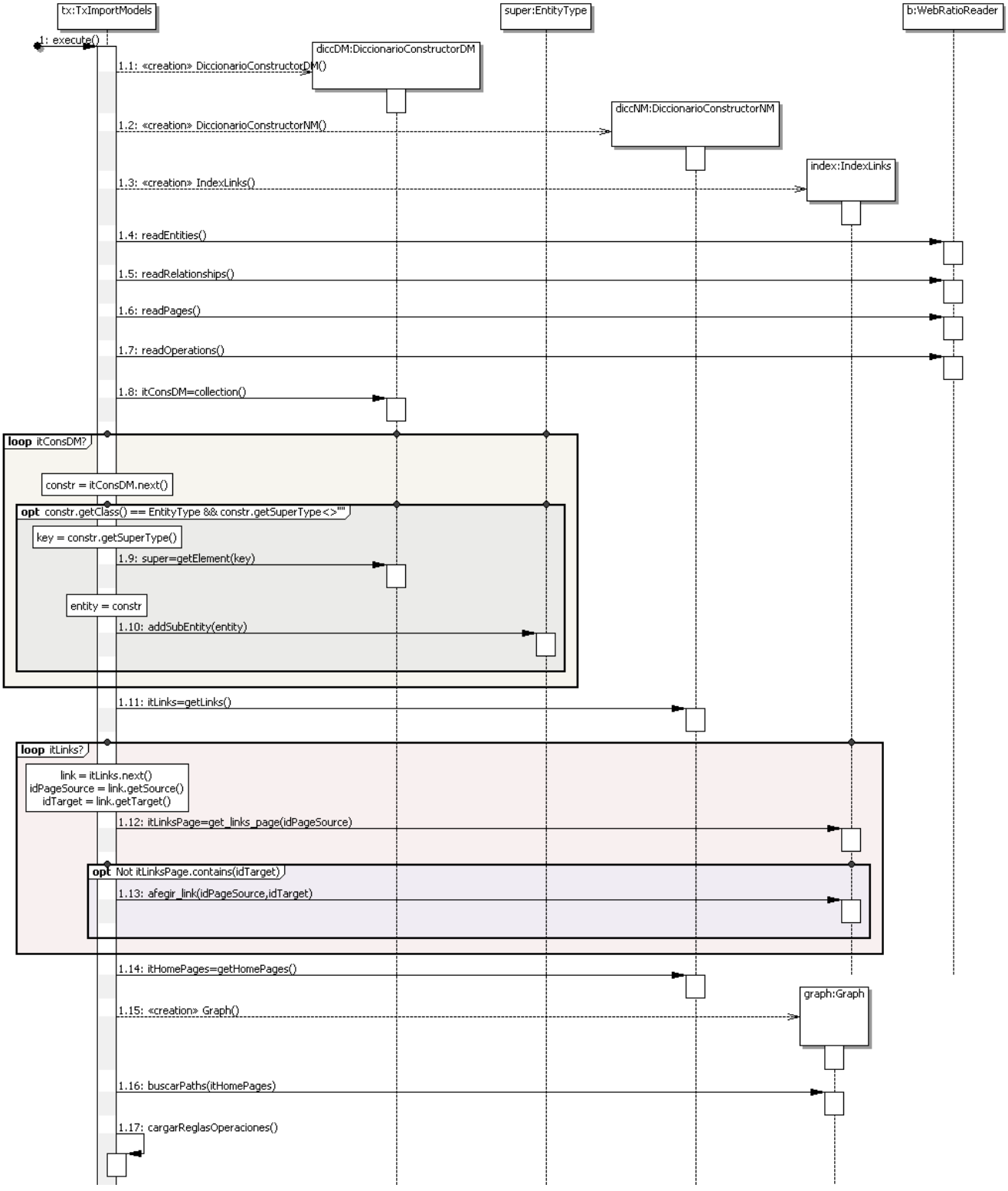
Para abrir los documentos con formato XML, se utilizan los objetos *DocumentBuilder* y *DocumentBuilderFactory* de la librería javax.xml.parsers, y para examinar el documento y su estructura se utilizarán clases de la librería org.w3c.dom. Todas estas librerías están incluidas en la versión de Java J2SE 5.0

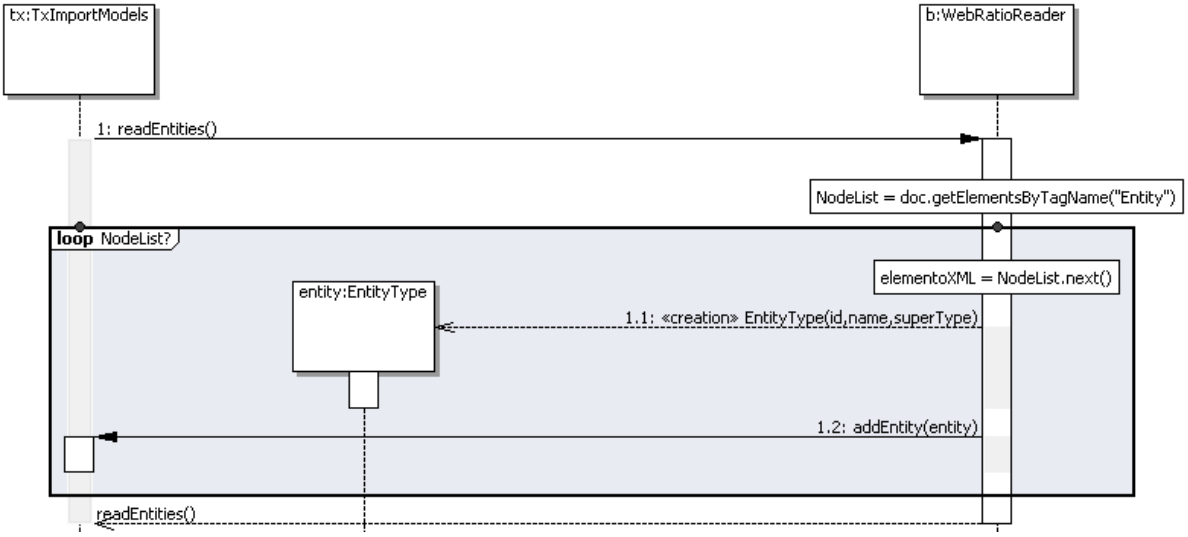


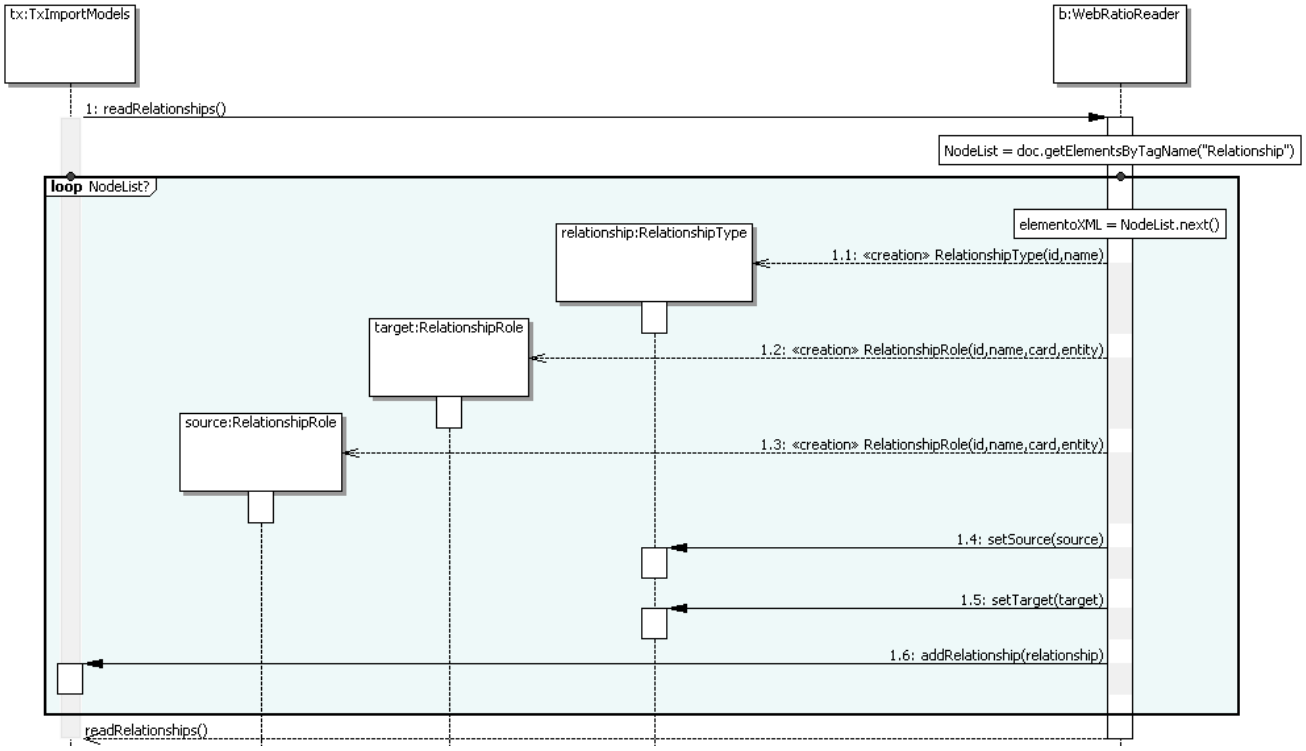
4.5.4 Diagramas de secuencia

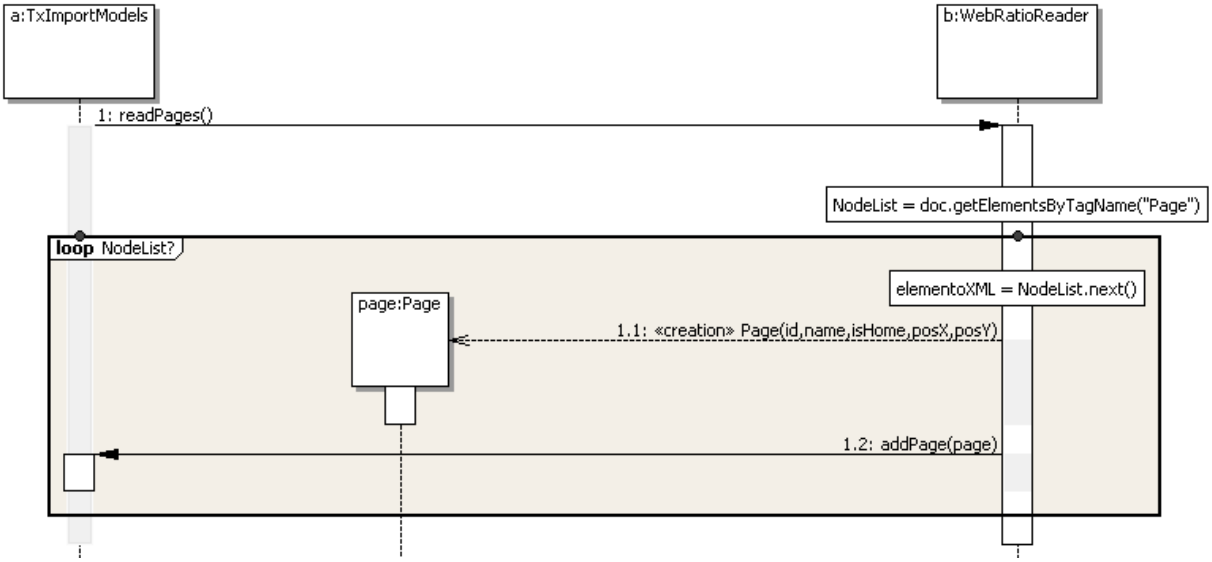
A partir de las diferentes clases definidas en las diferentes capas se procederá a definir la interacción entre las diferentes capas y clases incluidas en ellas mediante los diagramas de secuencias. Se definirá un diagrama para cada caso de uso.

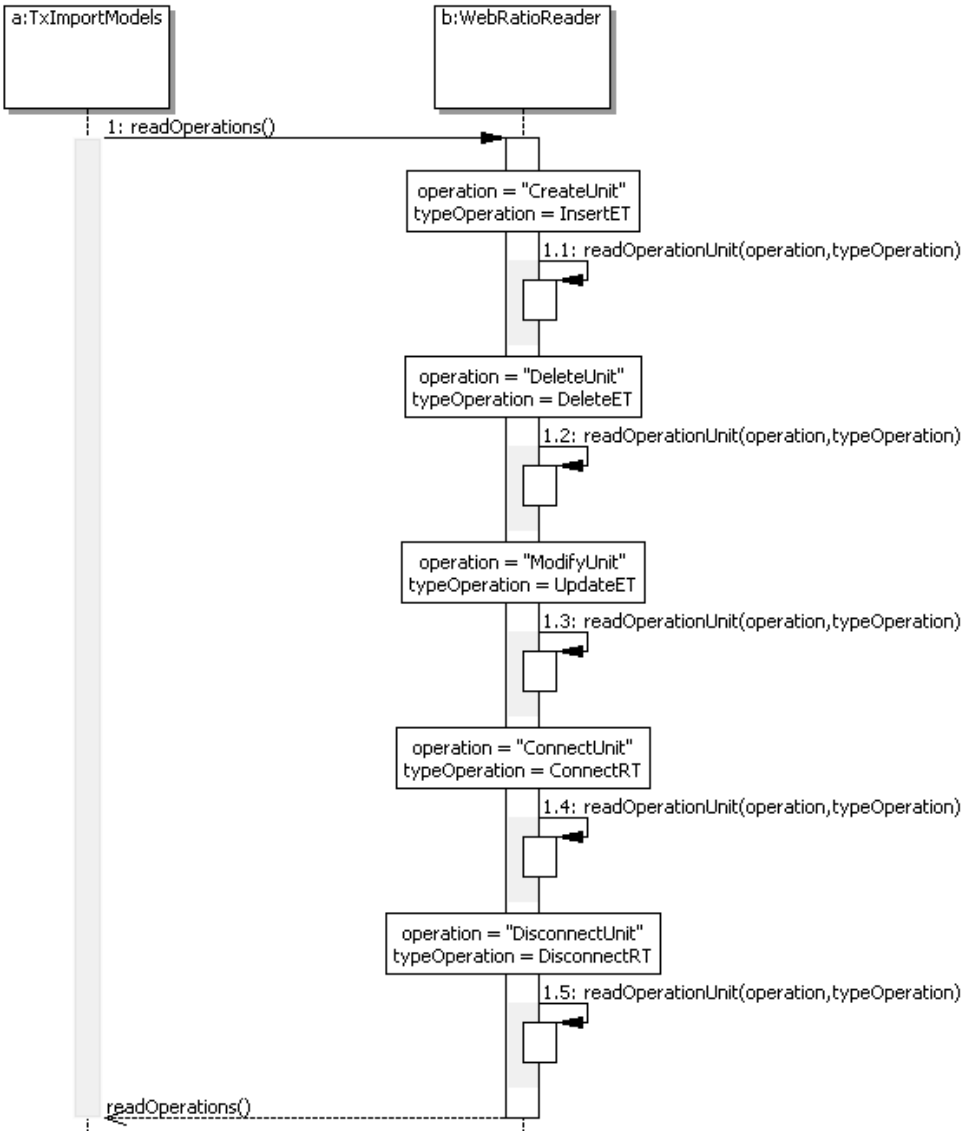
4.5.4.1 Caso de uso Importar Modelos

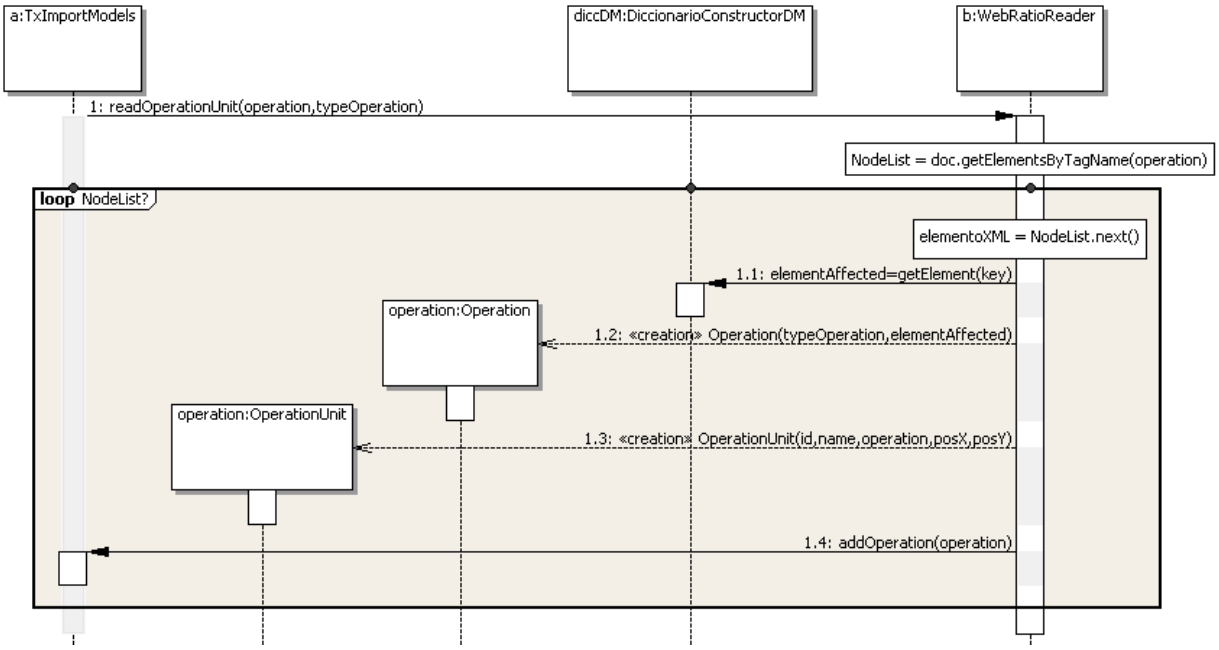


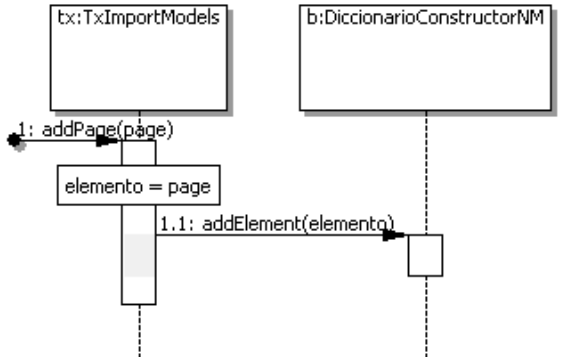

tx:TxImportModels
b:WebRatioReader
1: readEntities()
NodeList = doc.getElementsByTagName("Entity")
loop NodeList?
elementoXML = NodeList.next()
entity:EntityType
1.1: «creation» EntityType(id,name,superType)
1.2: addEntity(entity)
readEntities()

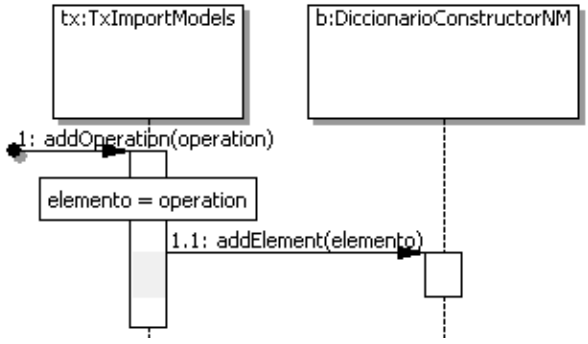

tx:TxImportModels
b:WebRatioReader
1: readRelationships()
NodeList = doc.getElementsByTagName("Relationship")
loop NodeList?
elementoXML = NodeList.next()
relationship:RelationshipType
1.1: «creation» RelationshipType(id,name)
target:RelationshipRole
1.2: «creation» RelationshipRole(id,name,card,entity)
source:RelationshipRole
1.3: «creation» RelationshipRole(id,name,card,entity)
1.4: setSource(source)
1.5: setTarget(target)
1.6: addRelationship(relationship)
readRelationships()

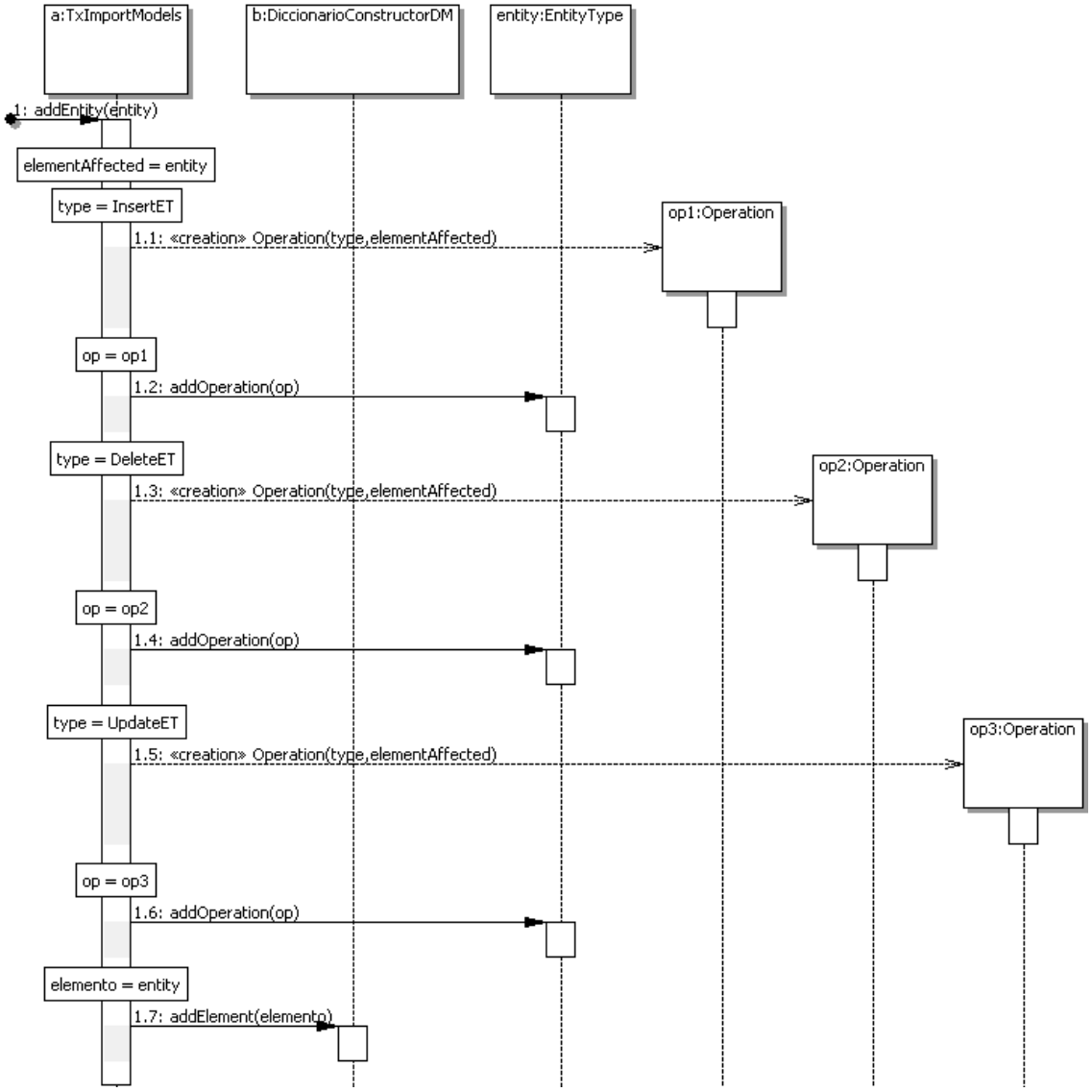

a:TxImportModels
b:WebRatioReader
1: readPages()
NodeList = doc.getElementsByTagName("Page")
loop NodeList?
elementoXML = NodeList.next()
page:Page
1.1: «creation» Page(id,name,isHome,posX,posY)
1.2: addPage(page)

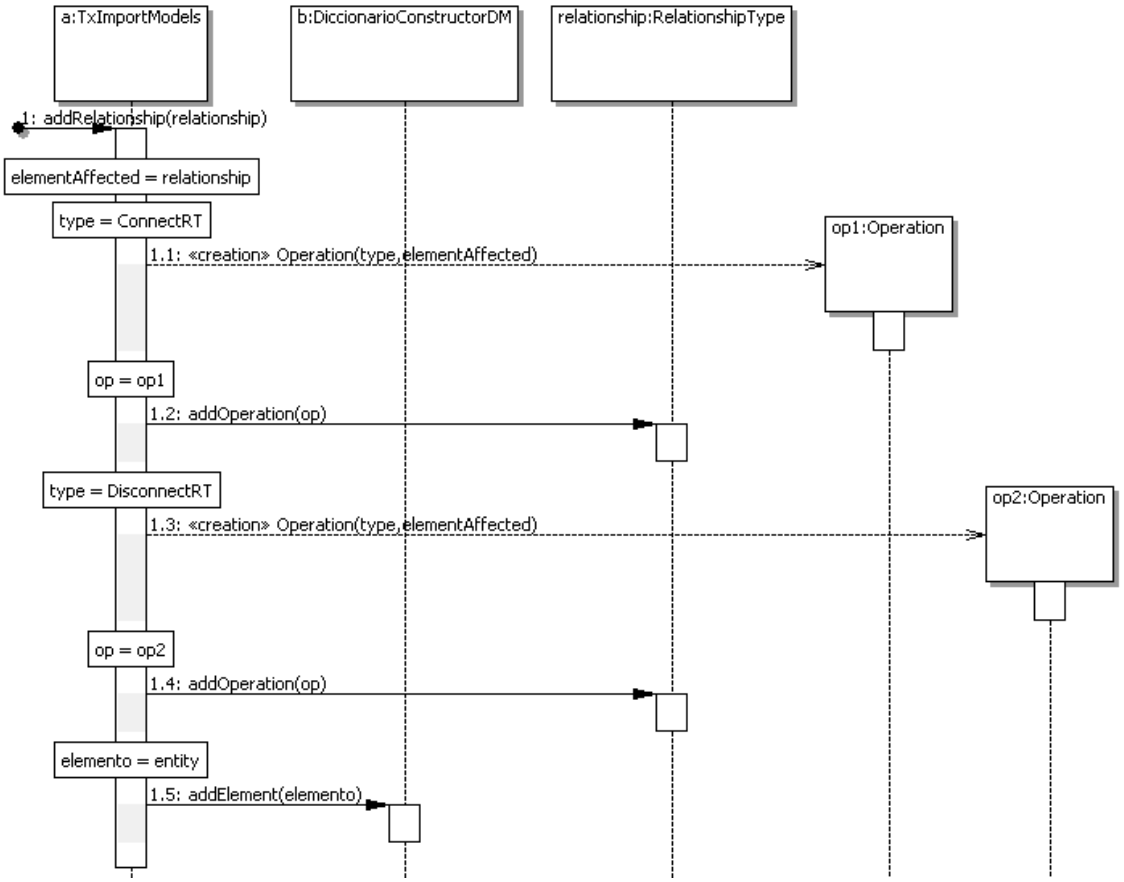

a:TxImportModels
b:WebRatioReader
1: readOperations()
operation = "CreateUnit"
typeOperation = InsertET
1.1: readOperationUnit(operation,typeOperation)
operation = "DeleteUnit"
typeOperation = DeleteET
1.2: readOperationUnit(operation,typeOperation)
operation = "ModifyUnit"
typeOperation = UpdateET
1.3: readOperationUnit(operation,typeOperation)
operation = "ConnectUnit"
typeOperation = ConnectRT
1.4: readOperationUnit(operation,typeOperation)
operation = "DisconnectUnit"
typeOperation = DisconnectRT
1.5: readOperationUnit(operation,typeOperation)
readOperations()


a:TxImportModels
diccDM:DiccionarioConstructorDM
b:WebRatioReader
1: readOperationUnit(operation,typeOperation)
NodeList = doc.getElementsByTagName(operation)
loop NodeList?
elementoXML = NodeList.next()
1.1: elementAffected=getElement(key)
operation:Operation
1.2: «creation» Operation(typeOperation,elementAffected)
operation:OperationUnit
1.3: «creation» OperationUnit(id,name,operation,posX,posY)
1.4: addOperation(operation)

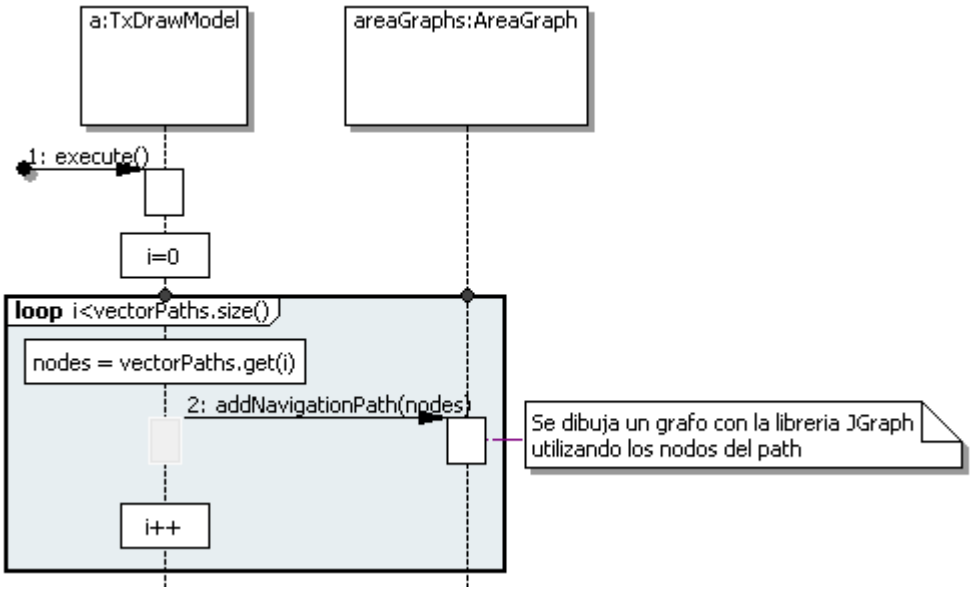

tx:TxImportModels
b:DiccionarioConstructorNM
1: addPage(page)
elemento = page
1.1: addElement(elemento)


tx:TxImportModels
b:DiccionarioConstructorNM
1: addOperation(operation)
elemento = operation
1.1: addElement(elemento)

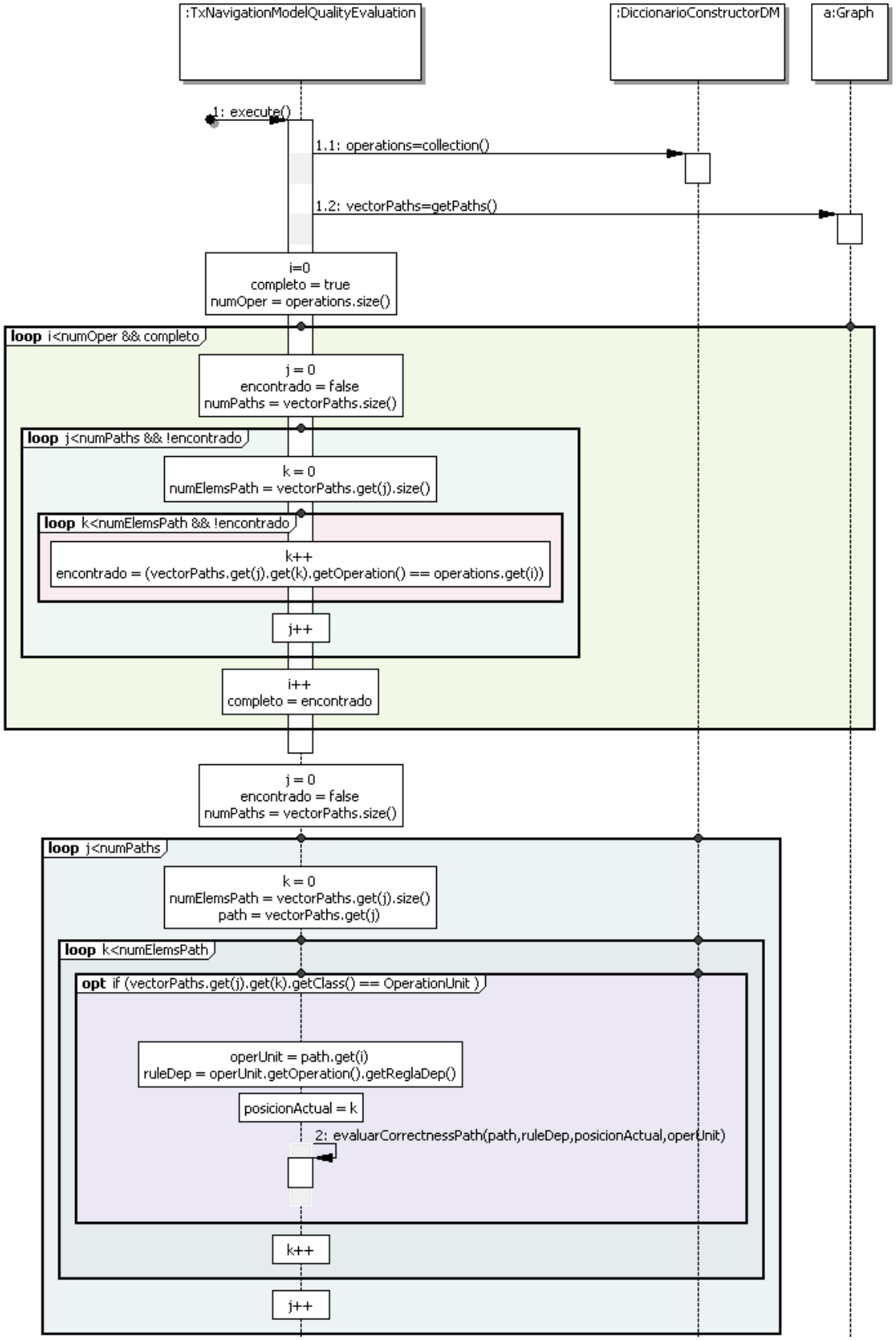

a:TxImportModels
b:DiccionarioConstructorDM
entity:EntityType
1: addEntity(entity)
elementAffected = entity
type = InsertET
1.1: «creation» Operation(type,elementAffected)
op1:Operation
op = op1
1.2: addOperation(op)
type = DeleteET
1.3: «creation» Operation(type,elementAffected)
op2:Operation
op = op2
1.4: addOperation(op)
type = UpdateET
1.5: «creation» Operation(type,elementAffected)
op3:Operation
op = op3
1.6: addOperation(op)
elemento = entity
1.7: addElement(elemento)

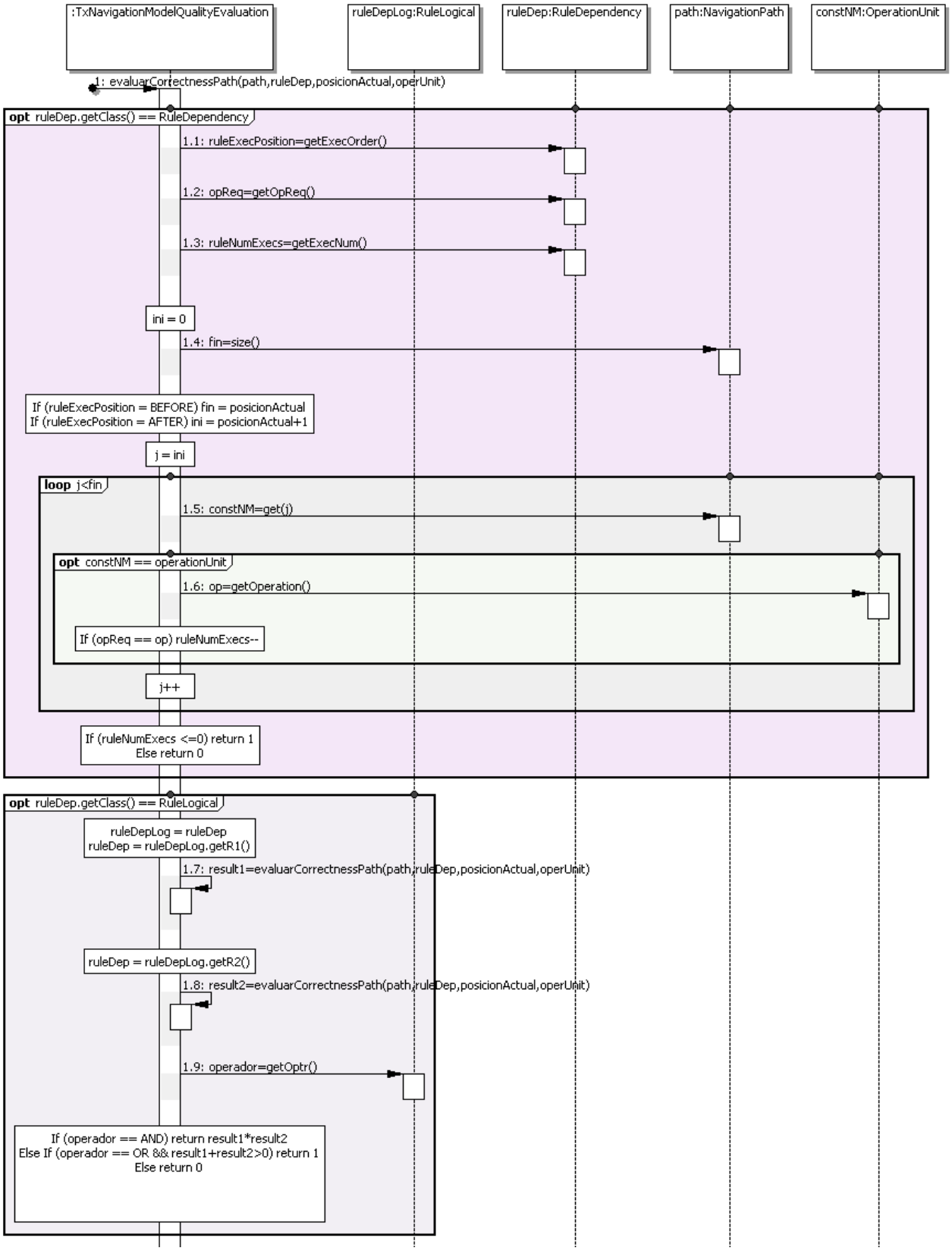


4.5.4.2 Caso de uso Visualizar las rutas navegacionales



4.5.4.3 Caso de uso Evaluación Cualitativa del Modelo Navegacional

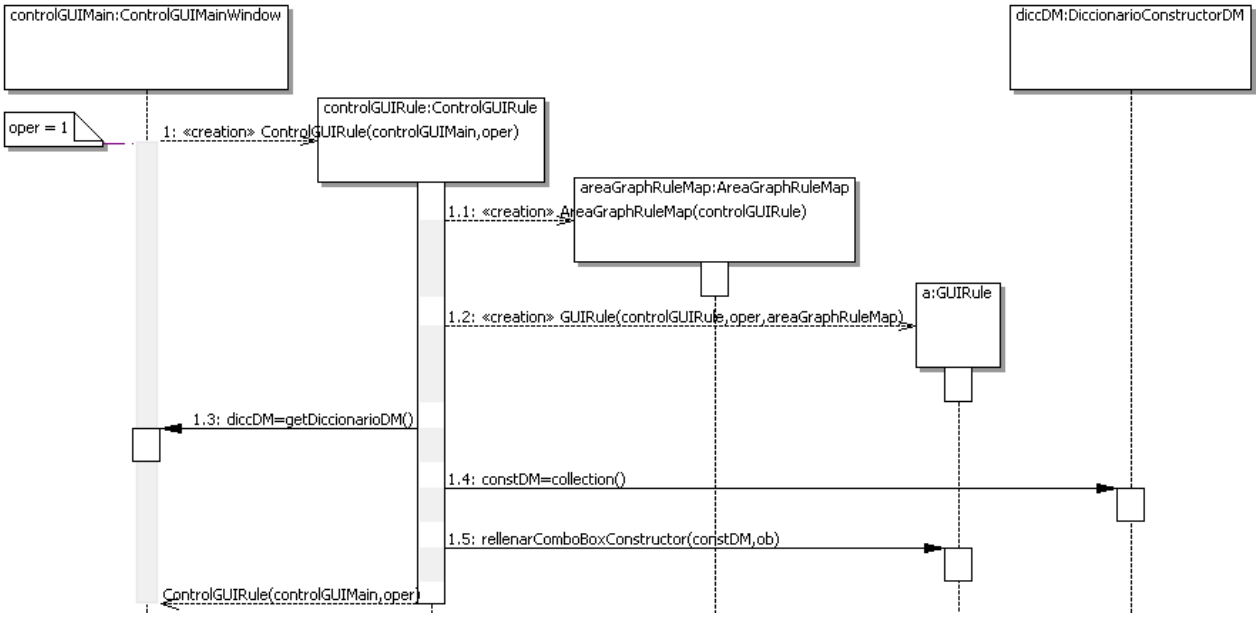



:TxNavigationModelQualityEvaluation
ruleDepLog:RuleLogical
ruleDep:RuleDependency
path:NavigationPath
constNM:OperationUnit
1: evaluarCorrectnessPath(path,ruleDep,posicionActual,operUnit)
opt ruleDep.getClass() == RuleDependency
1.1: ruleExecPosition=getExecOrder()
1.2: opReq=getOpReq()
1.3: ruleNumExecs=getExecNum()
ini = 0
1.4: fin=size()
If (ruleExecPosition = BEFORE) fin = posicionActual
If (ruleExecPosition = AFTER) ini = posicionActual+1
j = ini
loop j<fin
1.5: constNM=get(j)
opt constNM == operationUnit
1.6: op=getOperation()
If (opReq == op) ruleNumExecs--
j++
If (ruleNumExecs <=0) return 1
Else return 0
opt ruleDep.getClass() == RuleLogical
ruleDepLog = ruleDep
ruleDep = ruleDepLog.getR1()
1.7: result1=evaluarCorrectnessPath(path,ruleDep,posicionActual,operUnit)
ruleDep = ruleDepLog.getR2()
1.8: result2=evaluarCorrectnessPath(path,ruleDep,posicionActual,operUnit)
1.9: operador=getOptr()
If (operador == AND) return result1*result2
Else If (operador == OR && result1+result2>0) return 1
Else return 0

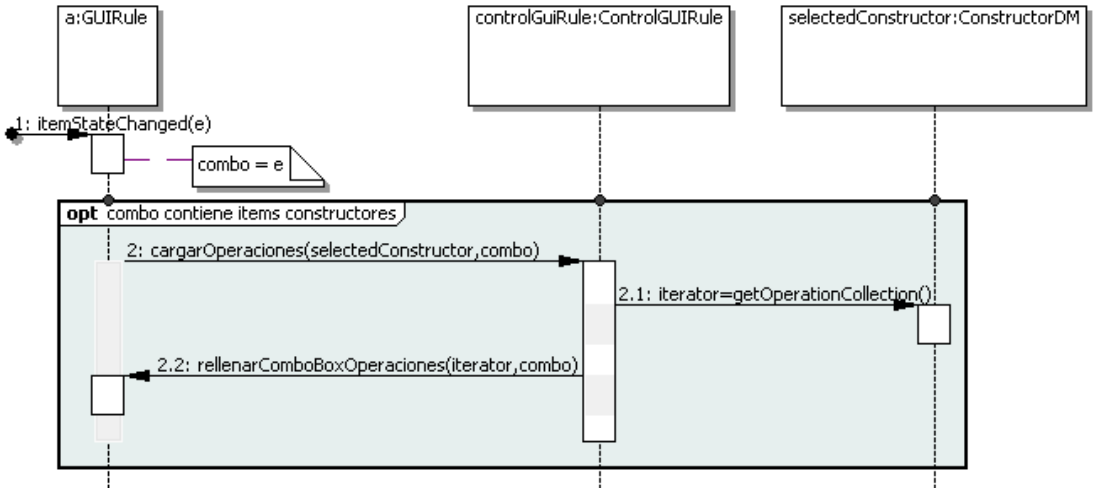
4.5.4.4 Gestión de las Reglas de Dependencia

Los casos de uso relacionados con las reglas de dependencia tienen un comportamiento común en la capa de presentación, ya que se aprovecha la interfaz gráfica de usuario y su controlador.

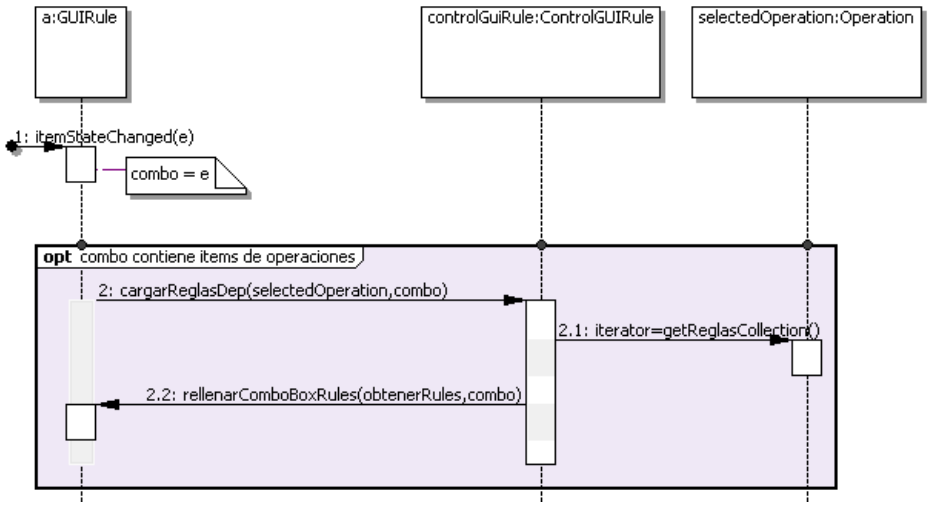
El usuario solicita una acción sobre las reglas de dependencia. Cuando oper = 1 el usuario ha solicitado una nueva regla, cuando es 2 solicita la edición y cuando es 3 está solicitando la eliminación de una regla.



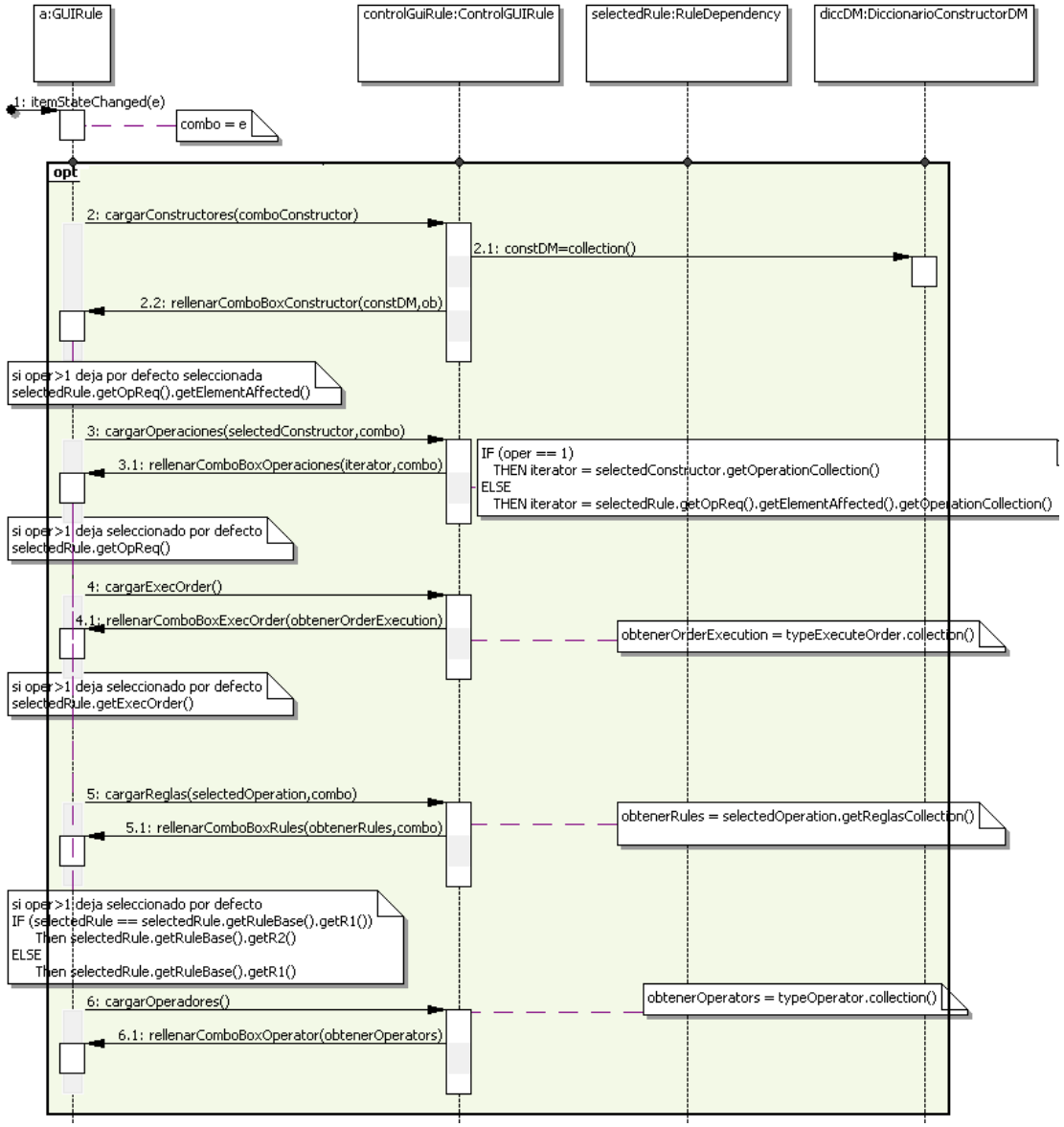
Cuando la ventana recibe un evento de un cambio en la selección de la lista desplegable de constructores, el sistema se comporta de la siguiente manera:



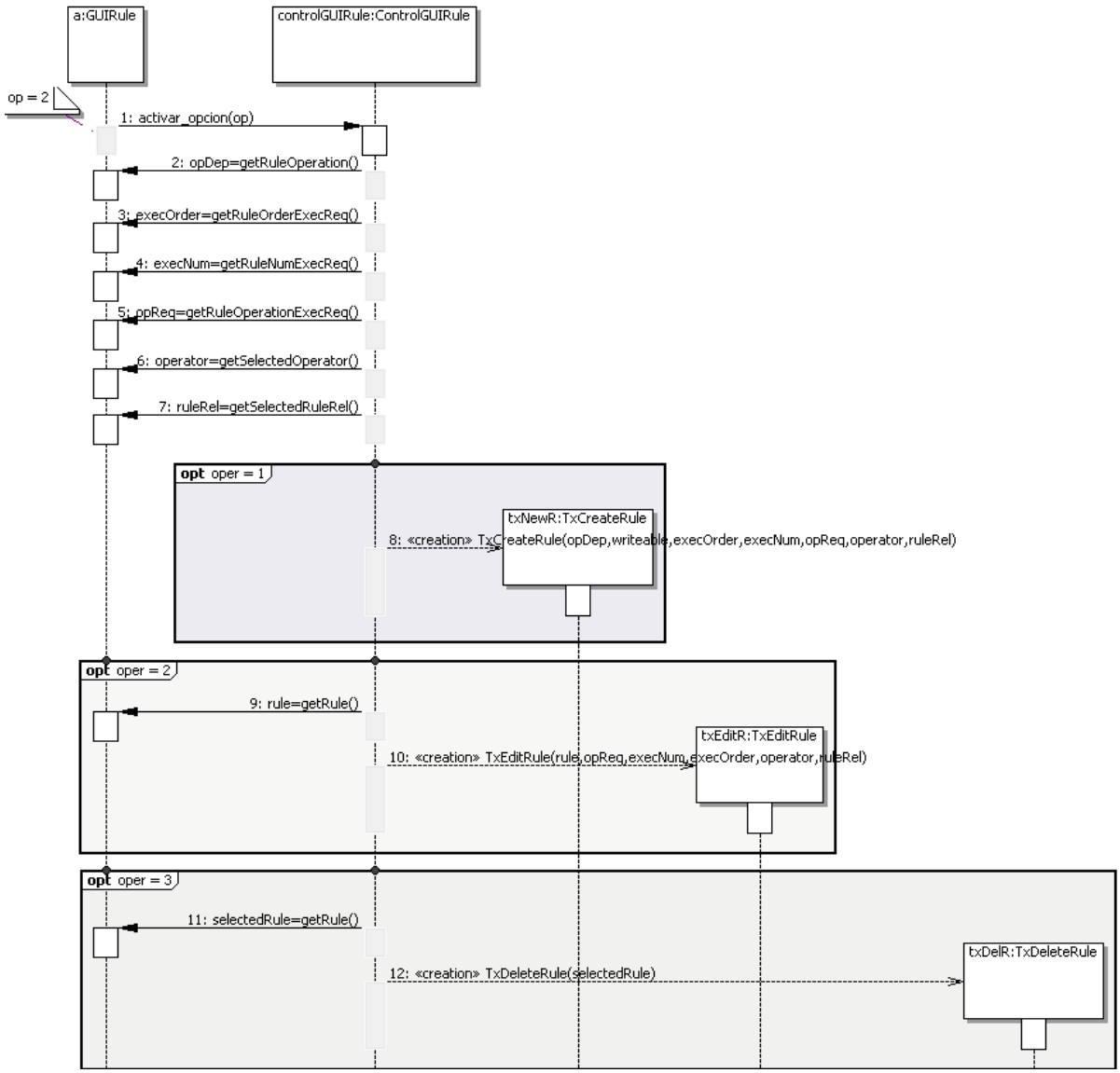
La siguiente secuencia sólo se producirá en el caso que *oper* >1, ya que se modificarán reglas existentes:



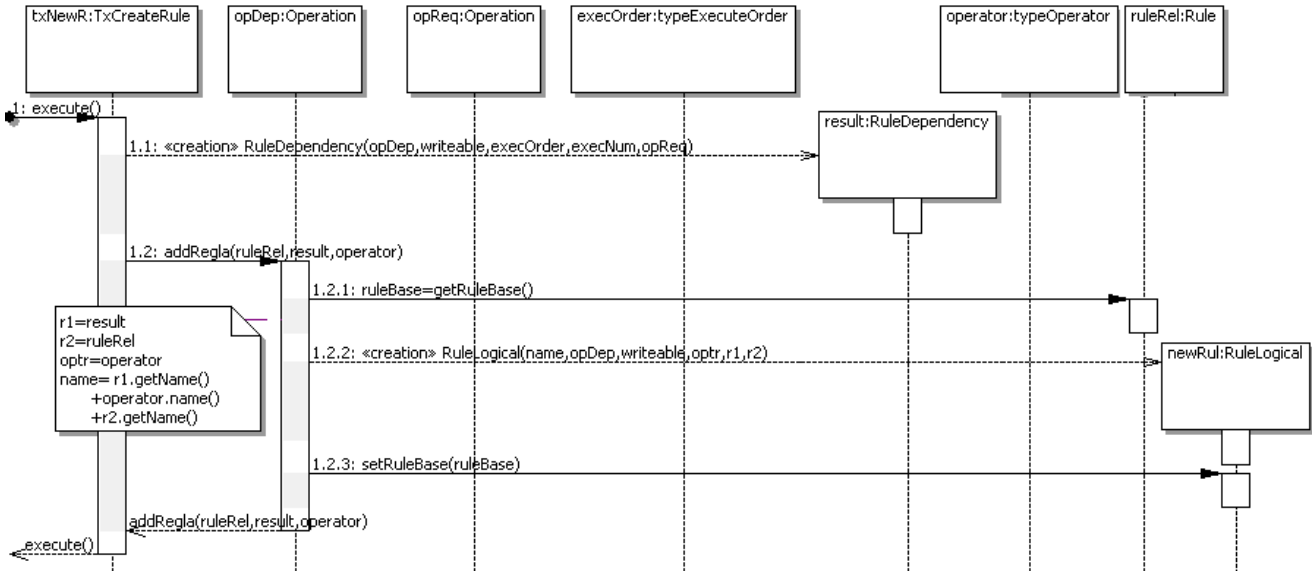
Cuando se ha seleccionado la regla en una edición o eliminación de dependencia (oper>1) o la operación en una nueva dependencia (oper=1):



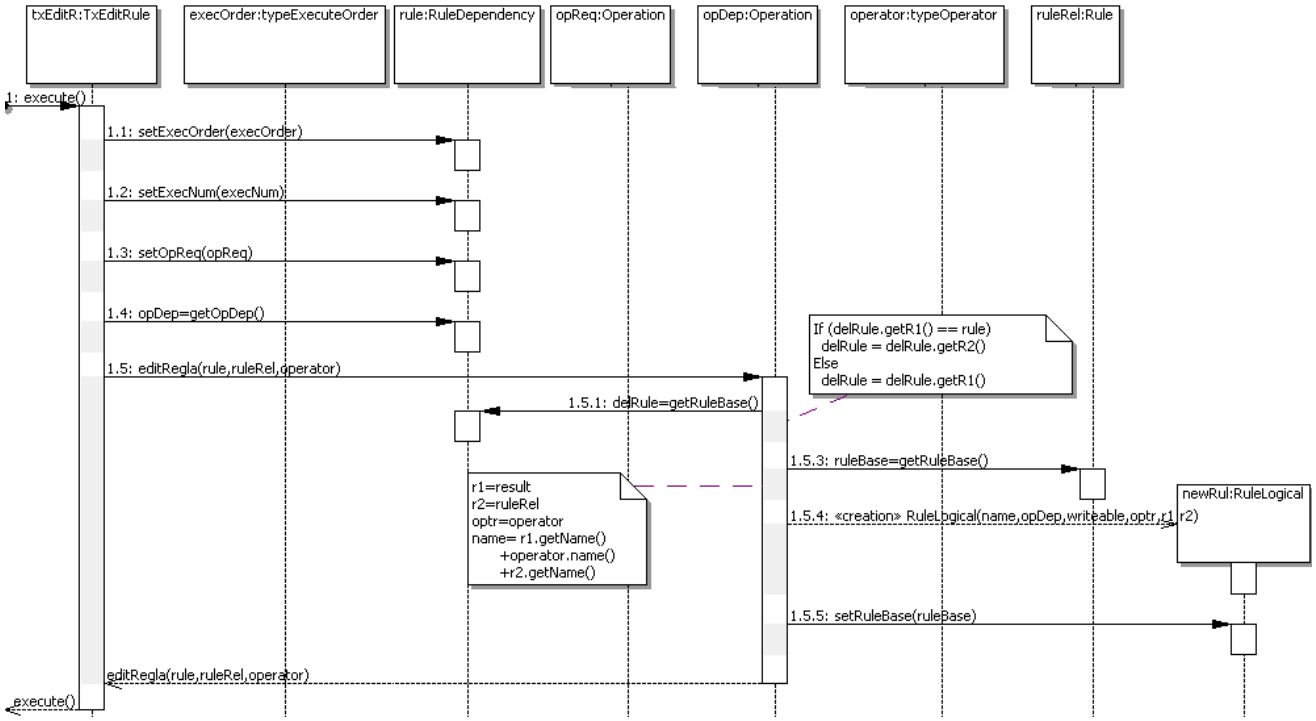
A continuación se muestra la interacción con los controladores de dominio.



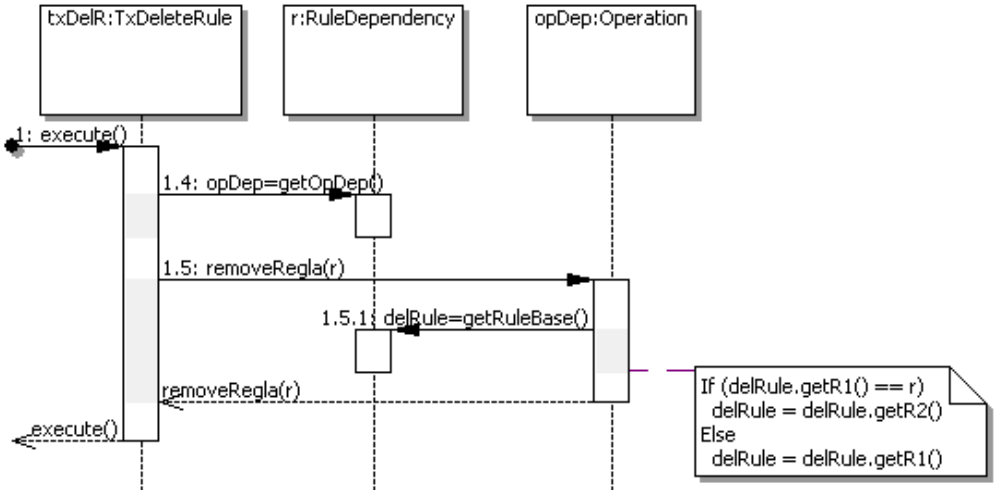
4.5.4.5 Caso de uso Nueva Regla de Dependencia



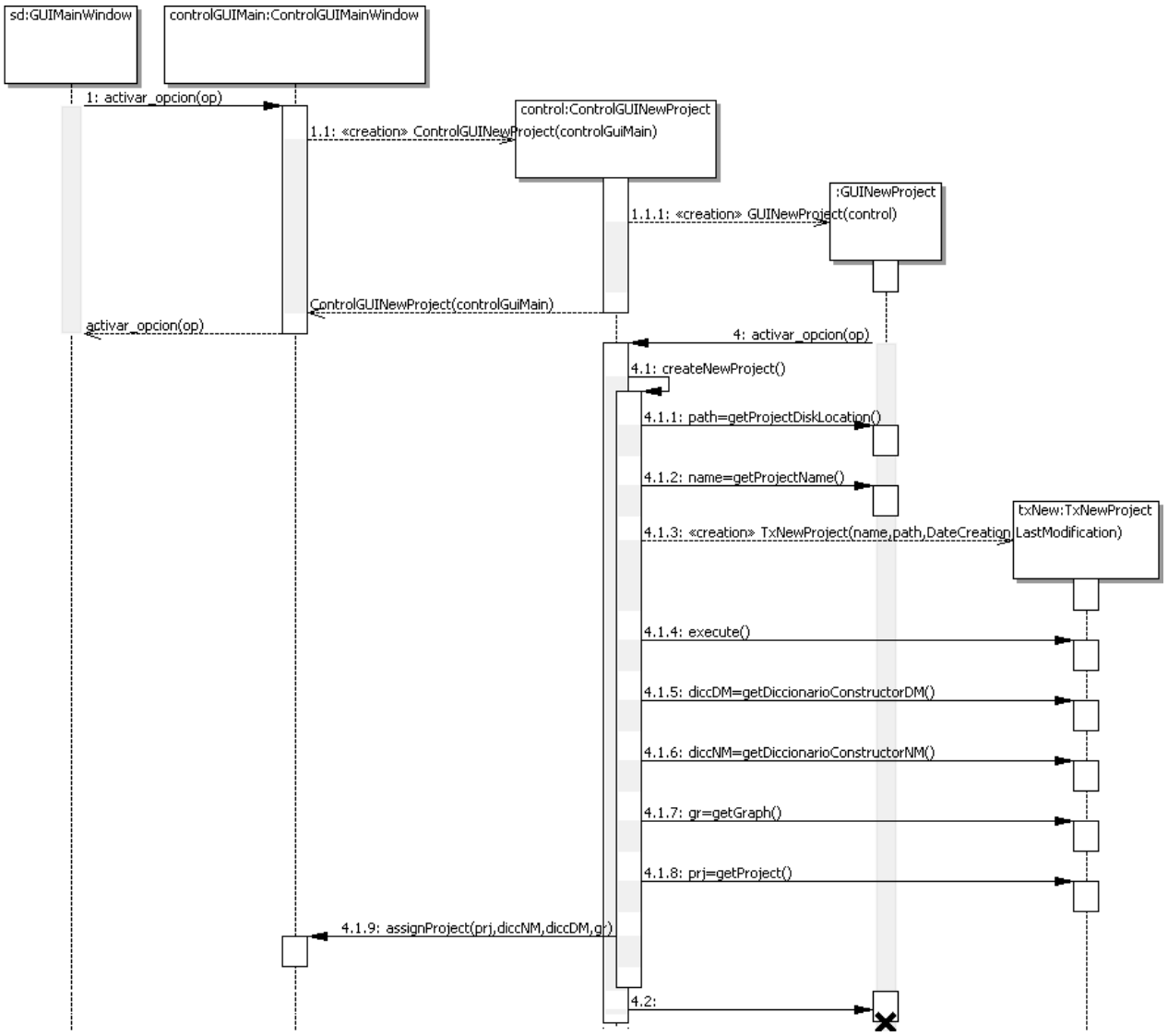
4.5.4.6 Caso de uso Editar Regla de Dependencia

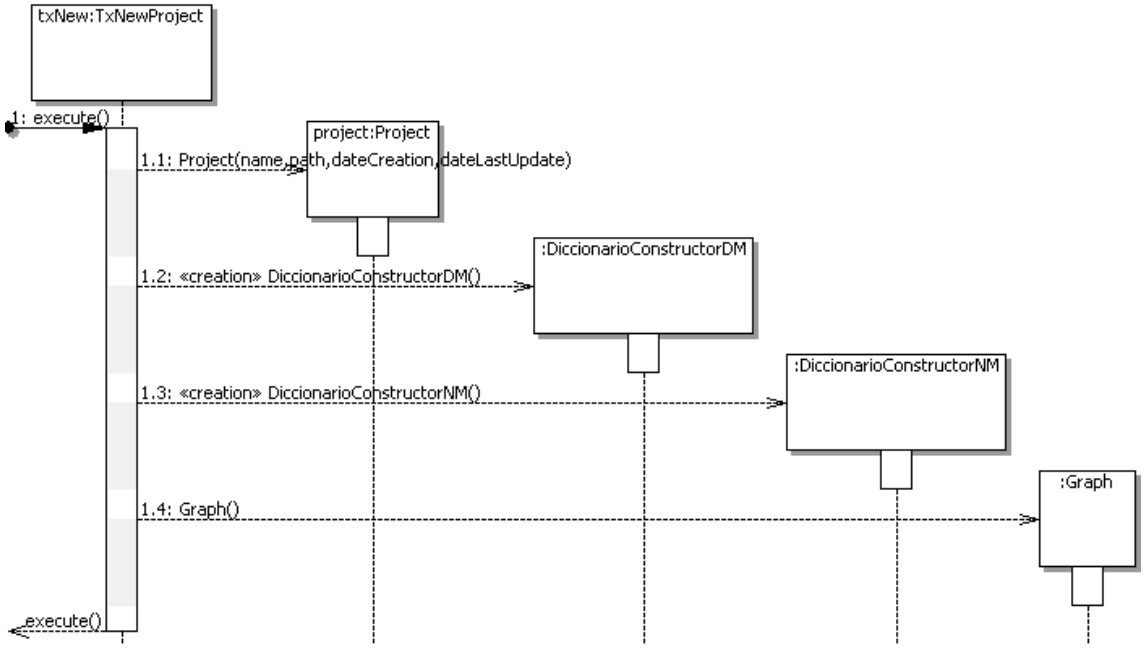


4.5.4.7 Caso de uso Eliminar Regla de Dependencia

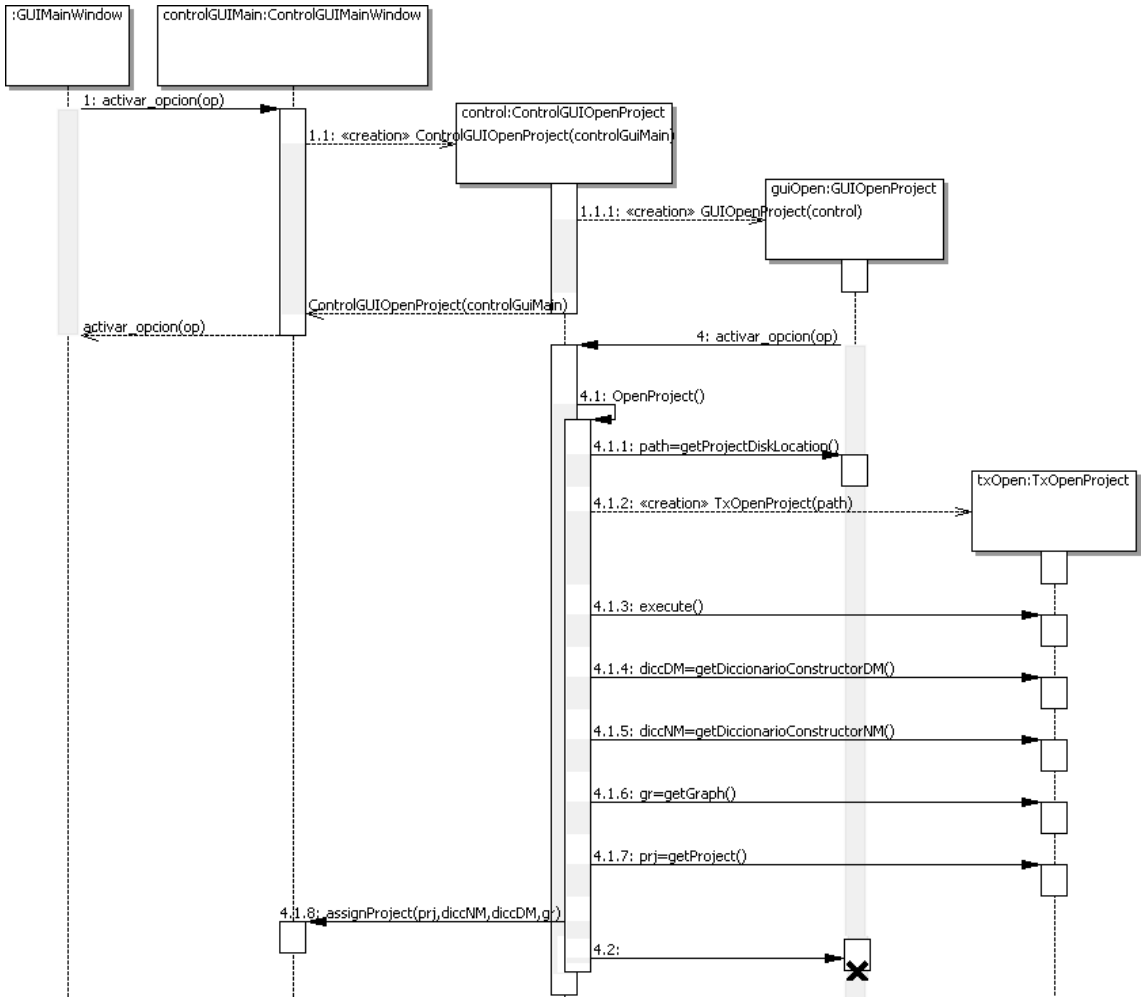


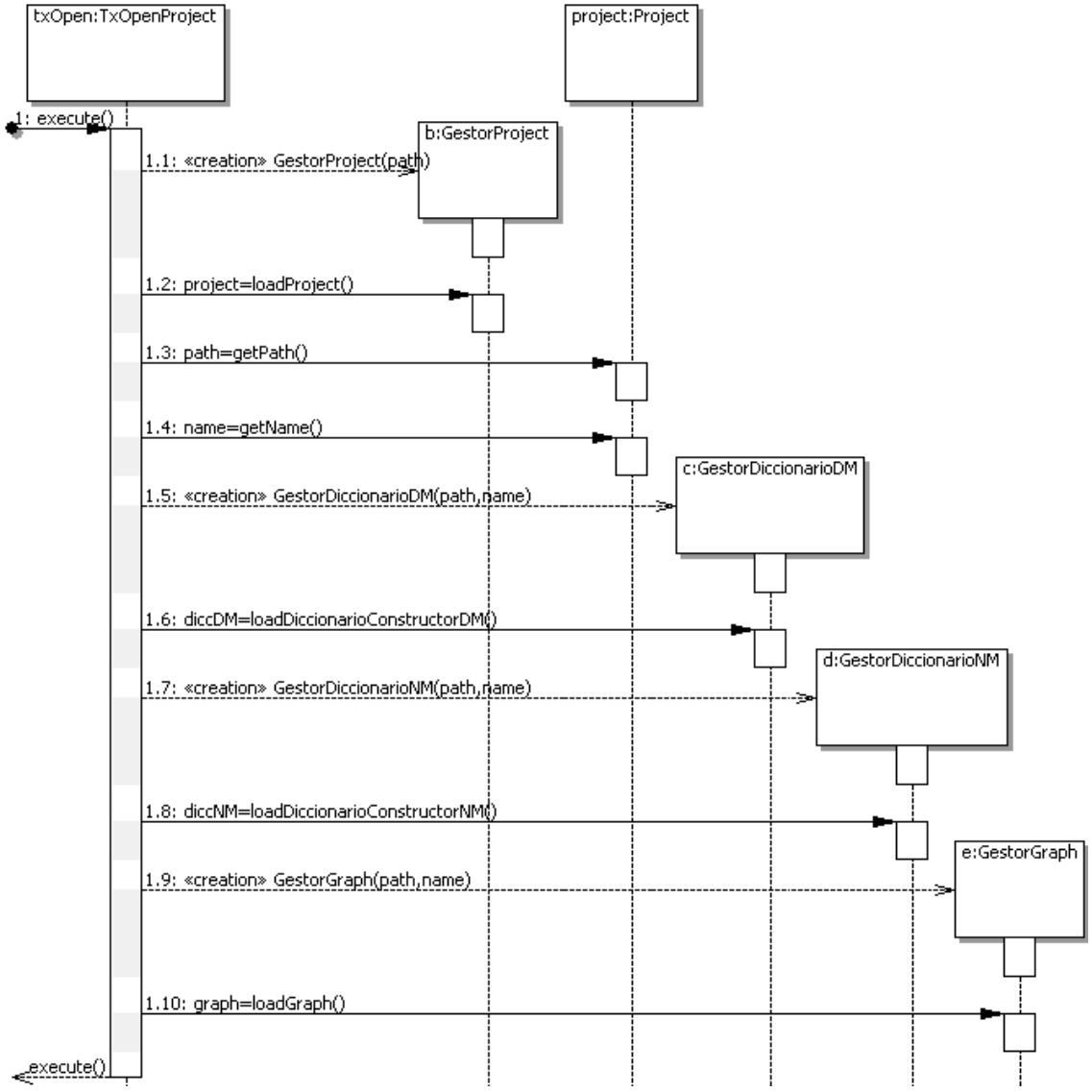
4.5.4.8 Caso de uso Nuevo Proyecto



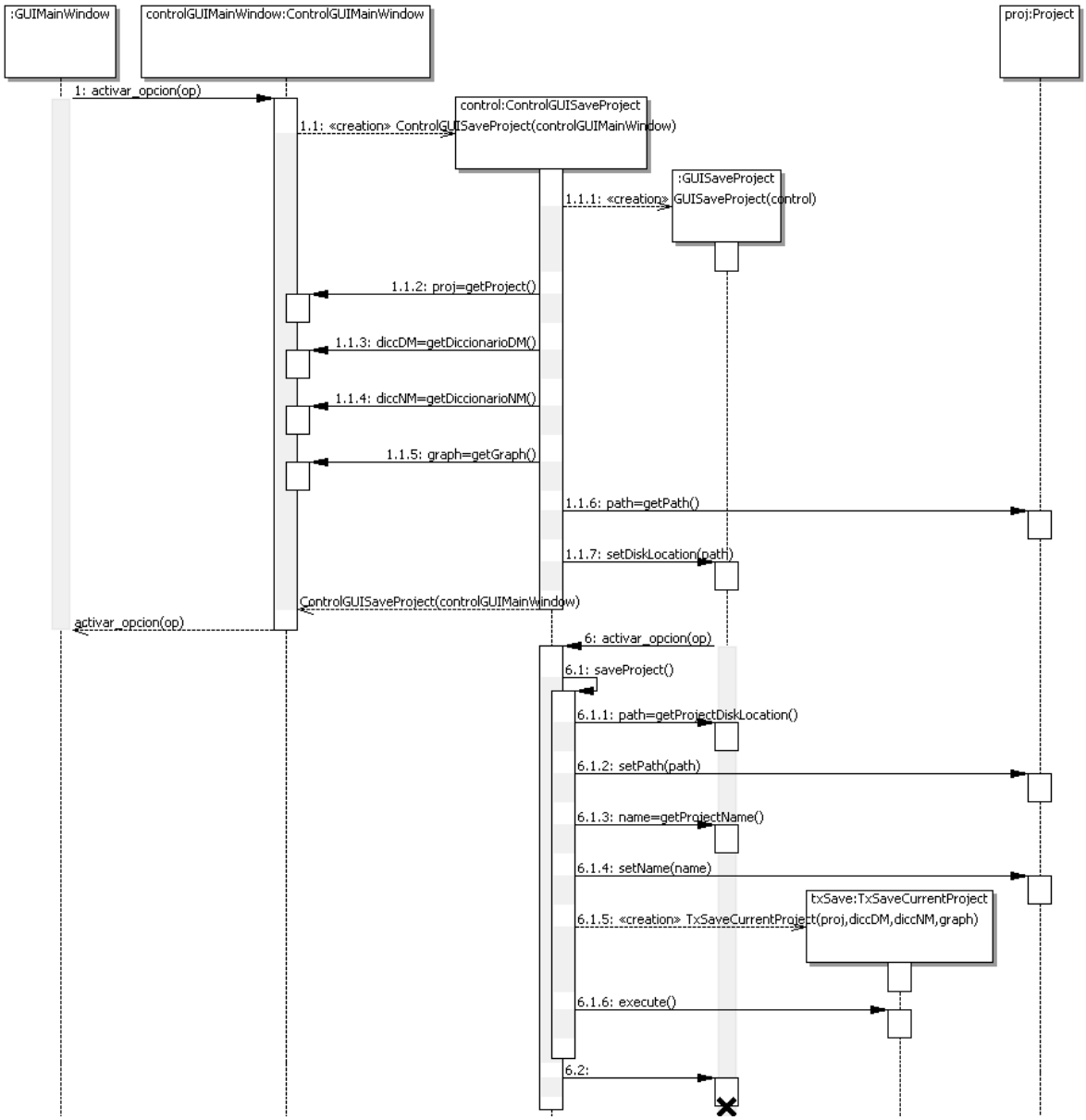


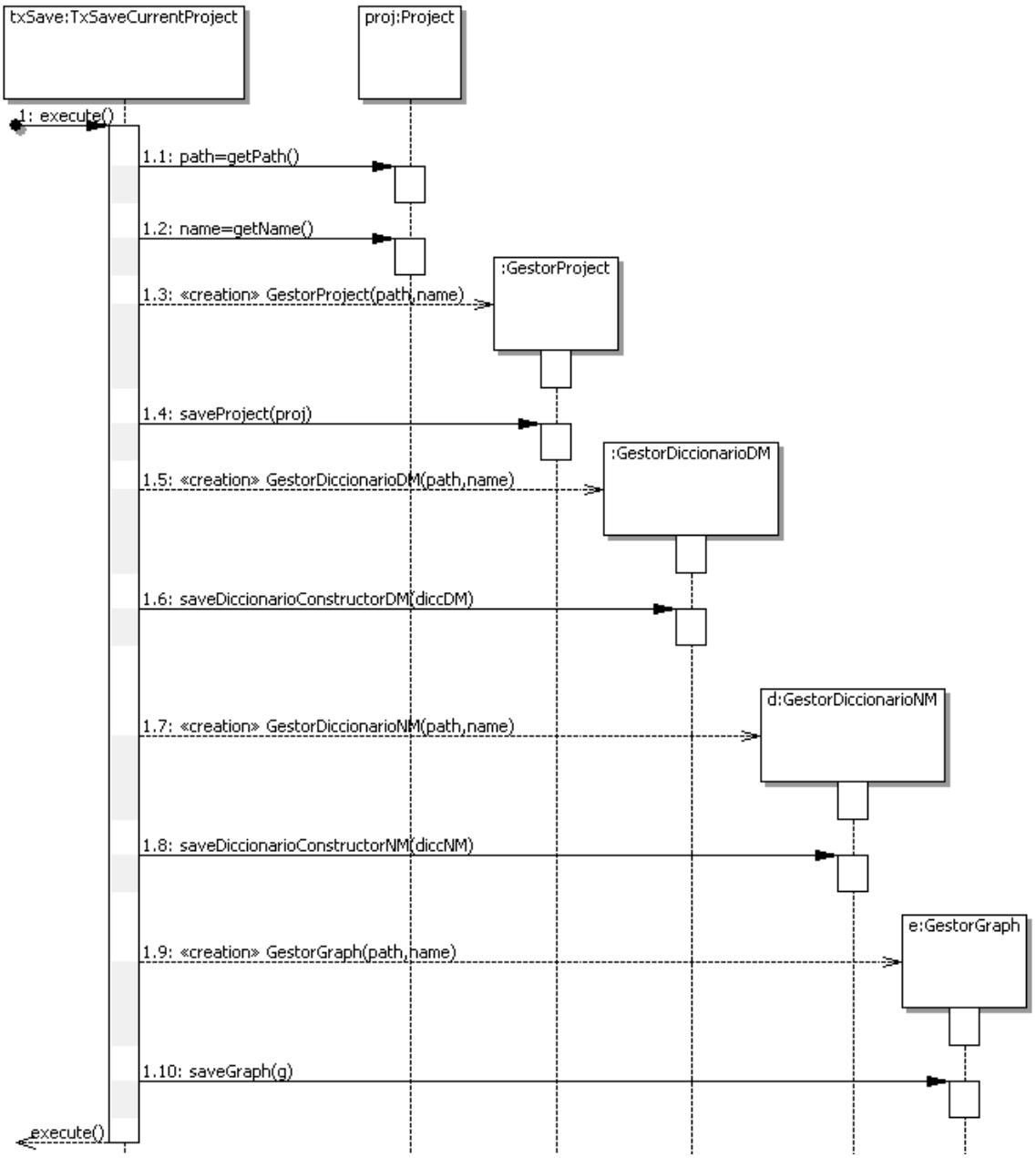
4.5.4.9 Caso de uso Abrir Proyecto



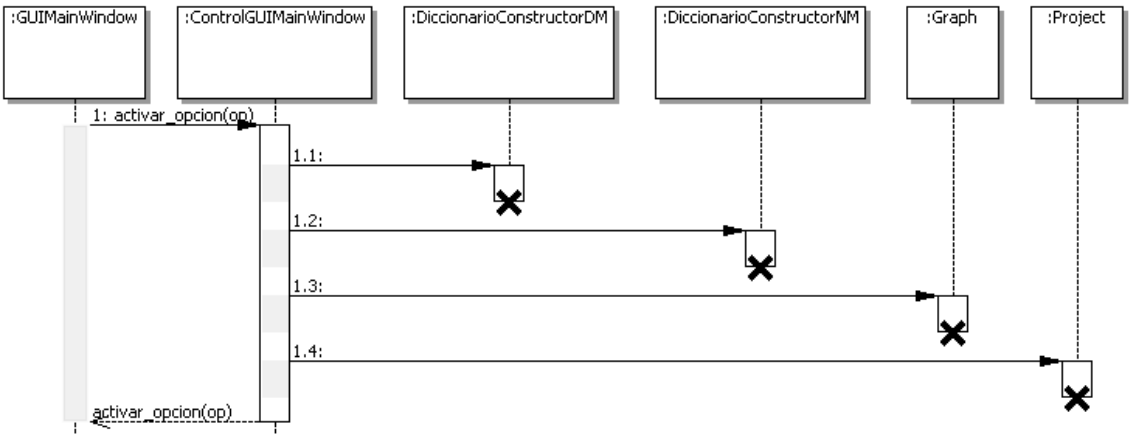

txOpen:TxOpenProject
project:Project
1: execute()
b:GestorProject
1.1: «creation» GestorProject(path)
1.2: project=loadProject()
1.3: path=getPath()
1.4: name=getName()
c:GestorDiccionarioDM
1.5: «creation» GestorDiccionarioDM(path,name)
1.6: diccDM=loadDiccionarioConstructorDM()
d:GestorDiccionarioNM
1.7: «creation» GestorDiccionarioNM(path,name)
1.8: diccNM=loadDiccionarioConstructorNM()
e:GestorGraph
1.9: «creation» GestorGraph(path,name)
1.10: graph=loadGraph()
execute()

4.5.4.10 Caso de uso Guardar Proyecto Actual





4.5.4.11 Caso de uso Cerrar Proyecto Actual



Capítulo 5. Implementación

5.1 Introducción

En este capítulo se detalla la tecnología utilizada a la hora de implementar el sistema. Se exponen los diferentes leguajes utilizados y las librerías externas utilizadas para llevar a cabo la programación del producto.

5.2 Lenguajes utilizados

5.2.1 Java

Java es un lenguaje de programación orientado a objetos. Creado por Sun Microsystems, es similar a C y C++, e incluso más simple que estos, ya que se eliminan ciertas características como son los punteros.

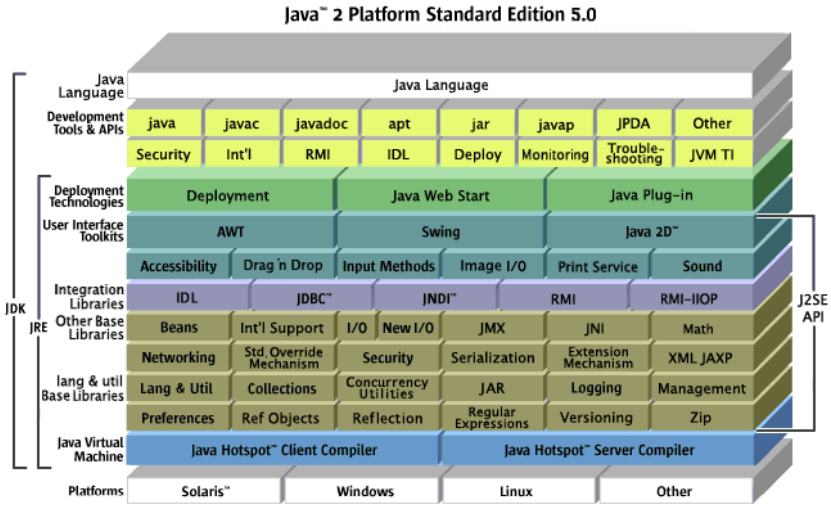
Las aplicaciones Java están compiladas en un código intermedio más abstracto que el código máquina, el bytecode, que se suele tratar como un fichero binario y compacto. Su finalidad es reducir la dependencia respecto del hardware y facilitar su interpretación. Durante el tiempo de ejecución es interpretado por una máquina virtual, que se encarga de analizar y traducir el bytecode al código máquina. De este modo, se facilita la portabilidad a diferentes plataformas y arquitecturas.

Otra de las ventajas de Java es el recolector de basura (*garbage collector*). Con este mecanismo, la propia máquina virtual gestiona la memoria disponible, reservando y liberando según en tiempo de ejecución. Por lo tanto, el programador no se tiene que preocupar de liberar la memoria ocupada por los objetos creados, tan sólo tiene que determinar el momento de creación de los objetos.

El sistema actual está desarrollado con la versión de Java J2SE 5.0.

5.3 Librerías

El entorno de desarrollo Java dispone de un conjunto de librerías muy potente, algunas de las cuales servirán para el desarrollo del sistema.



5.3.1 Swing

Swing es una biblioteca gráfica incluida en Java, que permite definir la apariencia externa de aplicaciones GUI (Graphical User Interface). Swing permite incluir en la interfaz interfaz gráfica de usuario objetos como botones, cuadros de texto, listas desplegables o menús, incluidos en contenedores.

Las interfaces de usuario de la aplicación, en la capa de presentación, se han desarrollado con esta librería. Al aprovechar las librerías integradas en la plataforma de Java, se mantiene la independencia de la plataforma.

5.3.2 JGraph

Para representar las rutas navegacionales se utilizará JGraph, una librería desarrollada en Java que permite representar gráficamente los grafos e interactuar con ellos. JGraph se integra perfectamente con Java Swing y requiere una versión de Java inferior a la requerida por el propio software.

Además de la representación, JGraph permite ordenar automáticamente la distribución de los vértices que forman el grafo utilizando estructuras de grafos preestablecidas implementadas por algoritmos de posicionamiento. Aunque esta funcionalidad no se utilizará, se debe tener en cuenta para el trabajo futuro.

5.3.3 Lectura de XML

Java J2SE 5.0 incluye una librería que permite manipular ficheros XML, **javax.xml.parsers**, que utiliza los objetos *DocumentBuilder* y *DocumentBuilderFactory* para abrir este tipo de documentos. Para examinar el XML y su estructura se utilizarán clases de la librería **org.w3c.dom**. Todas estas librerías se utilizan para poder importar los datos de WebRatio al propio sistema.

5.4 Ejemplos

A continuación se muestra la clase persistente *EntityType*:

```
import java.io.Serializable;
import java.util.Vector;

public class EntityType extends ConstructorDM implements Serializable
{
	private Vector<RelationshipType> relaciones;
	private Vector<EntityType> subEntities;
	private String idSuperEntity;

	public EntityType(String id, String name, String superType)
	{
		super(id, name);
		this.idSuperEntity = superType;
		this.relaciones = new Vector<RelationshipType>();
		this.subEntities = new Vector<EntityType>();
	}
	public Vector<RelationshipType> getRelacionesCollection()
	{
		return this.relaciones;
	}
	public void addRelacion(RelationshipType r)
	{
		this.relaciones.add(r);
	}
	public Vector<EntityType> getSubEntities()
	{
		return this.subEntities;
	}
	public void addSubEntity(EntityType entity)
	{
		this.subEntities.add(entity);
	}
	public String getSuperType()
	{
		return this.idSuperEntity;
	}
	public void setSuperType(String superEntity)
	{
		this.idSuperEntity = superEntity;
```

```
}}
```

Como ejemplo de implementación de una GUI, a continuación se muestra la clase *GUIOpenProject*:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class GUIOpenProject extends JDialog implements ActionListener
{
	private JPanel jPanel1;
	private JButton jButton4;
	private JButton jButton3;
	private JButton jButton2;
	private JTextField jTextField2;
	private JLabel jLabel3;
	private JLabel jLabel4;
	private JLabel jLabel5;
	private JSeparator jSeparator2;
	private JSeparator jSeparator1;
	private JLabel jLabel1;

	private ControlGUIOpenProject controlador;

	public GUIOpenProject(ControlGUIOpenProject control)
	{
		super(new JFrame());
		this.controlador = control;
		prepare();
		view();
	}

	private void prepare()
	{
		try
		{
			this.setTitle("Open Project");
			this.setResizable(false);
			this.setAlwaysOnTop(true);
			{
				jPanel1 = new JPanel();

				getContentPane().add(jPanel1,
BorderLayout.CENTER);
				jPanel1.setPreferredSize(new
java.awt.Dimension(380, 213));
				{
					jLabel1 = new JLabel();
					jPanel1.add(jLabel1);
					jLabel1.setText("Open Project");
```

```
                              jLabel1.setPreferredSize(new
java.awt.Dimension(359, 33));
                              jLabel1.setFont(new
java.awt.Font("Tahoma",1,16));

      jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
                              jLabel1.setBackground(new
java.awt.Color(192,192,192));
                        }
                        {
                              jSeparator1 = new JSeparator();
                              jPanel1.add(jSeparator1);
                              jSeparator1.setPreferredSize(new
java.awt.Dimension(340, 15));
                        }
                        {
                              jLabel3 = new JLabel();
                              jPanel1.add(jLabel3);
                              jLabel3.setText("Select project
source:");

      jLabel3.setVerticalAlignment(SwingConstants.BOTTOM);
                              jLabel3.setPreferredSize(new
java.awt.Dimension(340,20));
                        }
                        {
                              jTextField2 = new JTextField();
                              jPanel1.add(jTextField2);
                              jTextField2.setPreferredSize(new
java.awt.Dimension(256,21));
                        }
                        {
                              jButton2 = new JButton();
                              jPanel1.add(jButton2);
                              jButton2.setText("Browse");
                              jButton2.setFont(new
java.awt.Font("Tahoma",1,11));
                              jButton2.setPreferredSize(new
java.awt.Dimension(82,21));
                              jButton2.addActionListener(this);
                        }
                        {
                              jLabel4 = new JLabel();
                              jPanel1.add(jLabel4);
                              jLabel4.setText("");

      jLabel4.setVerticalAlignment(SwingConstants.BOTTOM);
                              jLabel4.setPreferredSize(new
java.awt.Dimension(340,20));
                        }
                        {
                              jSeparator2 = new JSeparator();
                              jPanel1.add(jSeparator2);
                              jSeparator2.setPreferredSize(new
java.awt.Dimension(340, 15));

      jSeparator2.setBorder(BorderFactory.createCompoundBorder(
                                          null,
                                          null));
                        }
```

```
                        {
                              jButton3 = new JButton();
                              jPanel1.add(jButton3);
                              jButton3.setText("Cancel");
                              jButton3.setFont(new
java.awt.Font("Tahoma",1,11));
                              jButton3.setPreferredSize(new
java.awt.Dimension(150, 21));
                              jButton3.addActionListener(this);
                        }
                        {
                              jLabel5 = new JLabel();
                              jPanel1.add(jLabel5);
                              jLabel5.setText("");

      jLabel5.setVerticalAlignment(SwingConstants.BOTTOM);
                              jLabel5.setPreferredSize(new
java.awt.Dimension(13, 20));
                        }
                        {
                              jButton4 = new JButton();
                              jPanel1.add(jButton4);
                              jButton4.setText("Open Project");
                              jButton4.setFont(new
java.awt.Font("Tahoma",1,11));
                              jButton4.setPreferredSize(new
java.awt.Dimension(150,21));
                              jButton4.addActionListener(this);
                        }
                  }
                  pack();

            //Centrar la pantalla
            Toolkit tk = Toolkit.getDefaultToolkit();
                Dimension screenSize = tk.getScreenSize();
                setLocation((screenSize.width - this.getWidth()) / 2,
(screenSize.height - this.getHeight())/ 2);

            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
      }

      private void view()
      {
            setVisible(true);
      }

      public void actionPerformed(ActionEvent e)
      {
            int opcion = 0;
            Object source = e.getSource();

            if (source == jButton2)
                  opcion = 1;
            if (source == jButton3)
                  opcion = 2;
            if (source == jButton4)
```

```
                    opcion = 3;

              if (opcion > 0)
                    controlador.activar_opcion(opcion);
        }

        public void setDiskLocation(String path)
        {
              jTextField2.setText(path);
        }

        public String getProjectDiskLocation()
        {
              return jTextField2.getText();
        }

        public void close()
        {
              dispose();
        }
}
```

Por último se muestra la clase *Gestor* de la capa de datos, que se encarga de escribir y leer datos del disco:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Gestor
{
        private String pathDades;
        private String path;

        public Gestor (String path, String filename)
        {
              this.path = path;
              char ultimCar = path.charAt (path.length()-1);
              if ((ultimCar == '/') || (ultimCar == '\\'))
              {
                    pathDades = path + filename;
              }
              else
              {
                    pathDades = path + '\\' + filename;
              }
        }

        public Gestor (String path)
        {
              this.pathDades = path;
              this.path = path;
        }
```

```
	public Object openFile() throws Exception
	{
		try
		{
			FileInputStream fileInput = new FileInputStream
(this.get_path_dades());
			ObjectInputStream objInput = new ObjectInputStream
(fileInput);
			return objInput.readObject();
		}
		catch (FileNotFoundException e)
		{
			return e;
		}
		catch (Exception e)
		{
			return e;
		}
	}

	public boolean saveFile (Object obj) throws Exception
	{
		try
		{
			File pathvalid = new File(this.path);
			if(!pathvalid.exists()) pathvalid.mkdirs();

			FileOutputStream fileOutPut = new FileOutputStream
(this.get_path_dades());
			ObjectOutputStream objOutput = new
ObjectOutputStream (fileOutPut);
			objOutput.writeObject (obj);
			return true;

		}
		catch (Exception e)
		{
			return false;
		}
	}

	public String get_path_dades()
	{
		return pathDades;
	}

	public boolean set_path_dades (String path)
	{
		pathDades = path;
		File pathvalid = new File(path);
		if(!pathvalid.exists()) pathvalid.mkdirs();
		return true;
	}
}
```

Capítulo 6. Pruebas

Antes de que el usuario pueda usar el software desarrollado, se deben llevar a cabo una serie de procesos o pruebas para garantizar que el sistema está libre de errores de implementación, calidad o usabilidad. En el caso que se detecten errores, se puede volver a fases anteriores para corregirlos.

Las pruebas que se han llevado a cabo en la elaboración del software son las siguientes:

- **Pruebas unitarias**: Son pruebas que se hacen individualmente a cada clase implementada en el sistema. El proceso consiste en crear una clase que se encargará de testear la clase ejecutando los métodos públicos del objeto, utilizando datos de prueba como entrada y comprobando que la salida es correcta.

  Estas pruebas se han ido realizado a medida que se implementaba cada clase del sistema.

- **Pruebas de integración**: Las pruebas de integración realizan comprobaciones de un conjunto de módulos que interaccionan entre ellos, para finalizar las pruebas del sistema en conjunto.

  Las pruebas de integración se han realizado a medida que se finalizaba el desarrollo de las diferentes funcionalidades correspondientes a los casos de uso. Estas pruebas han permitido comprobar que las diferentes clases se han integrado correctamente y se comportan como es esperado.

- **Pruebas de validación**: Se comprueba que el sistema cumple con la especificación y satisface los requerimientos funcionales. Estas pruebas se han realizado conjuntamente con las pruebas de integración, comprobando que el resultado de cada funcionalidad da el resultado esperado. Al finalizar la implementación de todo el sistema, se ha procedido a comprobar que el sistema ofrece todas las funcionalidades detalladas en la especificación.

Capítulo 7. Manual del usuario

El manual de usuario es un documento técnico de la aplicación, cuyo objetivo es dar soporte a todas aquellas personas que utilicen el producto que se plantea, tanto en la instalación como en el manejo del software. El producto actual ofrece dos manuales:

- Manual de instalación: Se describen los requisitos del sistema, el contenido del CD-ROM y el proceso detallado de instalación y desinstalación del software.

- Manual de uso: Se describen todas las funcionalidades del sistema y la interacción con el usuario mediante la interfaz gráfica del usuario.

Adicionalmente, en el CD-ROM se proporcionan estos manuales digitalizados en formato PDF.

7.1 Manual de instalación

7.1.1 Introducción

En este capítulo se encuentra la información referente a la instalación del software que permite la evaluación de los modelos navegacionales de aplicaciones web. El contenido de este manual contiene una descripción de todos los pasos que se deben seguir para instalar el sistema. También se detallan los requisitos del sistema.

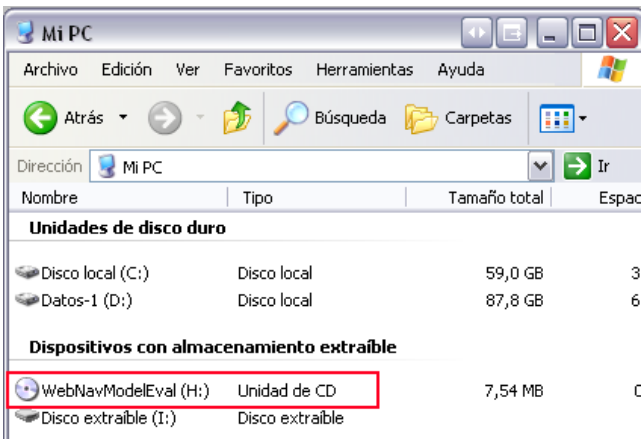
El software que se va a proceder a instalar se instalará en un único ordenador. En el caso de que se desee instalar sobre otros ordenadores, se deberá repetir el proceso de instalación para cada uno de éstos.

7.1.2 Requisitos del sistema

- Sistema operativo: Windows 9x, Windows NT/2000, Windows XP, Windows Vista, Windows 7 (32bits)
- Pantalla con un mínimo de resolución de 1024 x 768 pixeles y 256 colores.
- 3MB de espacio en disco requerido.
- 32MB de memoria.
- Java Runtime Environment 5 o superior.
- Adobe Reader, para poder abrir los manuales contenidos en el CDROM.

7.1.3 Contenido del CD-ROM

Para acceder al contenido del CD-ROM se puede usar el Explorador de Windows.



El CD-ROM contiene la siguiente estructura de ficheros y directorios:

.\Setup.exe
Fichero de instalación del software.

.\doc\Guide-Install-ES.pdf
Manual de instalación del software en formato PDF.

.\doc\Guide-Use-ES.pdf
Manual de uso de la aplicación en formato PDF.

.\doc\Memoria.pdf
Memoria del PFC, donde se detallan todas las fases del desarrollo del sistema.

.\src*
Directorio que contiene el código de la aplicación.

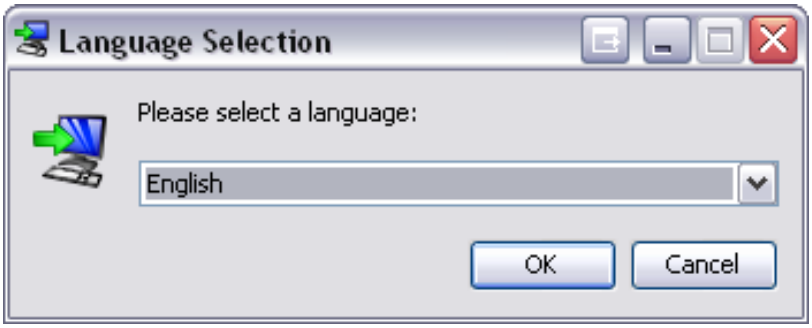
7.1.4 Instalación

7.1.4.1 Iniciar instalación

La instalación de la aplicación se inicia ejecutando el fichero .\Setup.exe contenido en el CD-ROM. En cualquier momento se puede cancelar la instalación, dejando el sistema operativo en el estado inicial.

7.1.4.2 Selección del idioma de la instalación

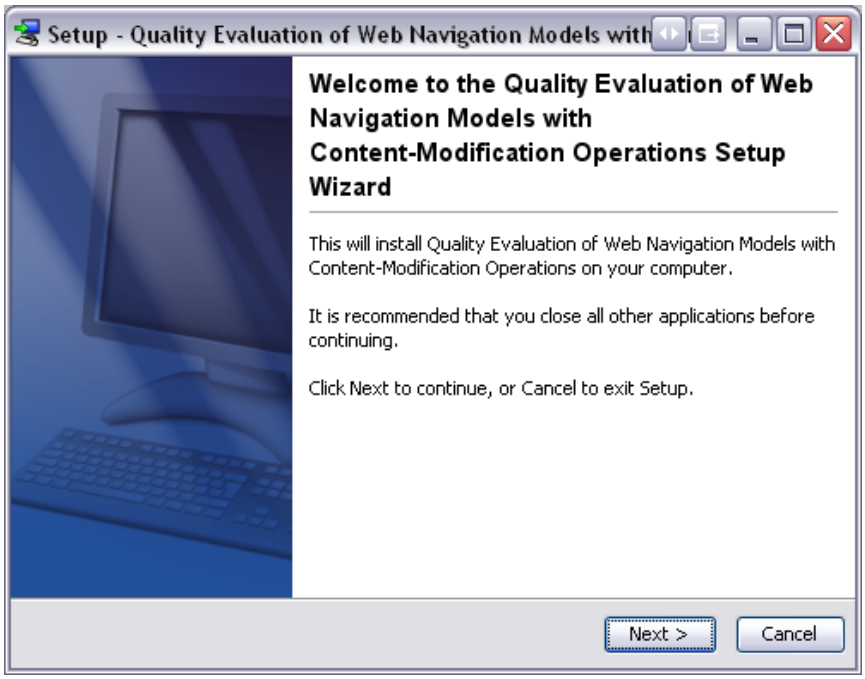
Al iniciar la instalación, el sistema permite seleccionar el idioma en el que se desea realizar la instalación, adaptando todo el proceso al idioma elegido.



Seleccionar el idioma deseado y presionar el botón **OK** para continuar el proceso.

7.1.4.3 Información general sobre la instalación

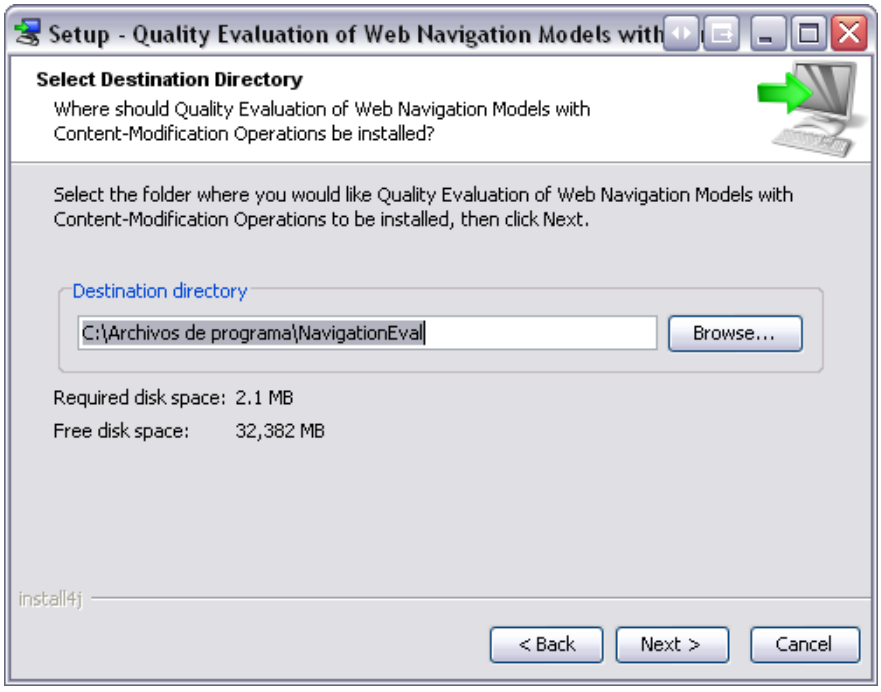
La venta que aparece presenta la información sobre el proceso de instalación.



Presionar el botón **Next** para continuar el proceso.

7.1.4.4 Directorio de instalación

El proceso de instalación permite seleccionar el directorio de destino de la aplicación.

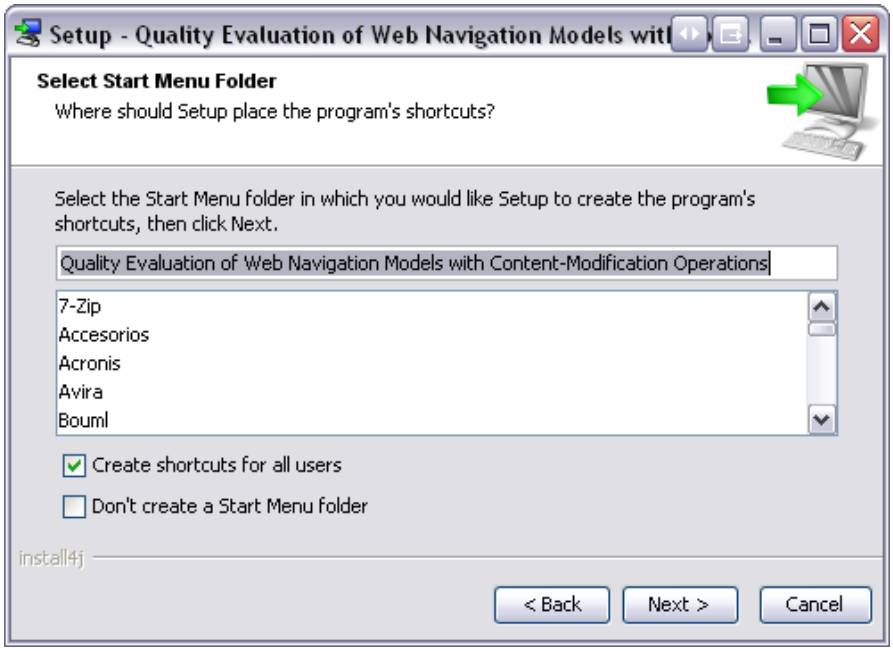


En la parte central se sitúa un cuadro de texto que contiene el directorio por defecto donde se instalará la aplicación. Se puede buscar otra ruta de instalación presionando el botón **Browse**. Igualmente se podrá cambiar la ruta directamente en el cuadro de texto.

Para continuar con la instalación en el directorio de destino indicado, presionar el botón **Next**.

7.1.4.5 Selección del grupo en el menú inicio.

A continuación se muestra el dialogo que permite al usuario asignar un grupo de accesos directos en el menú inicio de Windows.



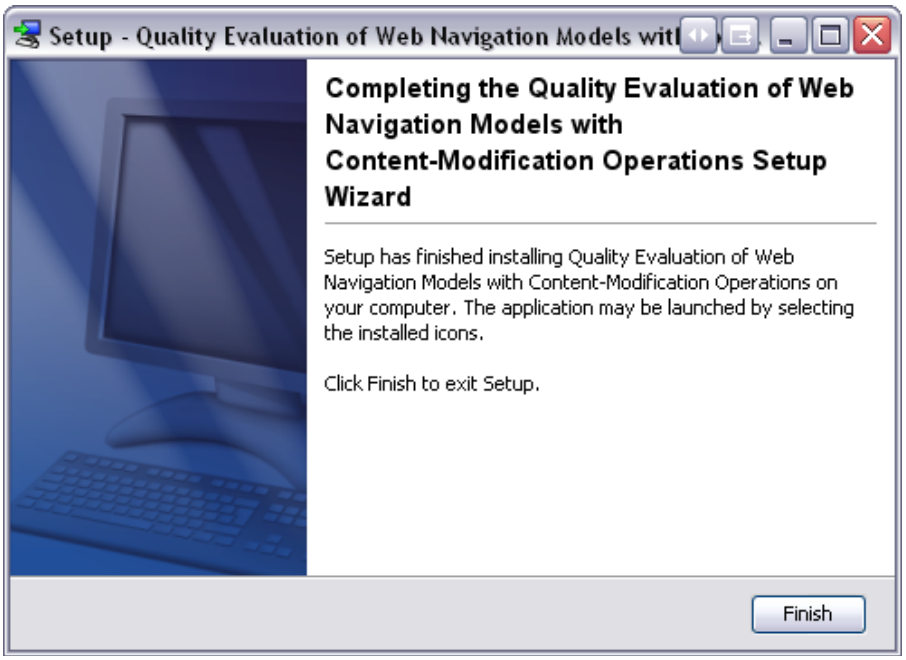
En el cuadro de texto de la parte superior se puede especificar el nombre del nuevo grupo que contendrá los accesos. Si el usuario desea incluirlos en un grupo existente, solamente tiene que seleccionar el grupo en la lista de grupos. El proceso permite crear los accesos directos visibles para todos los usuarios del sistema operativo, en el caso que la casilla de verificación (***Create shortcuts for all users***) esté marcada.

Por otro lado, si no se desean crear los accesos directos, se tiene que activar la casilla de verificación (***Don't create a start menu folder***)

Para proceder a la instalación presionar el botón **Next**.

7.1.4.6 Finalización de la instalación.

Al finalizar el proceso de instalación aparecerá el dialogo final.



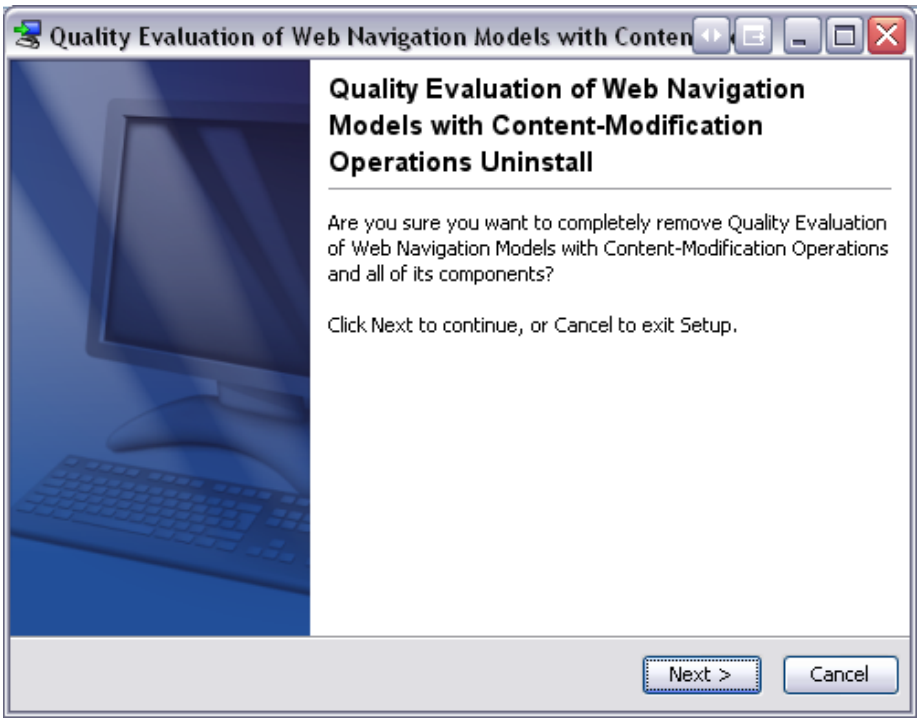
Para dar por finalizada la instalación se debe presionar el botón **Finish**.

7.1.5 Eliminar la instalación

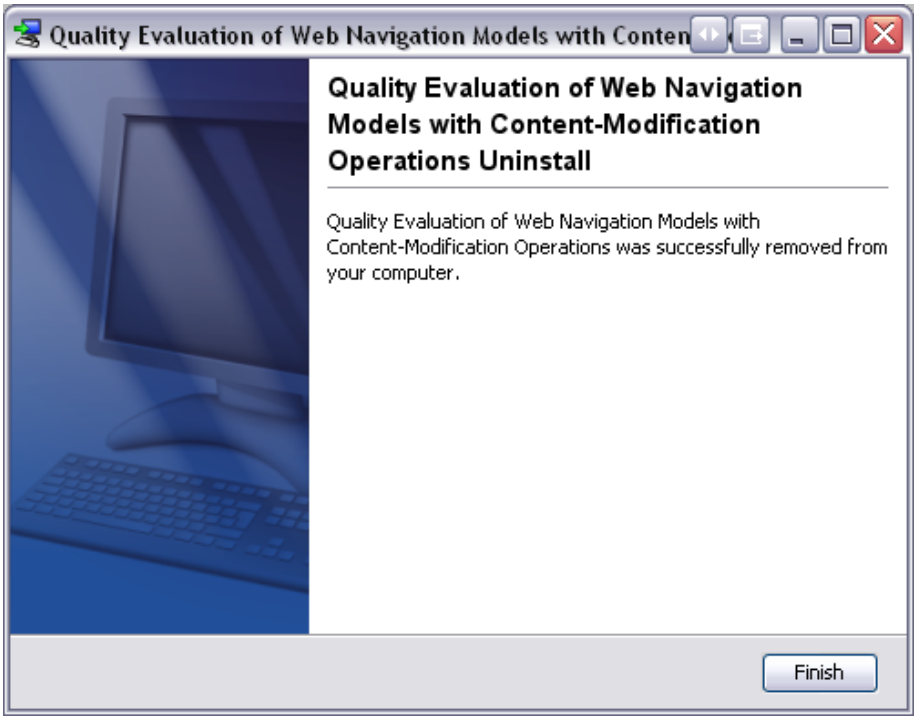
La desinstalación elimina por completo todos los ficheros y subdirectorios del directorio donde se encuentra instalado el software de evaluación de modelos navegacionales de aplicaciones web. En cambio, no elimina los ficheros correspondientes a proyectos creados con esta herramienta.

Para acceder a la desinstalación, solamente se tiene que ejecutar el fichero **uninstall.exe**, situado en el directorio de la aplicación. También se puede acceder mediante el menú de la aplicación situado en el menú de inicio de Windows, siempre y cuando se crease en el momento de la instalación.

Una vez se haya iniciado la instalación, aparecerá una ventana donde se informa sobre la eliminación del software.



Para proceder a la desinstalación del software, se tiene que presionar el botón **Next**. A continuación se eliminará el software y se mostrará la siguiente ventana, indicando que el proceso se ha realizado correctamente.



Se debe presionar el botón **Finish** para finalizar el proceso de desinstalación.

7.2 Manual de uso

7.2.1 Introducción

La aplicación de evaluación de modelos navegacionales es una herramienta que permite evaluar este tipo de modelos a partir de las propiedades de completitud y correctitud, que dependen del modelo de datos.

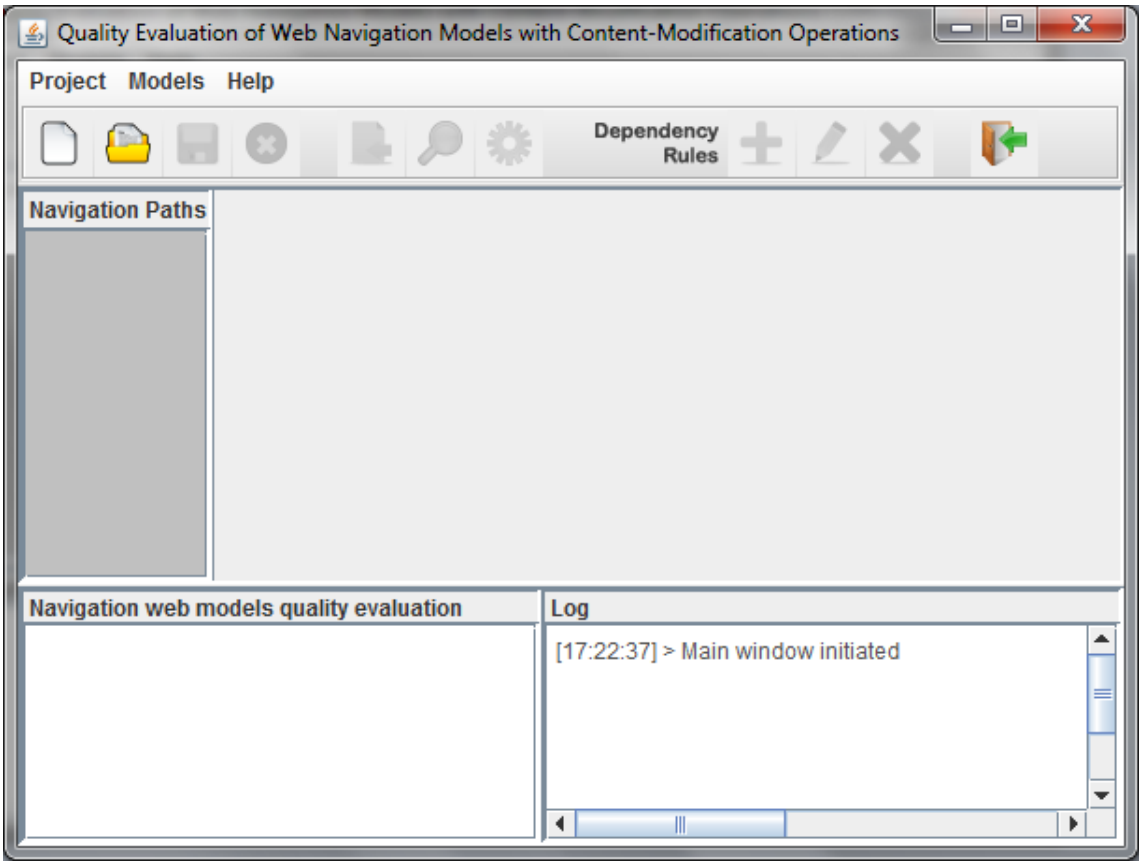
Estos modelos se deben definir con una herramienta de modelado de aplicaciones web que deja la información estructurada en ficheros XML. La aplicación está preparada para leer los datos definidos con la herramienta *WebRatio*.

El programa consta de dos bloques diferentes, cada uno con sus propias funcionalidades:

- Gestión de proyectos:
    - Crear un proyecto
    - Abrir un proyecto existente.
    - Guardar el proyecto actual
    - Cerrar el proyecto actual
- Gestión de los modelos:
    - Importación del modelo navegacional y de datos de WebRatio
    - Visualización de las rutas navegacionales
    - Evaluación cualitativa del modelo navegacional.
    - Gestión de las reglas de dependencia (Crear, Editar, Eliminar)

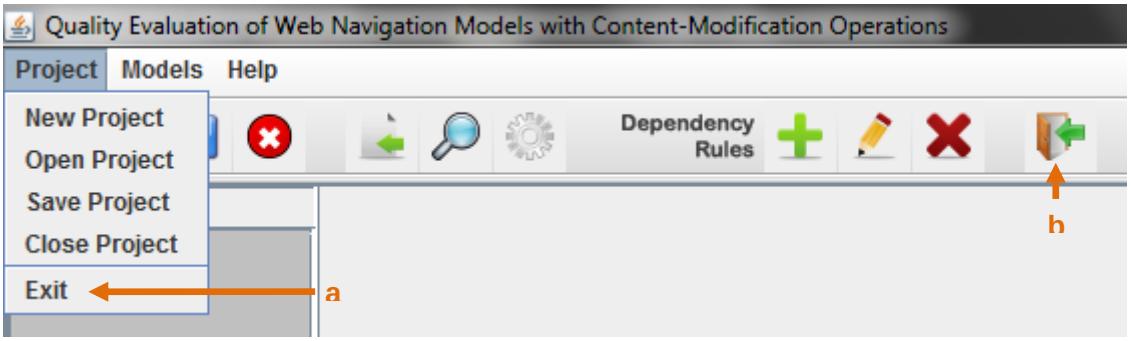
7.2.2 Acceso al sistema

Para acceder al sistema se tiene que ejecutar el fichero **NavigationEval.exe**, situado en la carpeta de la aplicación (por defecto *C:\Program Files\NavigationEval*). También se puede acceder a través del acceso directo situado en el menú de inicio de Windows, siempre y cuando este acceso se haya definido al instalar el programa. A continuación se abrirá el programa y mostrará la pantalla principal:



7.2.3 Cierre del sistema

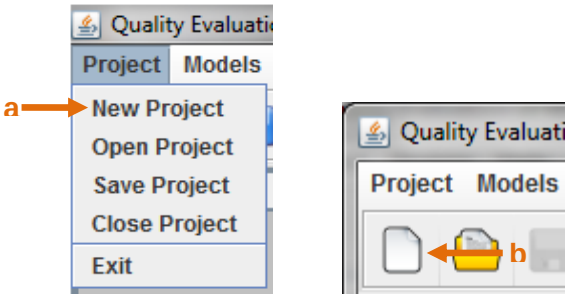
Para salir del sistema tan sólo es necesario acceder a la opción (a) *Exit* del menú *Project* o accediendo al icono correspondiente (b):



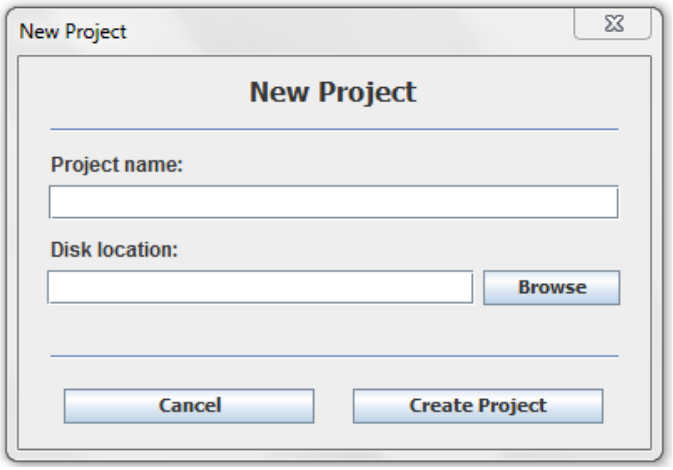
7.2.4 Gestión de proyectos

7.2.4.1 Crear un nuevo proyecto.

Para poder operar en la aplicación es necesario haber creado un proyecto previamente. Un proyecto se crea accediendo a la opción *New Project* (a) o accediendo al botón (b) correspondiente en la barra de iconos.



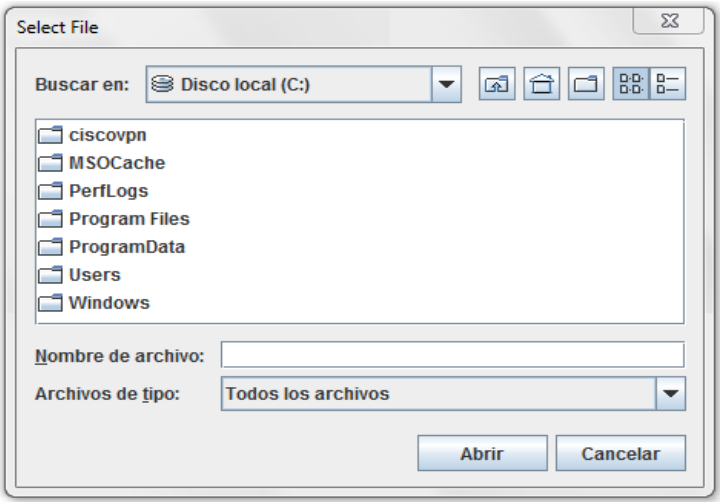
Después de solicitar el acceso a la creación del nuevo proyecto, el sistema mostrará una pantalla donde el usuario debe introducir el nombre del proyecto en el campo *Project Name* y la carpeta del proyecto en el campo *Disk Location*:



Una vez se han introducido los datos, se creará el proyecto en la carpeta indicada pulsando el botón *Create Project*, o podrá cancelar la creación del proyecto seleccionando el botón *Cancel.*

Adicionalmente, el sistema permite examinar la distribución de los ficheros del sistema operativo haciendo click en el botón *Browse*. Esto permite seleccionar una carpeta o fichero fácilmente sin necesidad de tener que introducir por teclado la ruta completa. En este caso, el sistema permitirá seleccionar un directorio, que es donde se creará el directorio.

La ventana de selección de ficheros contiene una zona de exploración de ficheros del sistema operativo con los controles típicos de navegación. En la parte inferior de la venta hay un campo de texto *Nombre de archivo* que contendrá el fichero o directorio seleccionado en el explorador.



La ventana de selección de ficheros permite seleccionar ficheros o una ubicación de carpeta. El comportamiento en ambos casos es prácticamente el mismo.

En el caso que se abra en modo *Selección de carpeta*, para seleccionar la carpeta deseada se tiene que acceder a la propia carpeta y hacer click en el botón Abrir para pasar la información al origen. El campo *Nombre de archivo* mantiene en todo momento la ruta de la carpeta explorada. Se transmite la ruta de la carpeta.

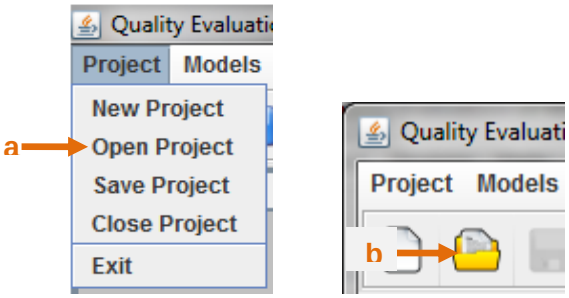
En el caso que se abra en modo *Selección de fichero,* se puede seleccionar un fichero haciendo doble click sobre el fichero situado en el explorador, o también se puede seleccionar un fichero y después hacer click en el botón *Abrir*. El campo *Nombre de archivo* se actualiza cuando se cambia el fichero seleccionado del explorador. La información que se transmite es la ruta completa del fichero.

En cualquier momento se puede cancelar la selección de un fichero o carpeta pulsando el botón *Cancelar*. En este caso no se transmite ninguna información.

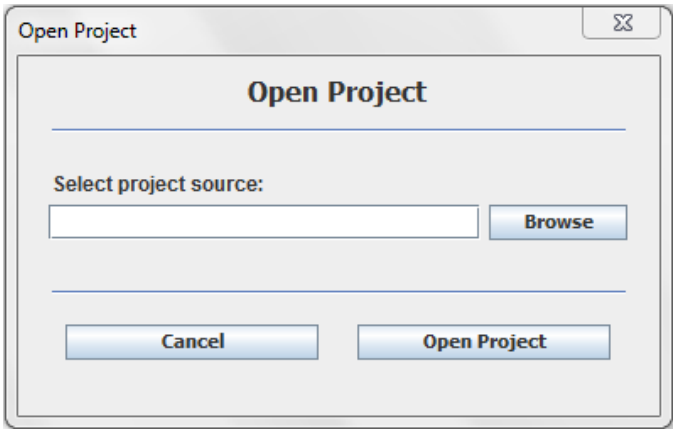
Este comportamiento se repite en todas las funciones que requieren interactuar con el disco, ya sea una acción de escritura como de lectura.

7.2.4.2 Abrir un proyecto existente

Un proyecto contiene los modelos importados y las reglas definidas en él. Para abrir un proyecto existente se tiene que acceder a la opción *Open Project (a)* del menú *Project* o a través del icono correspondiente (b).



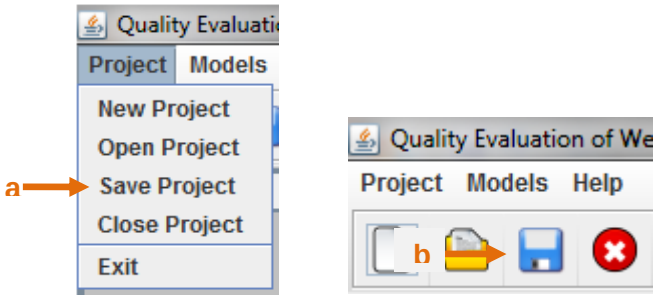
El sistema muestra la siguiente ventana, donde se tiene que introducir la ruta del proyecto:



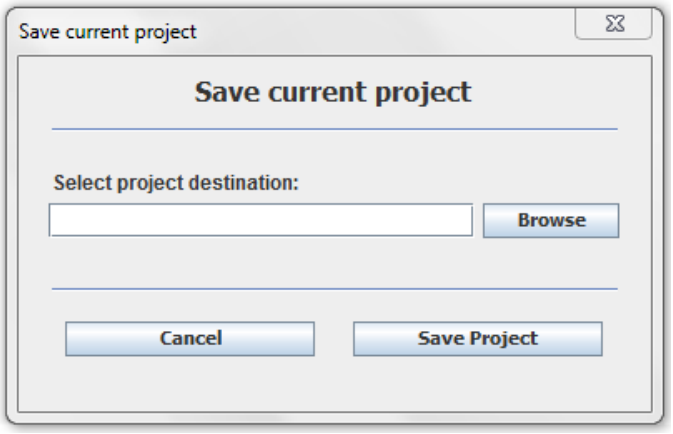
El sistema permite seleccionar el fichero a través de una ventana de selección, donde se tiene que seleccionar el fichero con la extensión .proj ubicado en la carpeta del proyecto. Una vez seleccionado el fichero, se abrirá haciendo click en el botón *Open Project*, o cancelar la acción (*Cancel*).

7.2.4.3 Guardar el proyecto actual

Un proyecto cargado contiene los modelos importados y las reglas definidas en él. Para guardar un proyecto cargado se tiene que acceder a la opción *Save Project (a)* del menú *Project* o a través del icono correspondiente (b).



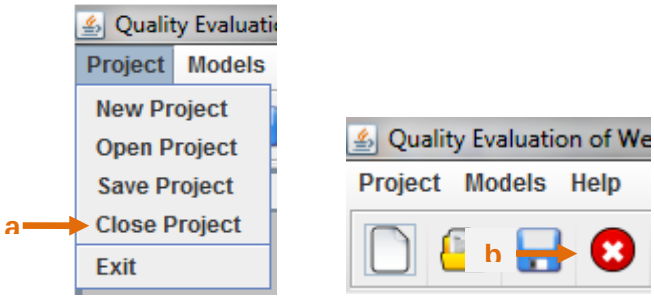
El sistema muestra la siguiente ventana, con el campo de texto rellenado con la ruta actual del proyecto cargado:



El sistema permite cambiar el directorio del proyecto través de una ventana de selección de directorio. Una vez seleccionado el directorio, el proyecto se guardará haciendo click en el botón *Save Project*, o se pueden cancelar la acción pulsando en el botón *Cancel*.

7.2.4.4 Cerrar el proyecto actual

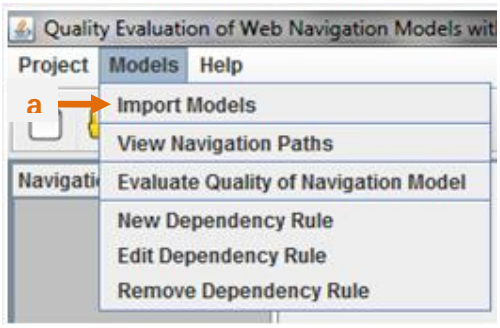
Para cerrar un proyecto cargado se tiene que acceder a la opción *Close Project (a)* del menú *Project* o a través del icono correspondiente (b). Cuando un proyecto se cierra, se perderán todos los cambios no guardados.

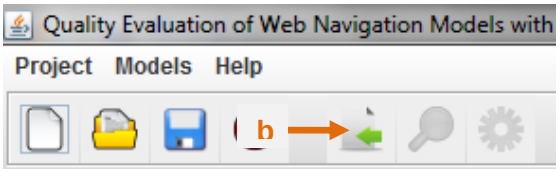


7.2.5 Gestión de los modelos

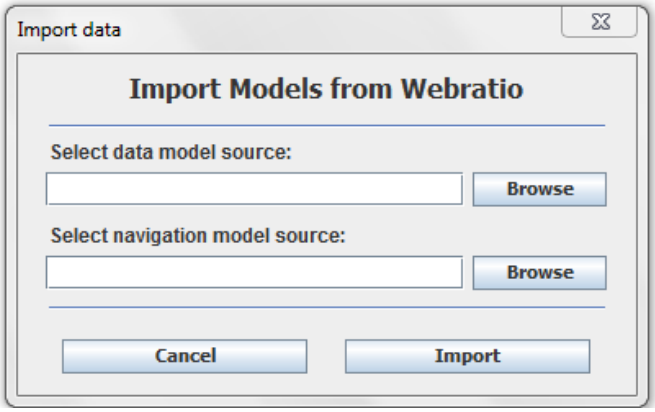
7.2.5.1 Importar Modelos

Para importar los modelos definidos en WebRatio a un proyecto existente se tiene que acceder a la opción *Import Models (a)* o al icono correspondiente (b) en la barra de iconos.





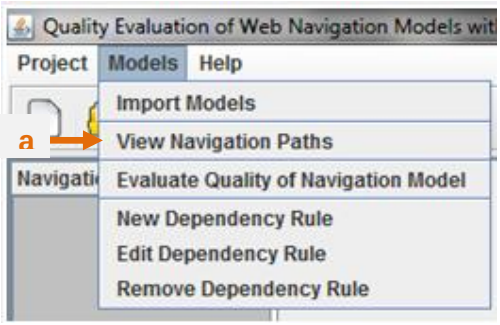
El sistema muestra una pantalla donde se debe indicar la ruta de los ficheros que contienen el modelo navegacional y de datos. El campo *Select data model source* contiene la ruta del modelo de datos, y el campo *Select navigation model source* la ruta el modelo navegacional. El sistema permite seleccionar mediante una ventana de selección de ficheros clicando los botones *Browse.*

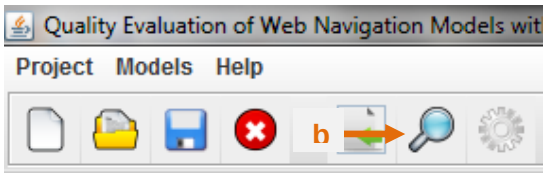


En el momento que los campos de las rutas de los modelos estén completos, se puede proceder a importar los datos al sistema clicando el botón *Import*. Se podrá cancelar la acción a través del botón *Cancel*.

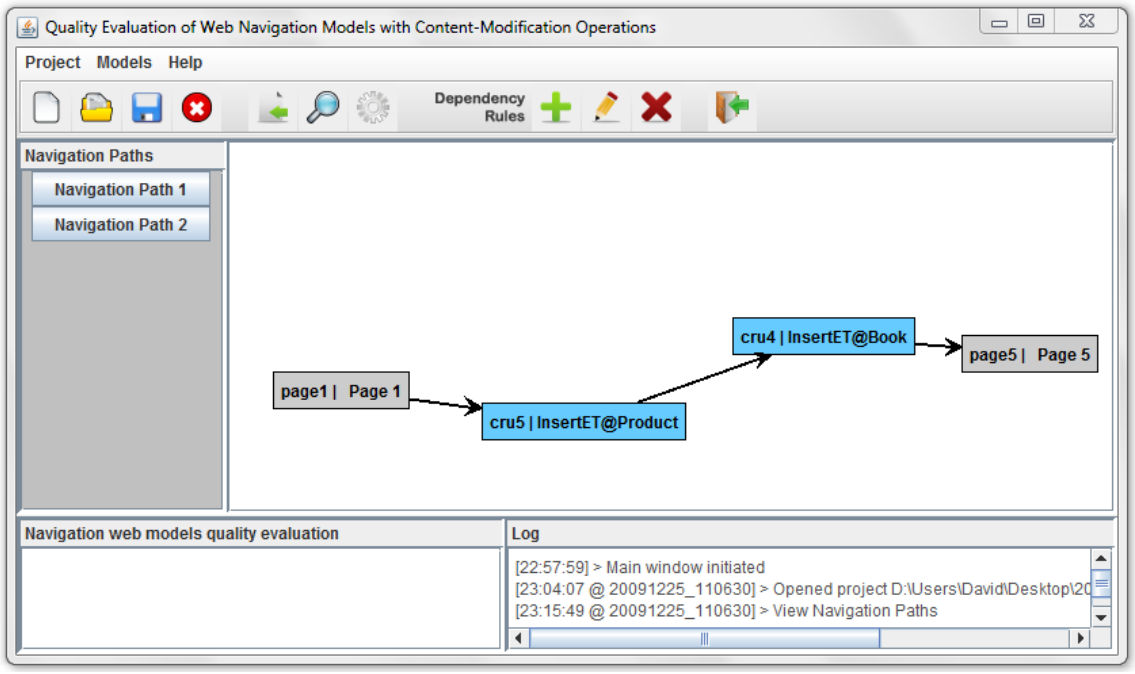
7.2.5.2 Visualizar las rutas del modelo navegacional cargado

Para visualizar las rutas navegacionales del modelo navegacional, se tiene que acceder a la opción *View Navigation Paths (a),* del menú *Models.* También se puede acceder clicando el icono correspondiente (b) de la barra de iconos.



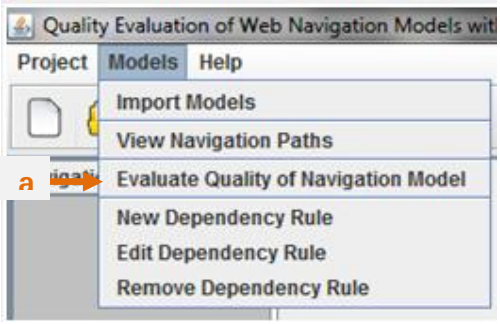


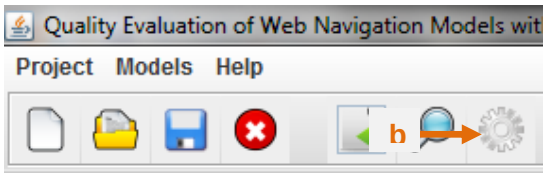
El sistema muestra en la izquierda de la ventana una lista de botones, a través de los cuales se podrá acceder a cada una de las rutas navegacionales del modelo navegacional. En parte central de la pantalla se muestra la ruta navegacional seleccionada.



7.2.5.3 Evaluar la calidad del modelo navegacional

La evaluación del modelo navegacional se inicia clicando la opción *Evaluate Quality of Navigation Model (a),* del menú *Models* o clicando el icono correspondiente (b) de la barra de iconos.

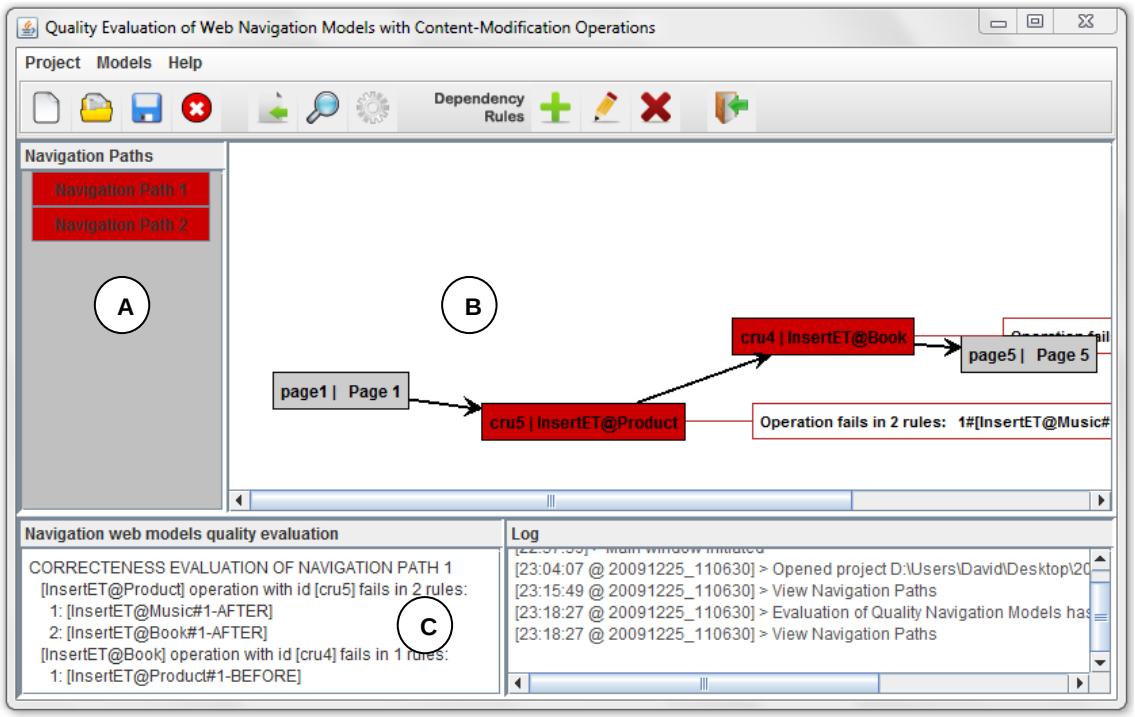




El sistema muestra el resultado de la evaluación en un área de texto (C) situado en la parte inferior izquierda de la ventana, determinando si el modelo es completo, mínimo y correcto. En el caso que el modelo no sea completo, el sistema informará de las operaciones que no se llegan a ejecutar.

Además del resultado textual, el sistema realiza una visualización de las rutas navegacionales, añadiendo anotaciones que hacen referencia a los errores de la

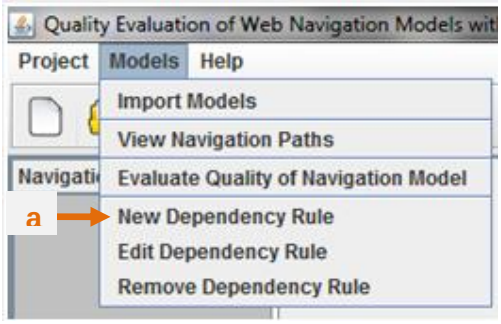
correctitud. A la izquierda de la ventana (A) aparece una lista de botones que dan acceso a cada una de las rutas del modelo navegacional. De este modo se puede ver qué operaciones incumplen las reglas de dependencia definidas por la propiedad de correctitud de una forma más gráfica. En parte central de la pantalla se muestra la representación gráfica de la ruta navegacional seleccionada (B).

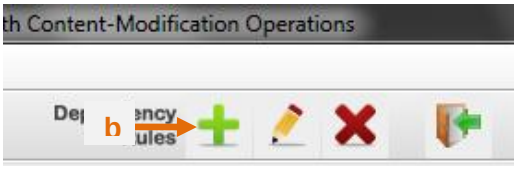


A medida que se va accediendo a las diferentes rutas, el texto de la evaluación (C) cambia para describir los errores detectados de la ruta seleccionada. En el caso que una ruta navegacional no sea correcta, el botón cambia el color a rojo. De igual modo, las operaciones que fallan de la ruta navegacional también aparecen marcadas en rojo y con un comentario indicando la dependencia que no cumplen.

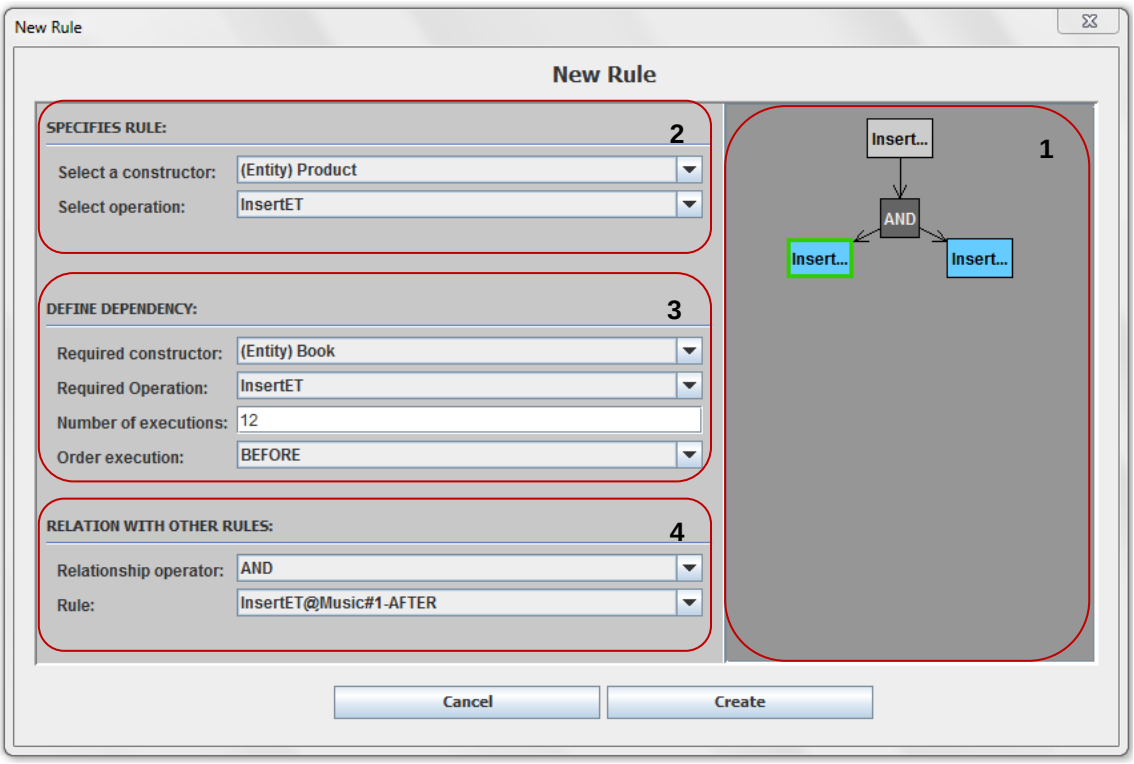
7.2.5.4 Nueva Regla de Dependencia

Para crear una nueva regla de dependencia se tiene que clicar la opción *New Dependency Rule (a)* del menú *Models* o clicando el icono correspondiente (b) de la barra de iconos.





El sistema abre la siguiente ventana:



La ventana se divide en 4 zonas:

1. Visor que permite ver las reglas de dependencia definidas en la operación seleccionada.

2. Selección de la operación para la que se define la regla. En esta zona aparecen los siguientes campos:

- *Select constructor*: Lista desplegable que contiene los constructores del modelo navegacional. Las entidades están etiquetadas con el prefijo *(Entity)* y las asociaciones entre entidades aparecen etiquetadas con el prefijo *(Relationship)*

- *Select operation*: Esta lista contiene las operaciones de modificación correspondientes al constructor seleccionado en el campo anterior. La nueva regla define una nueva dependencia de la operación seleccionada. En el momento que se cambie la operación, se actualizará el visor de reglas mostrando el árbol de dependencias correspondiente.

3. Definición de la regla:

- *Required constructor*: Igual que la primera, esta lista contiene los constructores del modelo de datos.

- *Required operation*: Se muestran las operaciones del constructor *Required constructor*. La operación seleccionada en esta lista es requerida en la dependencia que se está definiendo.

- *Number of executions*: Número de ejecuciones de la operación requerida para que la dependencia sea correcta.

- *Order execution*: En este campo se tiene que indicar si la operación se tiene que ejecutar antes (*BEFORE*) o después (*AFTER*) de la operación sobre la que se está estableciendo la dependencia. También puede ser que este criterio sea indiferente (*UNDEFINED*).

4. Relación con las demás reglas definidas para la operación indicada:

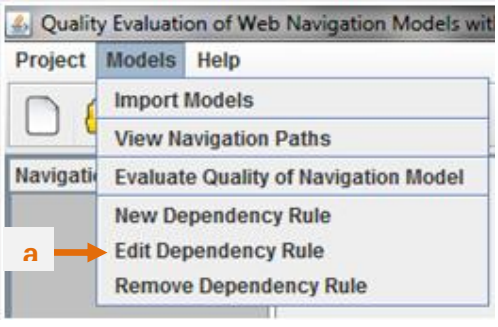
- *Rule*: Se muestran las reglas de la operación seleccionada y la regla seleccionada se relacionará con la nueva regla en la dependencia existente. En el momento que se seleccione una regla, el visor dibuja un borde más grueso sobre el elemento correspondiente a esta regla.

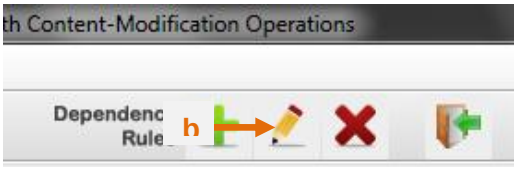
- *Relationship operator*: Se indica qué tipo de relación tendrá. Los operadores pueden ser AND u OR, donde AND indica que para que se cumpla la dependencia entre estas dos reglas, se deben cumplir las dos. En el caso que se seleccione OR, con el cumplimiento de una de las dos reglas ya se cumpliría la dependencia.

La dependencia se creará con los datos indicados clicando el botón *Create* situado en la parte inferior de la ventana. También se puede cancelar la operación clicando el botón *Cancel.*

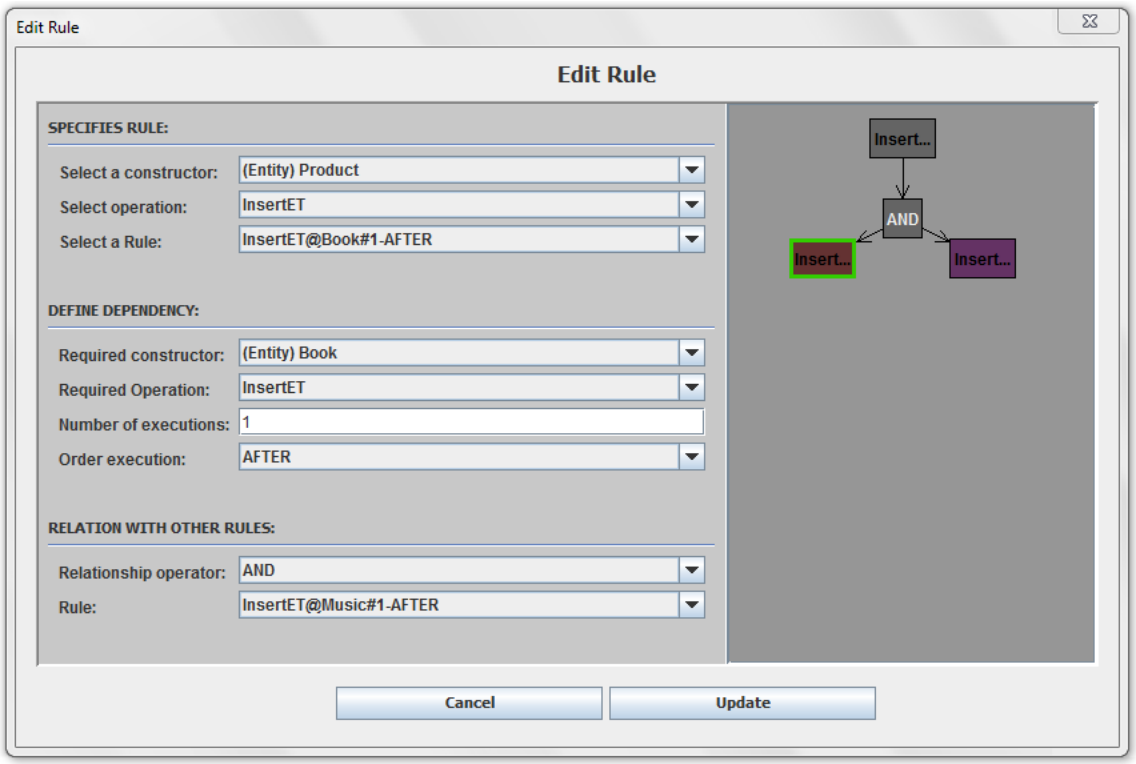
7.2.5.5 Editar Regla de Dependencia

De igual manera que se puede crear una regla, el sistema también permite la edición de las dependencias. Se puede acceder mediante la opción *Edit Dependency Rule (a)* del menú *Models* o clicando el icono correspondiente (b) de la barra de iconos.





El sistema muestra la siguiente pantalla:



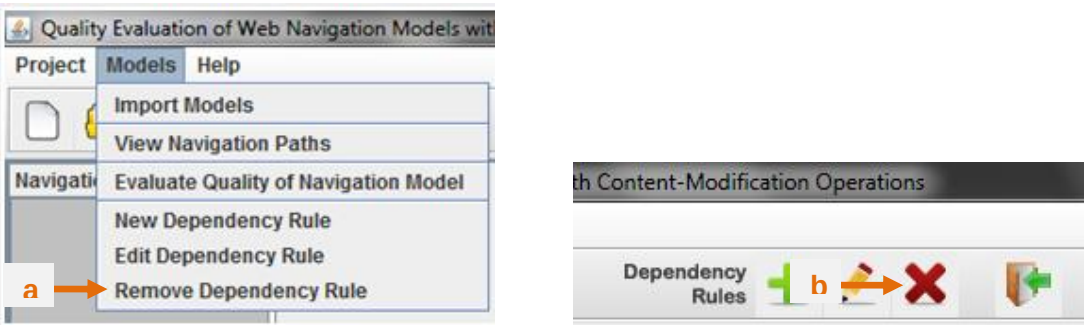
Es una ventana prácticamente igual que la ventana de creación de reglas. En este caso aparece un nuevo campo en la zona de la selección de la operación dependiente:

- *Select a Rule*: Se indica la dependencia que se quiere editar. La lista muestra las dependencias de la operación seleccionada. Los demás campos dejan por defecto los valores de la regla seleccionada. El usuario también puede seleccionar la regla clicando sobre uno de los nodos del árbol situado en el visor de dependencias de la operación seleccionada.

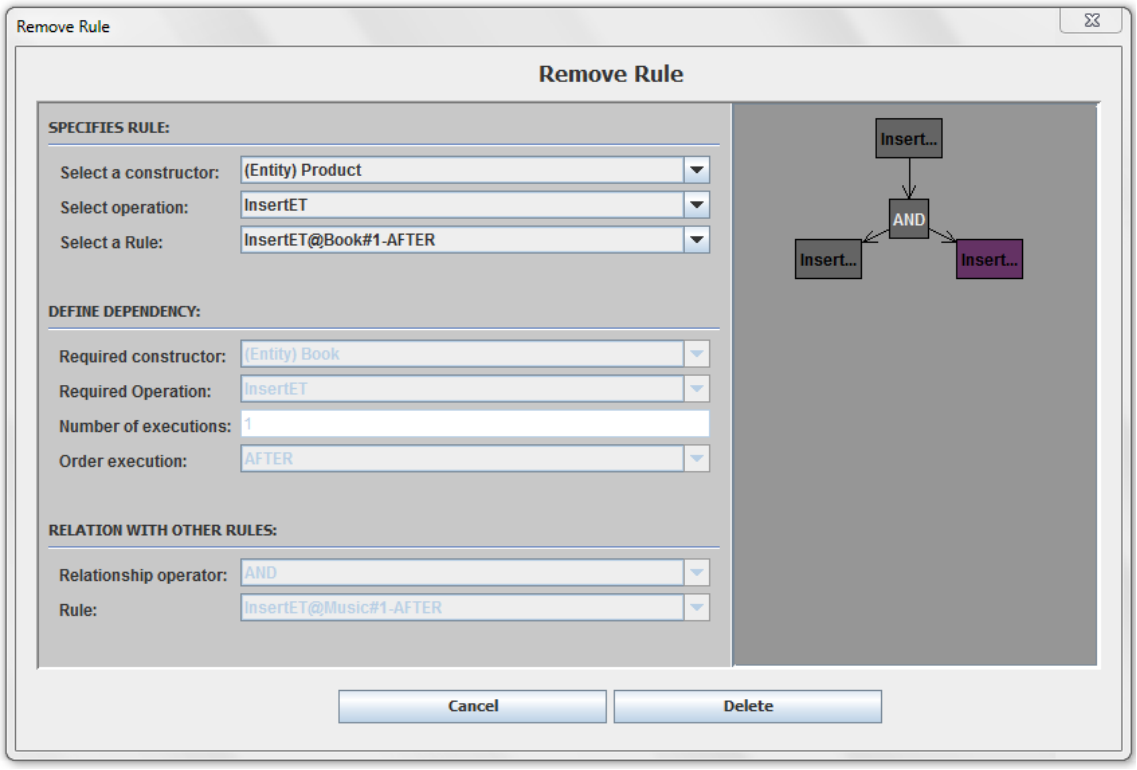
La dependencia se actualizará con los datos indicados clicando el botón *Update* situado en la parte inferior de la ventana. También se puede cancelar la operación clicando el botón *Cancel.*

7.2.5.6 Eliminar Regla Dependencia

El sistema permite eliminar dependencias de operaciones. El acceso a esta opción se realiza mediante la opción *Remove Dependency Rule (a)* del menú *Models* o clicando el icono correspondiente (b) de la barra de iconos.



El sistema muestra en una nueva ventana la siguiente pantalla para definir una nueva regla de dependencia:



La ventana tiene las mismas características que la edición, aunque en este caso los campos de definición de la regla aparecen deshabilitados.

La dependencia seleccionada se eliminará clicando el botón *Delete*. También se puede cancelar la operación clicando el botón *Cancel.*

Capítulo 8. Conclusiones

El objetivo principal de este proyecto es el desarrollo de una aplicación que permita la evaluación cualitativa de los modelos navegacionales de aplicaciones web, especificados con la herramienta WebRatio, a partir de las propiedades de correctitud y completitud.

Este objetivo se ha cumplido, desarrollando un sistema que integra los modelos especificados en WebRatio y que permite realizar una evaluación cualitativa de éstos. Además, el sistema permite una gestión de las dependencias del modelo de datos, asignando por defecto las dependencias descritas por la definición de correctitud.

El sistema permite visualizar, mediante gráficos generados con la librería JGraph, las diferentes rutas navegacionales del modelo navegacional.

Asimismo, el usuario podrá almacenar y recuperar la información en un soporte de almacenamiento, mediante la gestión de proyectos.

En el ámbito personal, he extraído las siguientes conclusiones:

He conocido el lenguaje de modelado de aplicaciones web (WebML) y los elementos que lo componen. También he aprendido a usar XML, conociendo los diferentes ámbitos de uso. Esto me ha servido de gran ayuda para otros proyectos debido a su gran difusión e integración en diferentes arquitecturas.

Durante el desarrollo del proyecto se han puesto en práctica los conocimientos de ingeniería del software, finalizando con una implementación en Java, lenguaje que ya conocía.

Como conclusión final, he de destacar la puesta en práctica de los conocimientos adquiridos durante la carrera, además del conocimiento propio en la elaboración de proyectos de desarrollo de software.

Capítulo 9. Trabajo futuro

El desarrollo del proyecto se ha basado en criterios de portabilidad, extensibilidad y reusabilidad, para poder ampliar el sistema o aprovechar algún modulo en otros sistemas.

Una de las características del sistema es la importación de los modelos generados en WebRatio. Una posible ampliación del sistema, y que ya se valoró al iniciar el proyecto, es una gestión de plantillas de importaciones para poder dar soporte a más sistemas de modelado de aplicaciones web que utilicen XML y que sea el propio diseñador el que gestione estas plantillas. A continuación se muestra una ilustración con esta idea:

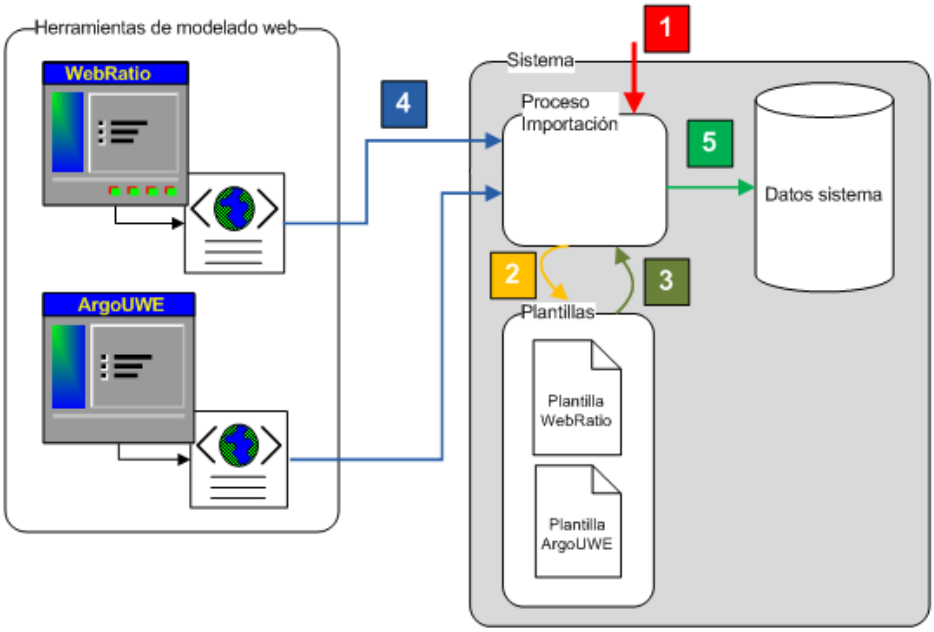


Fig.1. Esquema funcional de las plantillas en el proceso de importación: El usuario solicita la importación de datos (1), el sistema consulta las plantillas definidas en el sistema (2). A partir de la plantilla (3) y los datos (4) (modelos definidos en herramientas de modelado externas), el sistema, siguiendo el proceso de importación, adapta los datos (5) para el posterior uso en la aplicación.

Actualmente la evaluación de la calidad de los modelos navegacionales se limita a informar de los fallos, indicando las operaciones que faltan para ser completo y

las que vulneran las reglas de dependencia de la correctitud. La evaluación debería proponer una solución para la correctitud y completitud.

Para ayudar al diseñador, además de la propuesta de un modelo navegacional completo y correcto, es realmente interesante que el sistema ofrezca un modelo navegacional compatible con WebRatio. Por este motivo se plantea integrar este software con el ya desarrollado en otro PFC (Gómez S., 2008) que se basa en el mismo estudio (Cabot, Ceballos, Gómez, 2007) que el presente, y que se encarga de la generación modelos navegacionales correctos y completos a partir del modelo de datos.

Por otro lado, el sistema permite visualizar con la librería JGraph las rutas navegacionales, conservando los objetos definidos en WebRatio en las mismas coordenadas, sin distribuirlos óptimamente. En cambio, JGraph permite una distribución automática de los vértices que componen el grafo. Este aspecto se puede tener en cuenta para futuras modificaciones, ya que facilitaría la visión del gráfico, principalmente en los grafos más complejos o con un número de vértices muy alto.

Capítulo 10. Bibliografía

- Cabot, Jordi; Ceballos, Jordi y Gómez Cristina (2007). *On the Quality of Navigation Models with Content-Modification Operations*, LNCS 4607*,* Springer Verlag, pp. 59-73.

- Ceri, Stefano; Fraternali, Piero; Bongio, Aldo y Maurino Andrea (2003). *Modeling data entry and operations in WebML,* LNCS 1997*,* Springer Verlag, pp. 201-214.

- Ceri, Stefano; Fraternali, Piero; Bongio, Aldo; Brambilla, Marco; Comai, Sara y Matera, Maristella (2003). *Designing Data-Intensive Web Applications*

- Shaffer, Clifford A. (1998). *A Practical Introduction to Data Structures and Algorithm Analysis, Java Edition*

- Gómez Rigol, Sergio (2008). *Generador de models navegacionals per a sistemes web*. UPC.

- Manual JGraph.
  http://www.jgraph.com/jgraph5.html

- The Web Modeling Language
  http://www.webml.org/

- WebRatio
  http://www.webratio.com/

- Extensible Markup Language (XML)
  http://www.w3.org/XML/

Agradecimientos

En primer lugar quiero agradecer a Cristina Gómez Seoane la oportunidad de haber realizado este proyecto, su colaboración y paciencia en todo momento.

También quiero expresar mi agradecimiento a los miembros del tribunal Enrique Mayol Sarroca y Lluis Ametller Congost, por la atención prestada y por el tiempo dedicado a la evaluación de este trabajo.

Por último, agradezco a mis padres, amigos y todas las personas que han estado cerca durante todo este tiempo, por el apoyo y la confianza que me han transmitido.