



FACULTAD DE INFORMÁTICA
UNIVERSIDAD NACIONAL DE LA PLATA

Desarrollo Dirigido por Modelos de Aplicaciones de Hipermedia Móvil

TESIS PRESENTADA PARA OBTENER EL GRADO DE
DOCTOR EN CIENCIAS INFORMÁTICAS

Autor: **Cecilia Challiol**

Directora: **Dra. Silvia E. Gordillo**

Codirector: **Dr. Gustavo H. Rossi**

- Septiembre 2011 -

Agradecimientos

A mis seres queridos, gracias por haberme acompañado en esta etapa con el apoyo incondicional de siempre.

A todas aquellas personas que en forma directa o indirecta han colaborado en la realización de esta tesis. En particular, quiero agradecerle a Silvia Gordillo por su guía continua, no solamente en el desarrollo de esta tesis sino también en mi formación.

ÍNDICE

1	INTRODUCCIÓN	7
1.1	Motivación	7
1.2	Objetivo de la tesis	11
1.3	Estructura de la tesis	11
2	BACKGROUND	13
2.1	Desarrollo dirigido por modelos.....	13
2.1.1	Una implementación particular de desarrollo dirigido por modelos (<i>Model-Driven Architecture - MDA</i>)	16
2.1.2	Desarrollo dirigido por modelos en aplicaciones Web	18
2.2	Separación de Concerns	20
2.2.1	Separación de Concerns en aplicaciones Web	22
2.3	Realidad Aumentada	23
2.3.1	Aplicaciones de Realidad Aumentada para dispositivos móviles	25
2.3.2	Aplicaciones de Hipermedia Física	28
2.4	Resumen	32
3	MODELADO DE APLICACIONES DE HIPERMEDIA MÓVIL	35
3.1	Aplicaciones de Hipermedia Móvil: el mundo digital y el real	35
3.2	Utilización de Separación de Concern en Hipermedia Móvil	38
3.2.1	Generación de links caminables derivados	40
3.3	Enfoque para modelar aplicaciones de Hipermedia Móvil	42
3.3.1	Modelo de Contenido	44
3.3.2	Modelo Unificado	46
3.3.3	Modelo Navegacional	50
3.3.4	Modelo de Presentación y Modelo del Usuario	53
3.4	Resumen	54
4	EXTENSIÓN DE UWE PARA REPRESENTAR APLICACIONES DE HIPERMEDIA MÓVIL	55
4.1	Metamodelo para aplicaciones de Hipermedia Móvil	55
4.2	Perfil de <i>UML</i> para aplicaciones de Hipermedia Móvil	60
4.3	Resumen	65
5	ESPECIFICACIÓN DE LOS MODELOS PROPUESTOS	67
5.1	Modelo de Contenido	67
5.2	Modelo Unificado	68
5.3	Transformación del Modelo de Contenido al Modelo Unificado	70
5.4	Modelo Navegacional	71

5.5 Transformación del Modelo de Unificado al Modelo Navegacional	73
5.6 Instanciación del Modelo Navegacional	76
5.7 Modelo de Presentación	79
5.8 Modelo del Usuario	83
5.9 Resumen	85
6 MAGICMOBILEUWE: HERRAMIENTA PARA HIPERMEDIA MÓVIL	89
6.1 Características básicas del plugin <i>MagicMobileUWE</i>	89
6.2 <i>MagicMobileUWE</i> : Modelo de Contenido	91
6.3 <i>MagicMobileUWE</i> : Modelo Unificado	94
6.4 <i>MagicMobileUWE</i> : Modelo Navegacional	98
6.5 <i>MagicMobileUWE</i> : Modelo de Instancias Navegacional	101
6.6 <i>MagicMobileUWE</i> : Modelo de Presentación	106
6.7 <i>MagicMobileUWE</i> : Modelo del Usuario	109
6.8 Detalles adicionales de <i>MagicMobileUWE</i>	113
6.9 Estereotipos creados para el funcionamiento de <i>MagicMobileUWE</i>	116
6.10 Resumen	120
7 CASOS DE ESTUDIO	123
7.1 Asistencia Turística	123
7.2 Ejemplos de <i>Mobile Urban Drama</i>	130
7.3 Características particulares de los ejemplos presentados	142
7.4 Resumen	145
8 DERIVACIÓN DE APLICACIONES MÓVILES	149
8.1 Modelo básico de la Aplicación de Hipermedia Móvil	149
8.2 Aplicación de Hipermedia Móvil - J2ME	152
8.3 Aplicación de Hipermedia Móvil – Web Móvil	155
8.4 Derivación de aplicaciones	157
8.4.1 Derivación de aplicaciones J2ME	158
8.4.2 Derivación de aplicaciones Web Móvil	159
8.4.3 Consideraciones en ambas derivaciones	161
8.4.4 Ejemplos de aplicaciones derivadas	162
8.4.4.1 Aplicaciones <i>J2ME</i>	162
8.4.4.2 Aplicaciones Web Móvil	169
8.5 Resumen	175
9 TRABAJOS RELACIONADOS	177
9.1 Trabajos relacionados con las aplicaciones de Hipermedia Móvil	177
9.2 Trabajos relacionados con el enfoque propuesto	183

9.3 Resumen	190
10 CONCLUSIONES Y TRABAJOS FUTUROS	193
ACRÓNIMOS	205
GLOSARIO DE TÉRMINOS Y DEFINICIONES	207
BIBLIOGRAFÍA	211
ANEXO A: CARACTERÍSTICAS BÁSICAS DE MAGICMOBILEUWE	217
ANEXO B: TRANSFORMACIONES DEFINIDAS EN MAGICMOBILEUWE ...	225
B.1. Transformación del Modelo de Contenido al Modelo Unificado	226
B.2. Transformación del Modelo de Contenido al Modelo Unificado seleccionando algunos concerns	235
B.3. Transformación del Modelo Unificado al Modelo Navegacional	237
B.4. Métodos invocados por las transformaciones	244
ANEXO C: CREACION DE MODELOS EN MAGICMOBILEUWE	247
C.1. Modelos de Contenido, Unificado y Navegacional	247
C.2. Modelo de Instancias Navegacional	248
C.3. Modelo de Presentación	249
C.4. Modelo del Usuario	251

1. INTRODUCCIÓN

1.1. Motivación

En los últimos años los dispositivos móviles se han incorporado como un objeto de uso común dentro de la vida cotidiana. Esto, sumado a la disponibilidad cada vez más alta de redes inalámbricas, permite que los usuarios puedan estar conectados a la Web en cualquier momento desde sus dispositivos móviles (por ejemplo desde su celular), accediendo a información y a servicios.

En la actualidad, no solamente se cuenta con acceso a Internet inalámbrico como un servicio habitual sino también con una variada disponibilidad de tecnologías de sensado (por ejemplo: *GPS [Global Positioning System]*, *Bluetooth*, *RFID [Radio Frequency Identification]*, Lectores de código de barras, etc.). Estas tecnologías permiten que el usuario además tenga información del lugar donde se encuentra. De esta manera, puede contar con información relevante del mundo real que lo rodea además de navegar por información digital. Usando estas tecnologías el usuario puede recibir entre otros, servicios sensibles al contexto; el ejemplo más común es brindar servicios acordes a la posición actual del usuario. Cuando un usuario cambia de posición estos servicios se actualizan.

Como un ejemplo de aplicación de este tipo, supongamos que una persona que cuenta con un dispositivo móvil (de ahora en adelante lo llamaremos usuario) realiza un viaje a *Roma*. La persona cuenta solamente con dos días para visitar esa hermosa ciudad. Esta información está registrada en la agenda del usuario, también tiene registrada la dirección del hotel en el cual se hospeda durante su estadía. Imaginemos que el usuario arriba a *Roma* en tren. Cuando baja del tren corre una aplicación que tiene información disponible de la ciudad. De esta forma, el usuario puede solicitar información a cerca de dónde está el hotel en el que se va a alojar y cómo llegar hasta él. El sistema muestra un mapa indicando tanto, dónde se encuentra el usuario, como la ubicación del hotel. A medida que va caminando, el sistema va actualizando su posición en el mapa para que el usuario visualice dónde se encuentra en cada momento y le indica cuánto avanzó en el recorrido.

Una vez instalado en el hotel, el usuario decide salir a realizar un recorrido por la ciudad, el sistema detecta (mediante el *GPS*) que se encuentra en la puerta del hotel y le ofrece una serie de lugares a visitar; el usuario elige dirigirse a visitar los *Museos Vaticanos*. El hecho de elegir la opción que se corresponde con los *Museos Vaticanos*, es interpretado por el sistema como que el usuario tiene intención de caminar a este lugar, como resultado, recibe un mapa indicando el camino que debe realizar para llegar al destino. Una cuadra antes de llegar a los museos, ve que hay una cola de casi dos cuadras para entrar, en ese momento evalúa que va a demorar mucho tiempo y como sólo se queda por dos días en dicha ciudad decide cancelar el recorrido porque quiere conocer la mayor cantidad de lugares posibles. El sistema detecta que el usuario canceló el camino que estaba realizando y en ese momento le ofrece una lista de posibles lugares que puede visitar desde donde se encuentra ubicado. El usuario elige de esta lista, visitar la *Basílica de San Pedro* y así, el sistema le brinda un mapa con el camino hasta la *Basílica*. Al llegar a la misma, recibe una página Web con información sobre la *Basílica* y con varios links relacionados. Entra al lugar y mientras va visitando el interior de la basílica, el dispositivo lo provee con información detallada del lugar donde se encuentra y la posibilidad de navegar entre diferentes links que le permiten acceder a información relacionada.

Al salir de la *Basílica* recibe una lista de posibles lugares a visitar y decide que quiere ir a conocer el *Coliseo*, otro símbolo emblemático de *Roma*, al elegir este lugar, como

resultado, el sistema le brinda un camino al *Coliseo*. El usuario comienza a realizar el recorrido y al pasar por el *Castillo San Ángel* ve que hay un código de barra, lo lee con la cámara de su dispositivo móvil y recibe una página Web con información del lugar. A su vez recibe un audio explicativo de la historia del lugar que va escuchando mientras observa el lugar y navega por los links de la página Web. Aún cuando el usuario recibe esta información, el sistema no pierde de vista que tiene un camino pautado al *Coliseo*. Puede navegar por los links relacionados al *Castillo* pero siempre se mantiene vigente el camino que quiere realizar.

Unos minutos más tarde el usuario sale del *Castillo* y continúa su camino. Mientras va caminando ve una tienda de regalos que está a una cuadra del camino que tiene pautado y decide dirigirse allí, el sistema detecta que se salió del camino establecido y le consulta para saber si desea cancelar el camino pautado o si quiere retomar el camino pautado. El usuario no le presta atención a la pregunta que le hizo sistema y se dirige a mirar la vidriera de la tienda, al ver que nada es de su interés, ve que el sistema le consulta si desea cancelar o retomar el camino pautado, el usuario selecciona la segunda opción (retomar el camino pautado), como resultado recibe un nuevo camino que considera como origen la posición actual donde se encuentra el usuario.

Siguiendo el mapa sugerido por el sistema, el usuario llega al *Coliseo* en ese momento el sistema le indica que a llegado (información controlada mediante el uso del *GPS*) a su destino y recibe una página Web del lugar.

Como puede observarse en el ejemplo presentado, este tipo de aplicaciones son muy complejas y dependen de las necesidades de cada usuario. Como el usuario además de navegar información también se encuentra ubicado en un lugar específico, puede ser que le surjan inquietudes relacionadas al espacio que lo rodea, por ejemplo, desviarse del recorrido porque hay algo que le llamó la atención. Además, el usuario puede contar con información relevante de los lugares como horarios, costo de la entrada o información histórica o arquitectónica. Es decir, hay información de distinta naturaleza (información de contexto) que va a ir recibiendo, tanto mientras está ubicado en un lugar, como cuando va realizando un recorrido y pasa por un lugar de interés en el transcurso del mismo.

Como hemos visto a través del ejemplo, algunas de las funcionalidades que debe brindar este tipo de aplicaciones móviles son:

- Desplegar información del lugar donde se encuentra un usuario.
- Calcular caminos cuando el usuario elige un lugar para visitar, (es decir, calcular el camino desde donde se encuentra ubicado hasta el destino elegido), posiblemente utilizando diferentes criterios de búsqueda (camino más corto, más transitado, etc.).
- Sugerir lugares a recorrer desde donde se encuentra ubicado el usuario.
- Permitir que un usuario cancele el recorrido que está realizando, en cualquier momento del mismo.
- Considerar que el usuario puede desviarse del camino pautado. En este caso además, el sistema debe poder asistirlo para retomar su camino.
- Considerar información adicional de los sitios de interés (por ejemplo, los horarios de apertura y cierre del destino elegido).
- Mostrar información de los lugares por los que va pasando un usuario mientras realiza un recorrido.
- Permitir al usuario volver a ver una página ya visitada como así también volver físicamente a un lugar ya visitado. En el caso de volver físicamente el sistema debe calcular el camino al lugar anteriormente visitado.

Las aplicaciones como la descrita, en donde el usuario navega tanto digitalmente

(como en las aplicaciones Web tradicionales o también conocidas como aplicaciones de Hipermedia), como físicamente (es decir, se mueve por el mundo real) se denominan aplicaciones de Hipermedia Móvil [Gronbaek et al., 2003]. En estas aplicaciones el usuario tiene la posibilidad de obtener información mediante algún dispositivo móvil de dos maneras distintas: una digitalmente es decir a través de la navegación tradicional de las aplicaciones de Hipermedia y otra de manera presencial acorde a su posición cuando un objeto del mundo real lo sensa y la aplicación automáticamente le provee información del lugar. La posición del usuario es fundamental para determinar qué información debe recibir en cada lugar. Estas aplicaciones deben considerar la movilidad del usuario como parte esencial de su funcionamiento sumando complejidad a la hora de la creación de las mismas. Por lo tanto, las aplicaciones de Hipermedia Móvil son un caso particular de las llamadas aplicaciones móviles, ya que tienen la particularidad de contar con el concepto de link y navegación.

Se puede decir que estas aplicaciones tienen además algunas de las características de las aplicaciones de Realidad Aumentada [Azuma et al., 2001], ya que el usuario recibe en su dispositivo información acorde a su posición. Es decir, se aumenta los lugares que el usuario está visualizado en el mundo real con información referente a los mismos.

En los últimos años, las expectativas de los usuarios han aumentado paralelamente al crecimiento de las funcionalidades ofrecidas por las aplicaciones móviles. Los usuarios se acostumbran cada vez más rápido al uso de los nuevos tipos de aplicaciones móviles, generando que sus intereses vayan variando a medida que estas aplicaciones evolucionan. Esto genera que estas aplicaciones se vayan volviendo más complejas y difíciles de mantener debido a que deben innovar constantemente para satisfacer los nuevos requerimientos o intereses de los usuarios.

Estas exigencias del mercado actual, requieren la creación constante de nuevas aplicaciones móviles, haciendo que, muchas veces, no se cuenten con los tiempos necesarios para la creación de las mismas y por lo tanto, se generan aplicaciones móviles específicamente para solucionar un problema, sin hacer hincapié en la metodología empleada en la creación de la misma. El problema surge cuando se debe realizar algún tipo de actualización o mejora en estas aplicaciones, ya que muchas veces se tiene que crear una nueva aplicación por no haber utilizado una metodología adecuada que permita realizar cambios o incorporar nuevos requerimientos. Por lo tanto, es necesario al momento de crear este tipo de aplicaciones tener en cuenta que los requerimientos de las mismas van variando en el tiempo y que estas aplicaciones evolucionan o requieren mantenimiento constante. Es decir, los requerimientos que deben tener en cuenta este tipo de aplicaciones van variando en el tiempo por lo que es necesario concebirlas considerando que las mismas cambian constantemente.

Otro aspecto fundamental a la hora de crear aplicaciones móviles, es la complejidad dada por las cuestiones de movilidad del usuario. Cuando un usuario utiliza una aplicación móvil puede, por ejemplo, distraerse fácilmente, perderse mientras realiza un camino, entre otros diversos factores que son impredecibles y tan complejos como el comportamiento humano. Por esta razón, las aplicaciones móviles deben crearse considerando poder adaptarse al comportamiento dinámico del usuario. Otra característica esencial de tipo de aplicaciones es poder representar los puntos de interés relevantes para la aplicación y donde los mismos están ubicados respecto del espacio físico en el cual se va a mover el usuario. Luego estos puntos de interés son los que sensen al usuario y la aplicación automáticamente le provee información del mismo.

Paralelamente al crecimiento de las expectativas de los usuarios, ha aumentado la variada gama de dispositivos móviles así también las características técnicas asociadas a los mismos. Esto genera, para los desarrolladores, la necesidad de tener en cuenta demasiadas variables tanto a la hora de crear como de mantener estas

aplicaciones. Estas variables, que deben tener en cuenta los desarrolladores, abarcan desde el sistema operativo del dispositivo móvil hasta cuestiones puntuales como el ancho de banda actual de la red en uso o sensores internos propios del dispositivo. Si el desarrollador crea aplicaciones que estén atadas a determinadas especificaciones técnicas hace que las mismas no se puedan reutilizar en otros dispositivo móvil. Es decir, se debe crear una aplicación por dispositivo, esto trae como problemática que los desarrolladores deben ser expertos en muchos aspectos puntuales de cada uno de los aspectos técnicos de los dispositivos móviles. En el caso de tener ligada la aplicación al tipo de dispositivo móvil se hace muy difícil rehusar las cuestiones propias del dominio (de la aplicación) para crear aplicaciones para otro tipo de dispositivo móvil. Por esta razón, las cuestiones propias del dominio de la aplicación deben ser concebidas de manera independiente a las características técnicas del dispositivo. También se debe considerar que las aplicaciones móviles no funcionan de manera aislada sino que muchas veces deben interactuar con productos ya existentes, más aún pueden llegar a interactuar, en un futuro, con productos que todavía no fueron creados. Esto requiere que este tipo de aplicaciones sean diseñadas pensando en que soportar interoperabilidad.

Las características antes mencionadas se enuncian en [Lehman, 1996] y [Nazim et al., 2006] como leyes de la evolución del software. Para el foco de esta tesis, se pueden mencionar las siguientes leyes (enunciadas en [Lehman, 1996] y [Nazim et al., 2006]): cambios continuos que requieren que la aplicación se adapte a nuevas situaciones, aumento de la complejidad a medida que la aplicación cambia, crecimiento continuo de la funcionalidad para satisfacer al usuario y disminución de la calidad en el tiempo.

De esta manera quedan planteadas distintas cuestiones que se deben tener en cuenta al momento de crear aplicaciones móviles. A partir de estas consideraciones surge la necesidad de contar con un enfoque que permita agilizar la tarea de crear aplicaciones móviles, en particular aplicaciones de Hipermedia Móvil. Este enfoque debe contar con mecanismos de abstracción y procesos de diseño, los cuales se vuelven cruciales para desarrollar este tipo de aplicaciones. En este sentido, la posibilidad de reuso brindado por los desarrollos dirigidos por modelos aceleran significativamente el proceso de diseño en general permitiendo que los desarrolladores trabajen con un nivel de abstracción alto despreocupándose por cuestiones puntuales de la plataforma en la que corra la aplicación. Es decir, se logra separar las cuestiones propias del dominio de la aplicación respecto de las características técnicas de los dispositivos. Una característica fundamental asociada al desarrollo dirigido por modelos, es permitir la derivación automática de código a partir de modelos facilitando incrementar la productividad y logrando que un mismo modelo pueda ser derivado a varias plataformas. Según [Atkinson and Kuhne, 2003], [Hailpern and Tarr, 2006], [Bézivin, 2006] y [Gherbi et al., 2009] los beneficios más destacados del desarrollo dirigido por modelo son: mejorar la productividad, permitir portabilidad, permitir soportar interoperabilidad y facilitar el mantenimiento de la aplicación.

Las características y beneficios descriptos del desarrollo dirigido por modelos, hacen que el mismo sea adecuado para llevar a cabo el enfoque necesario para crear aplicaciones de Hipermedia Móvil. Este tipo de desarrollo va a facilitar particularmente la creación de aplicaciones de Hipermedia Móvil y además, va a permitir trabajar en un alto nivel de abstracción, en particular, el desarrollo de la aplicación sin pensar en el tipo de dispositivo móvil que se utiliza independizando ambos niveles.

El enfoque, además, debe considerar tanto los aspectos de movilidad del usuario como los puntos de interés y donde están ubicados. También debe permitir representar los conceptos propios de las aplicaciones de Hipermedia Móvil, por ejemplo el concepto de link entre objetos del mundo real. Es fundamental que el enfoque soporte la creación de aplicaciones de Hipermedia Móvil para diferentes dominios.

1.2. Objetivo de la Tesis

El objetivo general de la tesis es desarrollar un enfoque de modelado que permita la especificación de aplicaciones de Hipermedia Móvil usando derivación semi-automática a partir de los modelos especificados. El enfoque elegido se basa en el desarrollo dirigido por modelos, esto quiere decir que no solamente se usan modelos para el modelado de las aplicaciones sino que se hace hincapié en la necesidad de contar con transformaciones entre los distintos modelos.

Nuestro enfoque permite la especificación de modelos que representan conceptos referidos tanto al dominio de la aplicación como de los aspectos navegacionales de la misma. Dentro de los aspectos navegacionales se consideran tanto aquellos relacionados a la navegación digital como así también los relacionados a la navegación en el mundo real.

El enfoque mencionado esta basado en conceptos avanzados de separación de concerns (típicos en la orientación a objetos y aspectos) y utiliza distintos tipos de concerns: paradigmáticos (como la navegación, la movilidad, etc.) o aplicativos (como diversos temas dentro de un dominio específico).

Los aportes realizados por esta tesis son:

- La determinación de los conceptos específicos de aplicaciones de Hipermedia Móvil, y que las diferencian de las aplicaciones de hipermedia convencionales y de las aplicaciones móviles.
- La definición de un enfoque que pueda ser utilizado para las aplicaciones de Hipermedia Móvil.
- La incorporación de elementos de separación avanzada de concerns tanto verticales como horizontales (también llamados aplicativos y paradigmáticos respectivamente).
- Disponer de un lenguaje genérico que pueda ser aplicado a dominios particulares, por ejemplo, asistencia turística y drama urbano móvil (*Mobile Urban Drama*) entre otros.
- Especificación e implementación de herramientas de transformación de modelos para derivar sucesivamente los modelos de nuestro enfoque hasta obtener aplicaciones ejecutables en el contexto del desarrollo dirigido por modelos.

1.3. Estructura de la Tesis

A continuación se detalla el contenido de los siguientes capítulos de esta tesis.

En el Capítulo 2 se introducen los conceptos básicos del desarrollo dirigido por modelo que son la base para el enfoque desarrollado en esta tesis. Por otro parte, se describen las características y beneficios del uso de separación de concerns como parte del desarrollo de aplicaciones, en particular se describe como dos metodologías de Hipermedia, como son *OOHDM* y *UWE*, usan la separación de concerns como parte de su metodología. Además, como las aplicaciones de Hipermedia Móvil pueden ser consideradas aplicaciones de realidad aumentada se detallan las características básicas de este tipo de aplicaciones. Se presentan algunas aplicaciones existentes de realidad aumentada relacionadas a aplicaciones móviles.

En el Capítulo 3 se presentan los conceptos básicos relacionados a Hipermedia Móvil como son, por ejemplo, la navegación digital y en el mundo real especificando las características particulares de cada una. Además, se describe en este capítulo cómo se utiliza la separación de concerns, en particular, en aplicaciones de Hipermedia Móvil. Se describe también, de manera conceptual, el enfoque definido

para modelar aplicaciones de Hipermedia Móvil, mostrando los diferentes modelos que se definen y sus principales características.

En el Capítulo 4 se describe la extensión que se realizó a la metodología de Hipermedia UWE (*UML-based Web Enginnering*). En este capítulo se muestran los elementos definidos a nivel de metamodelo para modelar los conceptos propios de las aplicaciones de Hipermedia Móvil. Además, se detallan cómo cada uno de los conceptos creados a nivel de metamodelo es representado usando un perfil de *UML* (*Unified Model Language*), logrando así crear una extensión conservativa de *UWE*.

Los modelos de nuestro enfoque son detallados de manera concreta en el Capítulo 5 destacando las características propias de cada uno de ellos e indicando que elementos de nuestro perfil se utiliza en cada uno. Además se especifica cómo funciona cada una de las transformaciones de modelos de nuestro enfoque.

En el Capítulo 6 se presenta nuestra herramienta *MagicMobileUWE*, que permite crear los modelos de nuestro enfoque y realizar nuestras transformaciones entre modelos. Se usan ejemplos para mostrar cómo utilizar esta herramienta para crear cada uno de nuestros modelos y cómo se realiza cada una de las transformaciones. Se hace hincapié en indicar cómo el desarrollador (diseñador) puede usar nuestra herramienta para agilizar su tarea de crear aplicaciones de Hipermedia Móvil.

Tres ejemplos de aplicaciones móviles con diferentes características se presentan en el Capítulo 7, mostrando como mediante el uso de nuestra herramienta *MagicMobileUWE* se pueden crear estas aplicaciones usando nuestro enfoque. Uno de los ejemplos es la típica aplicación de asistencia turística, los otros dos ejemplos esta basados en el concepto de *Mobile Urban Drama* [Hansen et al., 2008].

Nuestra herramienta *MagicMobileUWE* permite derivar dos posibles tipos de aplicaciones: aplicaciones *Java* para dispositivos y aplicaciones Web móviles. En el Capítulo 8 se muestra mediante ejemplos cómo derivar ambos tipos de aplicaciones indicando la estructura de las aplicaciones resultantes y cómo el usuario percibe cada una cuando las utiliza en su dispositivo móvil. Se muestra cómo las derivaciones son realizadas a partir de los modelos de nuestro enfoque.

En el Capítulo 9 se detallan distintos trabajos relacionados con esta tesis. Por un lado se presentan trabajos relacionados desde el punto de vista de las aplicaciones que permite derivar nuestro enfoque, especificando sus características principales y comparándolos con las aplicaciones (de Hipermedia Móvil) que permite modelar nuestro enfoque. Por otro lado, se presentan las principales características de dos trabajos relacionados con nuestro enfoque y se realiza una comparación con los mismos.

Se detallan en el Capítulo 10 las conclusiones obtenidas como resultado del enfoque presentado en esta tesis, indicando cómo se cumplieron cada uno de los objetivos propuestos para esta tesis. Además, se detallan distintos trabajos futuros relacionados tanto sea con nuestro enfoque como con nuestra herramienta.

2. BACKGROUND

En este capítulo se presentan los conceptos principales de tres temáticas relacionadas con la tesis: el desarrollo dirigido por modelos, la separación de concerns y las aplicaciones de Realidad Aumentada.

Se describen los conceptos básicos del desarrollo dirigido por modelos, los cuales forman la base de nuestro enfoque. En particular se presenta un estándar concreto de este tipo de desarrollo, MDA. Luego se describen distintas metodologías de desarrollo de aplicaciones de Hipermedia, también desde el punto de vista del desarrollo dirigido por modelos.

Se presentan las características y beneficios de la separación de concerns según diferentes autores, haciendo foco particularmente en mostrar el concepto de separación de concerns en aplicaciones Web.

Como las aplicaciones de Hipermedia Móvil cuentan con algunas características de las aplicaciones de realidad aumentada, por este motivo, se introducen en este capítulo las características principales de este tipo de aplicaciones mostrando en particular ejemplos de aplicaciones de realidad aumentada para dispositivos móviles. Además, se presentan las aplicaciones de Hipermedia Física como una forma de aumentar los objetos del mundo real mediante información Hipermedial. Las aplicaciones de Hipermedia Móvil son un caso particular de aplicación de Hipermedia Física donde se considera la movilidad del usuario y el contexto del mismo.

2.1. Desarrollo Dirigido por Modelos

El desarrollo dirigido por modelos (*Model-Driven Development - MDD*) es una metodología basada en el desarrollo de la aplicación a partir de modelos y mediante el uso de herramientas adecuadas generar una aplicación a partir de los mismos.

Veamos cuales son los objetivos del desarrollo dirigido por modelo según la visión de distintos autores:

- En [Gherbi et al., 2009]¹ se indica que la motivación del desarrollo dirigido por modelos es aprovechar los esfuerzos empleados en el análisis y concepción de las aplicaciones facilitando la migración de aplicaciones de una plataforma a otra. Se hace hincapié en la importancia de modelar una única vez y luego generar aplicaciones para distintas plataformas.
- Según [Atkinson and Kuhne, 2003] el objetivo es automatizar las tareas complejas buscando reducir el tiempo de creación de las aplicaciones. En particular hacen hincapié en la derivación automática de aplicaciones.
- El objetivo que destacan los autores en [Hailpern and Tarr, 2006] es reducir las tareas del desarrollador y la complejidad de las aplicaciones para lo cual elevan el nivel de abstracción usando modelos para crear aplicaciones y hacer que las mismas evolucionen.

En los tres trabajos mencionados se hace hincapié en la importancia de separar los modelos del código ya que esto permite lograr independencia de plataformas; posibilitando crear aplicaciones para diferentes plataformas a partir de los mismos modelos. Los autores coinciden en el uso de herramientas que generen código a partir de los modelos especificados.

¹ Se usa en [Gherbi et al., 2009] el término ingeniería dirigida por modelos (*Model-Driven Engineering - MDE*) como sinónimo del desarrollo dirigido por modelos.

Los beneficios más destacados del desarrollo dirigido por modelo, según [Atkinson and Kuhne, 2003], [Hailpern and Tarr, 2006], [Bézivin, 2006] y [Gherbi et al., 2009], son:

- *Productividad.* El desarrollo dirigido por modelos permite aumentar la productividad debido a que los desarrolladores trabajan en un nivel de abstracción alto, facilitando que se concentren en los aspectos importantes del dominio de la aplicación, sin tener que aprender aspectos técnicos de los dispositivos, logrando de esta manera que se agilicen los tiempos del desarrollo. Además, contar con transformaciones, permite poder derivar desde un mismo modelo aplicaciones para distintos dispositivos, aumentando de esta manera la productividad ya que se logra reusar todas las cuestiones propias del dominio de la aplicación bajando de esta manera los tiempo en el desarrollo de las aplicaciones.
- *Portabilidad.* La portabilidad se logra al tener modelos independientes del dispositivo (o plataforma) donde se ejecute la aplicación. Estos modelos (independientes de la plataforma) pueden ser portables en el sentido de poder usarlos para derivar diferentes tipos de aplicaciones.
- *Reusabilidad.* Los diferentes modelos pueden ser reutilizados para derivar tanto otros modelos como así también aplicaciones para diferentes plataformas. Es decir, un modelo puede servir para derivar infinitos modelos o aplicaciones, sólo se necesita tener definida la transformación adecuada en la herramienta que permite llevar a cabo el desarrollo dirigido por modelos.
- *Interoperabilidad.* Tener transformaciones de un modelo a otro o desde un modelo a código hace que estos tengan definidos claramente la relación entre el origen y el destino de la transformación. Esto permite que haya interoperabilidad tanto a nivel de modelos como de código.
- *Mantenimiento.* Realizar cualquier cambio en la aplicación significa modificar los modelos y luego correr las transformaciones necesarias para llegar a tener el código con los cambios. Las transformaciones hacen que las modificaciones se propaguen hasta el código facilitando de esta manera la evolución de la aplicación.
- *Complejidad.* El desarrollo dirigido por modelos permite reducir la complejidad de las aplicaciones, ya que se trabaja con modelos que tienen diferentes niveles de abstracción. Además, las transformaciones generan de manera semi-automática tanto modelos como código, logrando así tener aplicaciones a partir de la definición de modelos.
- *Documentación.* Los modelos son una exacta representación de la aplicación que se crea a partir de los mismos, por lo tanto los modelos sirven como documentación con un alto nivel de precisión.

En el desarrollo dirigido por modelo se especifican en [Bézivin, 2006] distintos niveles de modelado, que se pueden apreciar en la Figura 2.1 (*C2* es la abreviatura usada para *conformsTo* y *repOf* para *representationOf*):

- El meta-metamodelo (o M3). Es un modelo que sirve para definir los conceptos del metamodelo (o tipos de metamodelos) y se especifica usando sus propios elementos (esto se puede apreciar en la figura con la relación *C2* que tiene el metametamodelo a si mismo).
- El metamodelo (o M2). Es un modelo que sirve para definir los conceptos del modelo (o tipos de modelos) y se especifica en base a los elementos definidos en el meta-metamodelo.
- El modelo (o M1). Es un modelo que sirve para especificar los conceptos particulares de una aplicación (o sistema) y se especifica en base a los elementos definidos en su metamodelo. Este modelo es el que especifica el

desarrollador de la aplicación.

- El sistema (indicado en la figura como *aSystem*), también se puede referenciar como M0. Es una instancia particular del modelo, donde se especifican datos reales de una aplicación. Los datos particulares son especificados por el desarrollador acorde a la aplicación que se está representado.

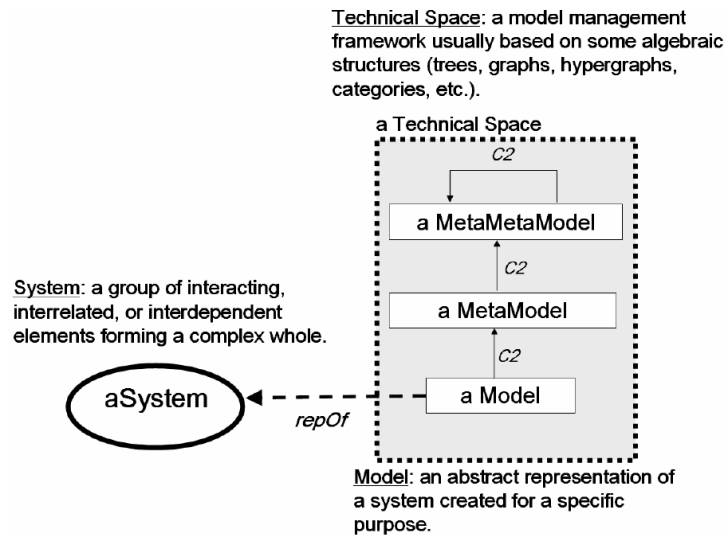


Figura 2.1: Niveles de modelado en el desarrollo dirigido por modelos [Bézivin, 2006].

Si bien hay muchas variantes a la hora de implementar un desarrollo dirigido por modelos, en [Bézivin, 2006] y [Gherbi et al., 2009] se especifican tres principios básicos que todas comparten:

- Representación directa. Usar lenguajes específicos de dominio (*Domain-Specific Language - DSL*) permitiendo la representación de cada concepto o situación propia de cada tipo de aplicación.
- Automatización. Permitir la relación de manera automática entre los modelos facilitando la transformación de un modelo a otro como así también la generación de aplicaciones.
- Uso de estándares. Facilitando el uso por parte de los desarrolladores y aprovechando las herramientas existentes (permitiendo interoperabilidad).

En la Figura 2.2, se puede apreciar por un lado la especificación de los principios relacionados al desarrollo dirigido por modelos (en este caso se usa como sinónimo el término *Model-Driven Engineering - MDE*), por otro lado diferentes estándares que implementan cada uno de manera particular estos principios y además algunas de las herramientas que facilitan este tipo de desarrollo.

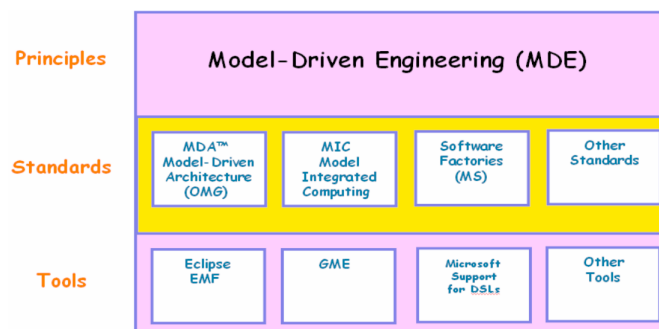


Figura 2.2: Principios, estándares y herramientas relacionadas con el desarrollo dirigido por modelos [Bézivin, 2006].

En [Hailpern and Tarr, 2006] se plantea una problemática que se debe considerar cuando se realiza un desarrollo dirigido por modelos como es la sincronización entre modelos. La problemática se plantea cuando se realizan cambios en el código y cómo poder reflejar los mismos en los modelos. Además, se plantea cómo reflejar los cambios realizados a un modelo en el resto de los modelos relacionados, y de esta manera tener información consistente en todos los modelos.

2.1.1. Una implementación particular de desarrollo dirigido por modelos (*Model-Driven Architecture - MDA*)

En la sección anterior se mostró en la Figura 2.2 que había varios estándares de desarrollo dirigido por modelos, en particular uno de ellos es *MDA - Model-Driven Architecture*² (arquitectura dirigida por modelos). Este estándar, que fue propuesto y patrocinado por la organización *Object Management Group (OMG)*³ es uno de los más difundidos actualmente.

En la Figura 2.3 se muestran los niveles de modelados particularmente para *MDA*. Se puede apreciar que el modelo M3 (o sea el nivel que se corresponde con el meta-modelo) está definido por *Meta-Object Facility (MOF)*⁴. *MOF* es un lenguaje propuesto por la organización *OMG* como un estándar para definir lenguajes de modelado.

En la figura se puede ver que como en el nivel M2 (o metamodelo) se tiene los conceptos de *Unified Modeling Language (UML)*⁵. *UML* también es un estándar propuesto por la organización *OMG* y se define en base a *MOF* (es decir que *MOF* es el metamodelo usado para la definición de *UML*).

Se puede apreciar además que en el nivel M1 (o modelo de la aplicación) se especifica en la figura como “*User concepts*”, esto representa un modelo *UML* que tenga la especificación de una aplicación particular. Y por último el modelo M0 que en la figura está indicado como “*Data concepts*”, representa un modelo de instancia particular del modelo *UML* especificado en el nivel M1.

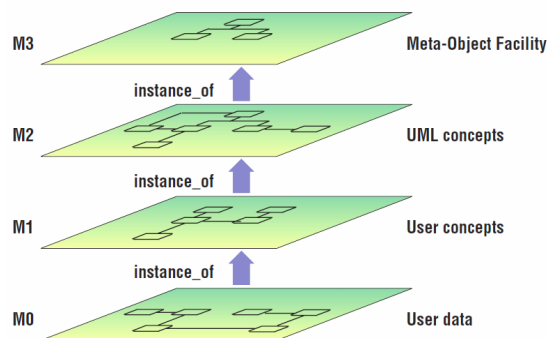


Figura 2.3: Niveles de modelado en *MDA* [Atkinson and Kuhne, 2003].

MDA define tres puntos de vista diferentes para analizar y representar un sistema, según [Gasevic et al., 2006], [Gherbi et al., 2009], [Singh and Sood, 2009] y [Little, 2010], estos son:

- *Computation Independent Viewpoint*. Se focaliza en el ambiente del sistema los requerimientos del mismo; los detalles de las estructuras y el procesamiento del sistema pueden estar ocultos o no.

² Página de *MDA*: <http://www.omg.org/mda/>

³ Página de *OMG*: <http://www.omg.org>

⁴ Página de *MOF*: <http://www.omg.org/mof/>

⁵ Página de *UML*: <http://www.uml.org/>

- *Platform Independent Viewpoint*. Se focaliza en las operaciones de sistema sin tener en cuenta cuestiones particulares de una plataforma determinada. Es una visión del sistema común para cualquier plataforma en la cual se lleve a cabo.
- *Platform Specific Viewpoint*. Se focaliza en especificar cuestiones particulares de una plataforma específica en la cual se lleva a cabo el sistema.

Para cada uno de estos tres puntos de vista *MDA* asocia una representación o modelo del sistema. Los tres modelos respectivamente son:

- Modelo de requerimientos o también conocido como *Computation-Independent Model (CIM)*. Es un modelo que describe los requerimientos de la aplicación (o sistema) sin referirse a una implementación particular, es un modelo de dominio.
- Modelo de análisis y diseño o también mencionado en la bibliografía como *Platform-Independent Model (PIM)*. Es un modelo que permite especificar de manera abstracta los conceptos, estructuras y funcionalidad de la aplicación (o sistema) sin especificar ningún detalle de la plataforma o sea debe ser independiente de la misma.
- Modelo de Código o *Platform-Specific Model (PSM)*. Es un modelo que describe en base a una implementación particular la aplicación (o sistema).

Esta caracterización de tres tipos de modelos (*CIM*, *PIM* y *PSM*) no solamente es usada por *MDA*, sino que otros desarrollos dirigidos por modelos también adoptan esta clasificación para caracterizar sus modelos.

Cada uno de los modelos mencionados en la clasificación (*CIM*, *PIM* y *PSM*) puede capturar una vista particular de una aplicación (o sistema). Por lo tanto, una aplicación completa puede contar con varios modelos de requerimientos, varios modelos de análisis-diseño y varios modelos de código (para poder tener especificada la aplicación para distintas plataformas).

En *MDA* las transformaciones no se dan solamente en un sentido, es decir desde los modelos más abstractos a los modelos específicos (*CIM* → *PIM* y *PIM* → *PSM*) sino que además se dan las transformaciones ejemplificadas mediante relaciones en la Figura 2.4 y descritas en [Liddle, 2010] como:

- *CIM* → *CIM*, sirve para realizar refinamiento de modelos *CIM*.
- *PIM* → *PIM*, permite realizar el refinamiento de modelos *PIM* necesario cuando se pasa de la fase de análisis a la etapa de diseño.
- *PSM* → *PSM*, esta transformación se necesita para configurar o empaquetar para un ambiente particular un *PSM*.
- *PIM* → *CIM*, se utiliza cuando hay que hacer reingeniería y reflejar los cambios de un modelo *PIM* en la especificación de requerimientos (*CIM*).
- *PSM* → *PIM*, esta transformación se utiliza cuando hay que reflejar algún cambio de un modelo *PSM* a un *PIM*, es decir hacer reingeniería.

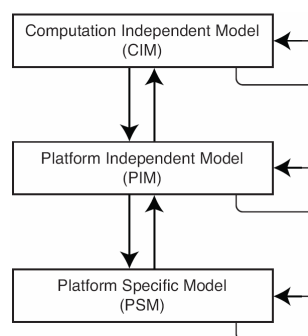


Figura 2.4: Modelos y transformaciones en *MDA* [Liddle, 2010].

Para cada uno de los modelos presentados (*CIM*, *PIM* y *PSM*) como se indica en [Gherbi et al., 2009] se debe tener su metamodelo correspondiente como se puede apreciar en la Figura 2.5, en donde cada uno de estos metamodelos está definido en base a *MOF*.

Puede suceder que *UML* no sea suficiente como metamodelo de estos modelos (*CIM*, *PIM* y *PSM*) y se deba usar otro estándar como metamodelo (por ejemplo el uso de perfiles *UML* para representar conceptos específicos o lenguajes específicos de dominio). Por esta razón en la figura no se especifica ningún metamodelo en particular sino que se indican de manera general la necesidad de contar con uno.

Se puede observar (en la figura) que también se especifican los modelos de las transformaciones con sus correspondiente metamodelo, esto facilita la automatización de las mismas ya que se cuenta con un metamodelo bien definido.

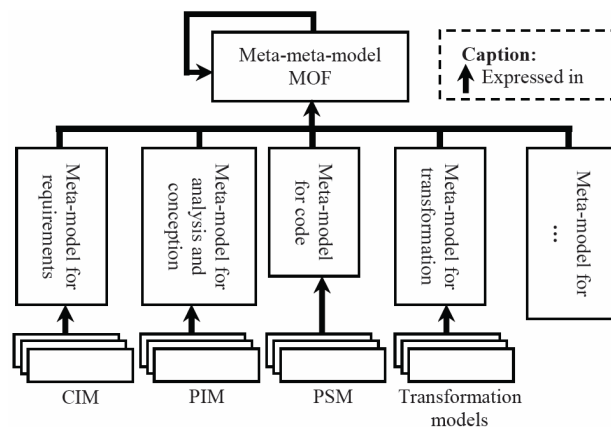


Figura 2.5: Modelos, Metamodelos y Metametamodelo en MDA [Gherbi et al., 2009].

Otros estándares (a parte de *MOF* y *UML*) usados por *MDA* y especificados por la organización *OMG* son:

- *XML Metada Interchange (XMI⁶)* que permite expresar en forma de archivo *XML*⁷ (*Extensible Markup Language*) cualquier modelo (o metamodelo) definido en base a *MOF*.
- *Query/View/Transformation (QVT⁸)* permite definir el modo en que se llevan a cabo las transformaciones entre modelos (los cuales deben estar definidos en base a *MOF*).

2.1.2. Desarrollo dirigido por modelos en aplicaciones Web

Hasta el momento, se describieron en las secciones anteriores las características básicas que tiene el desarrollo dirigido por modelo y en particular los conceptos básicos relacionados a un estándar (que implementa el desarrollo dirigido por modelos) como es *MDA*. Pero no se detalló en qué dominios se pueden llevar a cabo este tipo de desarrollo. El desarrollo dirigido por modelos se puede aplicar en varios dominios, en particular se aplica para el desarrollo de aplicaciones Web como se describe en [Koch et al., 2008b], [Moreno et al., 2008b] y [Liddle, 2010] denominándose Ingeniería Web dirigida por modelos (*Model-Driven Web Engineering - MDWE*).

Hipermedia Móvil se puede ver como una extensión de Hipermedia donde se agrega la complejidad de la movilidad del usuario. Por este motivo, es interesante describir cómo

⁶ Página de *XMI*: <http://www.omg.org/spec/XMI/>

⁷ Página de *XML*: <http://www.w3.org/XML/>

⁸ Página de *QVT*: <http://www.omg.org/spec/QVT/>

se aplica el desarrollo dirigido por modelos en las aplicaciones Web (o de Hipermedia). A continuación se describen algunos aspectos particulares para esta tesis, de la Ingeniería Web dirigida por modelos (de ahora en adelante mencionada como *MDWE*). Se describen en [Liddle, 2010] varios métodos de *MDWE* donde cada uno soporta niveles diferentes del desarrollo dirigido por modelos. Es decir, que no todos cumplen en forma completa con todos los objetivos que tiene este tipo de desarrollo. Las aplicaciones Web, tienen respecto a otras, la particularidad de soportar la operación de navegación [Gaedke and Meinecke, 2008]; por lo tanto, es necesario contar con modelos navegacionales que representen esta información. Es decir, dentro del desarrollo dirigido por modelos para aplicaciones Web es importante tener representado el modelo navegacional. Este modelo debe ser un *PIM*, es decir independiente de la plataforma.

Según [Liddle, 2010] las metodologías que poseen el conjunto de herramientas más completo para el desarrollo dirigido por modelos son *WebML* (*Web Modeling Language*) [Brambilla et al., 2008] y *UWE* (*UML-based Web Engineering*) [Koch et al., 2008]. Veamos algunos detalles de cada una de estas dos metodologías (*WebML* y *UWE*):

WebML

Es una metodología de modelado Web orientada a datos, esto se debe a que el modelo conceptual de una aplicación se representa usando un modelo de *ER* (*Entity Relationship*). Para esta metodología (*WebML*) se cuenta con la herramienta *WebRatio*⁹, para el desarrollo dirigido por modelos. Los requisitos se expresan a través de un modelo de alto nivel y el código de la aplicación se genera automáticamente (mediante el uso de reglas). El código resultante es una aplicación *Java*¹⁰ basada en estándares Web, ya que *WebRatio* está integrado a la plataforma *IDE* de *Eclipse*¹¹. Esta herramienta (*WebRatio*) se compone, para el desarrollo de aplicaciones, de tres pasos básicos: construcción de los modelos (en particular, el modelo de la aplicación se realiza utilizando *WebML*), personalización de las reglas y generación de la aplicación.

UWE

Es una metodología de modelado Web definida como una extensión conservativa de *UML*¹² (*Unified Modeling Language*). Es decir, *UWE* define un perfil *UML* [Fuentes and Vallecillo, 2004] para representar los elementos propios de su modelos y de esta manera no agregar nuevos elementos al metamodelo de *UML* (*MOF*). Por lo tanto, tiene las características adecuadas para el desarrollo dirigido por modelos ya que se basa en metamodelos para la definición de sus modelos (en [Kraus et al., 2007] se puede leer más detalle sobre desarrollos dirigidos por modelos usando *UWE*).

UWE cuenta con un plugin para *MagicDraw*¹³ (herramienta de modelado *UML*) llamado *MagicUWE*¹⁴ [Busch and Koch, 2009]. Este plugin permite la especificación de sus modelos y la realización de transformaciones entre los mismos.

Además, provee la herramienta *UWE4JSF* [Kroiss and Koch, 2009] que se integra a *Eclipse* para generar aplicaciones Web a partir de los modelos de *UWE*. Esta herramienta genera aplicaciones basadas en el framework de *JSF*¹⁵ (*JavaServer Faces Technology*).

⁹ Página de *WebRatio*: <http://www.webratio.com>

¹⁰ Página de *Java*: <http://www.oracle.com/technetwork/java/index.html>

¹¹ Página de *Eclipse*: <http://www.eclipse.org>

¹² Página de *UML*: <http://www.uml.org/>

¹³ Página de *MagicDraw*: <http://www.magicdraw.com/>

¹⁴ Página de *MagicUWE*: <http://uwe.pst.ifi.lmu.de/toolMagicUWE.html>

¹⁵ Página de *JSF*: <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>

Además se describe en [Liddle, 2010] la metodología *OOHDM (Object Oriented Hypermedia Design Method)* [Rossi and Schwabe, 2008] como una de las primeras en aplicar *MDWE*. Basada en esta metodología, *OOHDM* posee una herramienta denominada *HyperDE*¹⁶ que genera sistemas *Ruby* para plataformas *Rails*; esta herramienta es adecuada particularmente para la creación de prototipos de aplicaciones sobre la Web semánticas. Se hace mención a la metodología *OOHDM*, ya que los conceptos de separación de concerns [Nanard et al., 2008] aplicados a la misma se usaron como base de esta tesis.

En [Schwinger et al., 2008] se realiza una investigación comparando distintas metodologías de modelado Web en base a distintos aspectos. En particular se analizan (las metodologías Web) desde el punto de vista del desarrollo dirigido por modelos. En la Figura 2.6 se puede apreciar un resumen del análisis realizado por los autores donde sólo se hace foco en las tres metodologías (*OOHDM*, *WebML* y *UWE*) que se mencionaron anteriormente.

	Language Definition (M.L)			Model Transformation Type (M.T)				
	Metamodel	Grammar	Semantic Description	PIM2PIM	PIM2PSM	PIM2Code	PSM2Code	Platform Description Model (M.P)
WebML		DTD				Struts		
OOHDM	~		RDFS, OWL					
UWE	MOF			✓		J2EE/ Cocoon		

Legend:

✓	supported
	not supported

Figura 2.6: Desarrollo dirigido por modelos en las distintas metodologías [Schwinger et al., 2008].

2.2. Separación de Concerns

La evolución del software es un aspecto fundamental que se debe tener en cuenta a la hora de diseñar y desarrollar aplicaciones. En [Lehman, 1996] y [Nazim et al., 2006] se describe un conjunto de leyes relacionadas con la evolución del software, a continuación se describen aquellas leyes más relevantes para esta tesis:

- Cambios continuos. Una aplicación debe adaptarse constantemente a nuevas situaciones, de lo contrario se convierte en obsoleto.
- Aumento de la complejidad. A medida que un programa cambia, se incrementa su complejidad y se vuelve más difícil que el mismo evolucione a menos que se trabaje para mantenerlo o reducir la complejidad.
- Crecimiento continuo. La funcionalidad brindada por una aplicación debe incrementarse constantemente para lograr mantener la satisfacción del usuario, logrando de esta manera un mayor tiempo de vida de la aplicación.
- Disfunción de la calidad. La calidad de una aplicación tiende a disminuir con el paso del tiempo, a menos que la misma evolucione considerando los cambios del entorno operativo.

A partir de estas leyes enunciadas se puede apreciar las consideraciones que se deben tener en cuenta para que una aplicación pueda evolucionar. Es fundamental que una aplicación pueda adaptarse a nuevas funcionalidades, logrando mantener su calidad y además, que la complejidad agregada por la nueva funcionalidad permita que la aplicación pueda seguir evolucionando. Una aplicación debe concebirse considerando que la misma va a evolucionar y por lo tanto prevenir la complejidad

¹⁶ Página de *HyperDE*: <http://www.tecweb.inf.puc-rio.br/hyperde>

agregada por los cambios, logrando que los mismos no generen problemas en futuras evoluciones.

En [Parnas, 1972] se describe el concepto de modularización como un mecanismo para mejorar la flexibilidad y comprensión de un sistema, al mismo tiempo que permite la reducción del tiempo de desarrollo. El autor menciona y demuestra que la eficacia de la modularización depende del criterio que se use para dividir los módulos, es decir si el criterio usado no es adecuado puede ser que el sistema no sea flexible a cambios. Una buena modularización permite un mejor mantenimiento del sistema ya que cada módulo evoluciona de manera independiente.

Los conceptos presentados en [Parnas, 1972] fueron la base para el trabajo presentado en [Tarr et al., 1999], donde los autores describen la separación de concerns multidimensional. Los autores mencionan que para reducir la complejidad del software se requiere de mecanismos de descomposición para lograr piezas manejables, pero además contar con mecanismos de composición para lograr conectar estas piezas de manera adecuada para lograr un software útil. Es decir, los autores definen un concern como unidades (piezas) de software. Esta forma de reducir la complejidad permite que el mantenimiento y la evolución del software generen el menor impacto posible, ya que cada pieza se trata de manera aislada. Los autores presentan un enfoque que permite la descomposición y composición de concerns en múltiples dimensiones en forma simultánea. Demostrando cómo la separación de concerns permite simplificar el mantenimiento y la evolución del software. También mencionan que el uso de separación de concerns en la ingeniería de software trae como beneficios reducir la complejidad, mejorar la reusabilidad y simplificar las problemáticas que vienen relacionadas con la evolución del software.

El trabajo descrito por los autores en [Sutton and Rouvellou, 2002] toma como base a [Parnas, 1972] y a [Tarr et al., 1999], los autores presentan un esquema de modelado de concern-space de propósito general llamado *Cosmos (COncern-Space MOdeling Schema)*. Este esquema representa a los concerns como clases para lo cual los autores indican que deben definir cada concern de manera independiente de cualquier tipo de artefacto de software. Se considera un concern como cualquier cuestión de interés del sistema de software. Los autores definen el concepto de concern-space como una representación organizada de los concerns y sus relaciones, además identifican dos tipos de los concerns: lógicos y físicos. Los concerns lógicos representan los conceptos de interés del sistema, como por ejemplo, características o propiedades. Los concerns físicos representan elementos del sistema que se aplican a los concerns lógicos. Esta categorización de concerns lógicos y físicos les permite a los autores modelarlos de manera independiente entre sí.

En [Ossher and Tarr, 2000] se describe que una clara separación de concerns permite reducir la complejidad y comprensión del software como así también promover la trazabilidad del mismo. Los autores indican que una adecuada separación de concerns permite limitar el impacto que provocan los cambios del software, facilitando de esta manera la evolución y la adaptación no invasiva. Otro beneficio, que se menciona, es facilitar el reuso como así también simplificar la integración entre componentes. Indican que la incorporación de un nuevo concern en cualquier etapa del ciclo de vida del software no debe afectar a los concerns existentes.

En [Hafedh et al., 2004] los autores afirman que la separación de concerns es una técnica que permite la resolución de un problema general siguiendo la siguiente heurística, primero resolviendo sus restricciones por separado para luego combinar las soluciones parciales, buscando que estas partes se puedan componer logrando una solución óptima. Para lograr buenos resultados, los concerns deben ser tratados de

manera independiente para que no interfieran entre si, pero no perdiendo de vista que luego deben componerse. Los autores proponen en [Hafedh et al., 2004] un framework conceptual basado en transformaciones del software, pudiendo distinguir entre requerimientos separables e inseparables.

En [Moreira et al., 2005] proponen un modelo de separación de concerns multidimensional a nivel de requerimientos. Los requerimientos son tratados de manera uniforme ya sean funcionales o no, es decir en este modelo los concerns son una colección de requerimientos. Los autores usan una matriz para indicar las relaciones existentes entre los diferentes concerns, identificando de esta manera cuales son concerns transversales (crosscutting concerns).

2.2.1. Separación de concerns en aplicaciones Web

La separación de concerns se aplica a la Ingeniería de Software, en particular a la Ingeniería Web. En esta sección se presentan dos trabajos relacionados a la separación de concerns en Ingeniería Web, como son *OOHDM* [Nanard et al., 2008] y *UWE* [Koch et al., 2008a].

OOHDM

Los autores describen en [Nanard et al., 2008] cómo la separación de concerns permite mejorar la navegación en las aplicaciones Web, además del beneficio de mejorar la evolución y el mantenimiento de la misma. Los autores introducen el término de navegación sensible al concern (*concern-sensitive navigation*) como una herramienta conceptual y práctica para mejorar la estructura navegacional de las aplicaciones Web. La navegación sensible al concern identifica por un lado las propiedades de un objeto, las cuales son denominadas propiedades core, y además las propiedad adicionales que le agrega cada concern. Cuando un usuario está navegando por un determinado concern puede ver el objeto actual enriquecido acorde con las propiedades que ese concern le agrega. Es decir, la información que el usuario recibe de un objeto, depende del concern en el cual está navegando. Los autores presentan en [Nanard et al., 2008] y en [Firmenich et al., 2010] como estos conceptos pueden ser modelados usando la metodología *OOHDM*, presentando en el modelo navegacional las propiedades que cada concern agrega al objeto (core) como roles del mismo. En la Figura 2.7 se puede apreciar un ejemplo de modelo navegacional, donde el nodo *Product* tiene un core y sus roles (*Recomm Product* y *ProductInCart*).

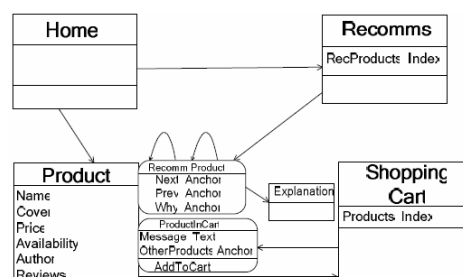


Figura 2.7: Modelo Navegacional de *OOHDM* considerando navegación sensible por concern [Nanard et al., 2008].

En ambos trabajos, los autores presentan dos niveles de separación de concerns: horizontal y vertical. La separación de concerns horizontal, está dada por las diferentes capas de modelado como son: navegación, presentación, proceso de negocios, etc. Esto lo representan en *OOHDM* con cada uno de los modelos que integran la metodología, y además tienen definido cómo cada uno de estos modelos se relacionan entre si. Si bien la separación de concerns horizontal es usada en

otras metodologías Web, la separación de concerns vertical es la principal diferencia presentada por los autores en [Nanard et al., 2008] y [Firmenich et al., 2010] respecto de otras metodologías Web. La separación de concerns vertical está relacionada íntegramente con la esencia de la aplicación, por lo tanto, para cada aplicación se definen sus concerns verticales acorde a su dominio. Estos concern verticales ayudan, no solamente a facilitar la evolución de la aplicación, sino que además permiten mejorar la navegación del usuario por concerns. Los autores muestran cómo estos conceptos se pueden usar para mejorar aplicaciones existentes, enriqueciéndolas con información de concerns particulares.

UWE

La metodología *UWE* describe en [Koch et al., 2008a] la utilización de la separación de concerns desde las primeras etapas del desarrollo mediante un desarrollo dirigido por modelos, el cual les permite construir la aplicación mediante el uso de modelos (separación de concerns horizontal) y transformaciones. Es decir, el desarrollo dirigido por modelos facilita poder tener separación de concerns horizontales. Los modelos en *UWE* son usados para representar las distintas vistas de la misma aplicación Web y los mismos se corresponden con diferentes concerns (contenido, navegacional, presentación, etc.). Por otro lado, las transformaciones del desarrollo dirigido por modelos de *UWE* les aseguran a los autores relacionar los distintos modelos de la metodología mediante una definición a nivel de metamodelo.

Los autores en [Moreno et al., 2008a] muestran cómo se incorporó un nuevo concern a *UWE* que permite el modelado del procesamiento de negocios a nivel navegacional, por ejemplo, el procesamiento que se debe realizar cuando se realiza una compra de un producto. Es decir, aquellas cuestiones que requieren procesamiento de información, por ejemplo, envío de datos de un formulario. Por lo tanto, el modelo navegacional de *UWE* queda conformado con nodos navegables y clases de procesamiento. Incorporar este nuevo concern a la metodología *UWE* requirió incorporar nuevos elementos en su metamodelo para poder representar los nuevos conceptos en el modelo navegacional. Además, este nuevo concern requirió la definición de un nuevo modelo, para representar en detalle cada uno de los procesos de negocios que se definen en el modelo navegacional. La incorporación de este nuevo modelo está facilitado por la flexibilidad que provee el desarrollo dirigido por modelo, teniendo además que definir la transformación acorde al nuevo modelo.

2.3. Realidad Aumentada

En [Milgram and Kishino, 1994] se define el término Mixed Reality como cualquier espacio entre los extremos del continuo de la virtualidad. Este continuo de la virtualidad, se extiende desde el mundo completamente real hasta el entorno completamente virtual, encontrándose entremedio las aplicaciones de realidad aumentada y realidad virtual. En la Figura 2.8 se puede visualizar como se define Mixed Reality como un conjunto de varios conceptos que van desde ambientes Reales a Virtuales.

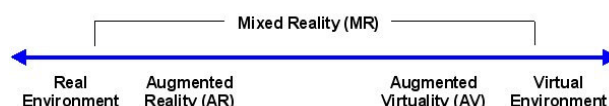


Figura 2.8: Explicación de Paul Milgram sobre MR [Milgram and Kishino, 1994].

De los conceptos que engloba Mixed Reality a continuación nos focalizaremos en aquellos más cercanos a los ambientes reales y de realidad aumentada.

Realidad aumentada (Augmented Reality [Mackay, 1998]) es una variación de los

ambientes virtuales (Virtual Enviroments), o realidad virtual como se llama más comúnmente. Las tecnologías de ambientes virtuales sumergen totalmente al usuario dentro de un ambiente sintético. Mientras que está sumergido, el usuario no puede ver el mundo real que lo rodea. En contraste, la Realidad Aumentada permite que el usuario vea el mundo real, con los objetos virtuales sobrepuestos sobre o compuestos con el mundo verdadero. Por lo tanto, suplementa la realidad, en vez de sustituirla completamente. Idealmente, el usuario percibe que los objetos virtuales y reales coexisten en el mismo espacio. Es decir, en lugar de sumergir a las personas en un mundo virtual artificialmente creado, la meta es aumentar los objetos del mundo real reforzándolos con una riqueza de información digital y capacidades de comunicación.

La información digital a menudo aparece como demasiado desconectada del mundo físico. Por lo general se utilizan los objetos reales en un ámbito, y la información digital relacionada en otro totalmente separado. La Realidad Aumentada se dirige a los problemas de re-integrar la información con el mundo real. Se tiene como meta permitir a las personas que aprovechen las habilidades existentes actuando recíprocamente con los objetos del mundo cotidiano mientras se benefician con el poder de conectarse a la información disponible.

Se puede aumentar la realidad a partir de estas tres estrategias:

- Aumentar al usuario: El usuario lleva un dispositivo, normalmente en las manos o en la cabeza, para obtener información sobre los objetos físicos. Ejemplos de esto pueden ser dispositivos del estilo palms que muestren información o también cascos de realidad virtual. [Azuma et al., 2001].
- Aumentar los objetos reales: En este caso los objetos reales cambian según lo que esté delante de ellos. Generalmente implica agregar sistemas de sensores, y un ejemplo son los robots que reaccionan acorde a la información que detecta sus sensores [Estrin et al., 2002].
- Aumentar el ambiente que rodea al usuario y a los objetos: Ni el usuario, ni el objeto son directamente afectados. En cambio, los dispositivos proporcionan y coleccionan información del ambiente circundante, desplegando la información hacia los objetos y capturando la información sobre las interacciones del usuario con ellos. Ejemplo de estos son cámaras que capturan las actividades que realiza una persona y dependiendo de ello generan cambios en algún dispositivo [Mackay, 1998].

Al diseñar las aplicaciones de realidad aumentada, es importante considerar cómo hacer la integración de lo real y lo virtual de la manera más clara posible. Cada aplicación debe elegir la mejor combinación de técnicas para descubrir la información del mundo real y presentarla al usuario. Para lograr una definición uniforme, independientemente de la tecnología que se utilice, en Azuma et al. (2001) se especifican tres características que deben cumplir los sistemas para ser considerados aplicaciones de esta naturaleza:

- Combinar objetos reales y virtuales en un ambiente real.
- Existir interacción en tiempo real.
- Alinear objetos reales y virtuales entre ellos. Es decir, contener objetos sintéticos como por ejemplo: texto, objetos 3D, etc. alineados en forma adecuada a la información del ambiente real.

El concepto de realidad aumentada es aplicado para realizar aplicaciones destinadas a distintas disciplinas, sin embargo para esta tesis nos concentraremos en aquellas aplicaciones que se focalicen en brindar asistencia al usuario cuando recorre el mundo real, como así también aquellas que brindan información al usuario acorde a la imagen que está visualizando.

2.3.1. Aplicaciones de Realidad Aumentada para dispositivos móviles

A continuación se describen distintos proyectos de Realidad Aumentada, donde cada uno utiliza distintas tecnologías para llevar a cabo cada aplicación. Los proyectos que se presentan están orientados a guiar al usuario para moverse en el mundo real. Además algunas de estas aplicaciones muestran información superpuesta a la imagen de la realidad que está visualizando el usuario en un determinado momento.

En [Liarokapis et al., 2006a], [Liarokapis et al., 2006b], [Liarokapis and Mountain, 2007a] y [Mountain and Liarokapis, 2007b] los autores presentan una interfaz de realidad mixta para usuarios móviles que navegan por distintos espacios. La investigación realizada se enmarca dentro del proyecto LOCUS (Location-context Tools for UMTS Mobile Information Services). La interfaz de realidad mixta permite superponer información tanto de mapas 3D, imágenes 2D, sonido 3D e información textual. Esta interfaz fue diseñada para proveer información navegacional en tiempo real acorde a la posición del usuario dentro del ambiente. Si bien esta investigación abarca realidad mixta, el foco del análisis sólo se hace en base a la interfaz de Realidad Aumentada que proveen.

Los autores presentan una herramienta denominada "*Routing*", que provee a los usuarios una asistencia avanzada en la navegación. Para la misma, se basan en la experiencia de usuarios previos y sugieren diferentes caminos dependiendo de cada recorrido elegido. Permiten que el usuario seleccione el lugar donde quiere ir y luego le van brindando asistencia a través de la información del camino. La información que captura el dispositivo móvil con la cámara, permite mostrar dicha imagen con la información u objetos virtuales superpuestos en tiempo real.

La Figura 2.9 muestra cómo el usuario visualiza la imagen real que está viendo y cómo después esa misma imagen es aumentada con información que lo guía indicando el camino que debe seguir.



Figura 2.9: Realidad Aumentada Wayfinding. Imagen antes y después de la detección [Liarokapis et al., 2006b].

La mayor diferencia con otras interfaces es que combinan cuatro tipos diferentes de información navegacional: mapas 3D, mapas 2D, sonido y texto. Además, tienen dos modos de registración, uno basado en marcas de referencias (fiducial markers) y otro en detección de características naturales. La principal ventaja de esta última registración es que pueden operar en ambientes desconocidos. Las marcas de referencias son puestas en puntos de interés (points-of-interest – POIs) en el ambiente, tales como un edificio, el final de una calle, etc. Estos puntos juegan un rol significativo en el proceso de toma de decisiones.

Los autores adaptaron el algoritmo de [Kato and Billinghurst, 1999] de matching de *ARToolKit*¹⁷ para detectar las marcas y lo están extendiendo para detectar características naturales (por ahora detectan cuadrados, rectángulos, rombos y trapecios). Estos dos modos pueden apreciarse en la Figura 2.10, por un lado la Figura 2.10.a muestra la información usando marcas de referencias y la Figura 2.10.b detecta características naturales y a partir de estas muestra la información adecuada (en este caso las flechas).

¹⁷ Página de *ARToolkit*: <http://www.hitl.washington.edu/artoolkit/>

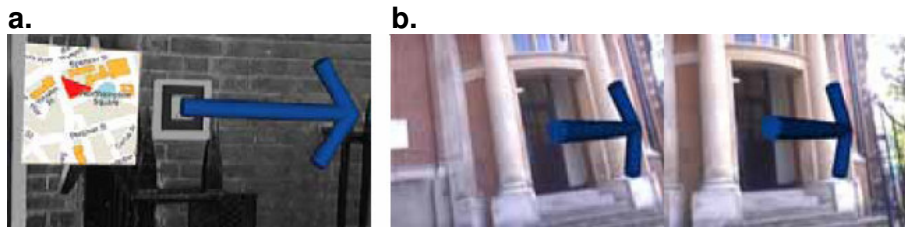


Figura 2.10: Navegación con Realidad Aumentada. a) Navegación usando marcas de referencias. b) Navegación usando la característica natural de la puerta [Liarokapis et al., 2006a].

Un trabajo que muestra como navegar usando Realidad Aumentada es presentado en [Kalkusch et al., 2002], [Reitmayr and Schmalstieg, 2003a], [Reitmayr and Schmalstieg, 2003b], [Reitmayr and Schmalstieg, 2004], [Schmalstieg and Reitmayr, 2005] y [Schmalstieg et al., 2007]. Los autores usan realidad aumentada para sistemas de navegación outdoor e indoor.

En [Reitmayr and Schmalstieg, 2003a] los autores usan como plataforma *Studierstube*¹⁸ para su software de Realidad Aumentada. Esta plataforma permite un prototipado rápido. Para el análisis de las imágenes recibidas de la cámara utilizan OpenTracker, el cual se basa en *ARToolkit*. Este software permite analizar las imágenes de video de la cámara (la cual se encuentra en el casco del usuario), y le agrega la información necesaria según se trate de la aplicación outdoor o indoor.

Para los sistemas indoor los autores implementan una aplicación llamada *Signpost* que permite guiar al usuario en la navegación para alcanzar alguna habitación en particular. La aplicación calcula el camino más corto desde la sala actual al destino y brinda la información necesaria para llegar de una sala a la otra. Para realizar la búsqueda del camino se utiliza un grafo de adyacencias. En la Figura 2.11 se puede apreciar como se reconocen marcas de referencias y se puede determinar en qué posición se encuentra el usuario, y cuál es el camino que debe seguir.

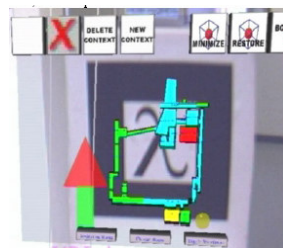


Figura 2.11: Marca para guiar al usuario en un edificio [Kalkusch et al, 2002].

Los autores tienen otra aplicación indoor que permite indicar donde están posicionados o ubicados distintos libros. Esto lo hacen mediante el reconocimiento de marcas de referencias que le indican el espacio de coordenadas para permitir remarcar la ubicación del libro que se busca. En la Figura 2.12 se puede apreciar las marcas de referencias y el recuadro del sector en que se debe buscar el libro.



Figura 2.12: Reconocimiento de marcas de referencias y marcado del sector donde buscar [Reitmayr and Schmalstieg, 2003b].

¹⁸ Página de *Studierstube*: <http://studierstube.icg.tu-graz.ac.at/>

En cuanto a las aplicaciones outdoor, en [Reitmayr and Schmalstieg, 2003b] los autores especifican un modelo de navegación basado en un grafo de nodos (estos nodos algunos coinciden con esquinas o intersecciones de caminos), los nodos son denominados waypoint. Cuando un usuario tiene que realizar un camino, se le marca el path con todos los waypoints.

En [Reitmayr and Schmalstieg, 2004] detallan el proceso de la siguiente manera: el usuario selecciona un lugar de destino, el cual puede ser por ejemplo: un supermercado. El sistema le calcula la ruta más corta dentro de una red conocida (a través de la información geográfica de la ciudad). El camino se recalcula continuamente mientras el usuario realiza su recorrido, de esta manera se mantiene la información actualizada por si el usuario se extravió o decidió tomar otra ruta. El usuario recibe una serie de waypoints que se le muestran mediante cilindros dentro del ambiente. Los cilindros se encuentran unidos por flechas que muestran la dirección del camino. El usuario sólo tiene que seguir los waypoints para alcanzar su destino. En el caso de no poder visualizar el próximo waypoint, se le muestra la línea del path con la dirección.

En la Figura 2.13 se puede apreciar información del path que tiene que realizar el usuario junto con los waypoint. Por un lado, en la Figura 2.13.a el usuario percibe el path superpuesto a la imagen y no hay tratamiento de profundidades y por otro el path real considerando las profundidades y los edificios puede observarse en la Figura 2.13.b. La Figura 2.13.c muestra cómo se calcula el path con los waypoint y luego este dibujo tridimensional es el que se superpone a la imagen real que ve el usuario. Para realizar la representación de los path y las profundidades, cuentan con una extensa base de datos con la altura de los edificios y la representación del sistema de calles.

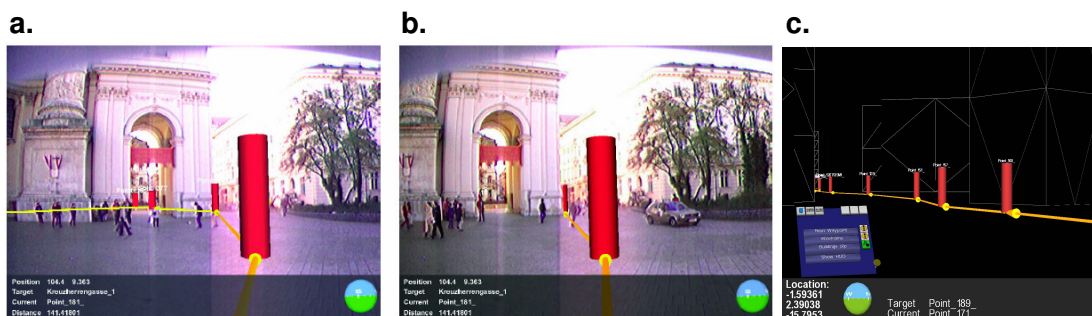


Figura 2.13: Path con waypoint. a) Camino indicando el path sobre los edificios. b) Camino indicando el path pero sólo las partes visibles. c) Información del path que se superpone a la imagen que está visualizado el usuario. [Reitmayr and Schmalstieg, 2004].

Cuando el usuario está frente a un edificio puede ver la geometría del mismo (para esto conocen la estructura de los edificios) y también puede apreciar iconos que indican que hay más información de ese lugar en particular. Cuando el usuario selecciona estos iconos, que se encuentran marcados sobre el edificio, recibe más información, esta situación puede visualizarse en la Figura 2.14 donde el usuario recibe tanto información textual como imágenes de un lugar particular del edificio.

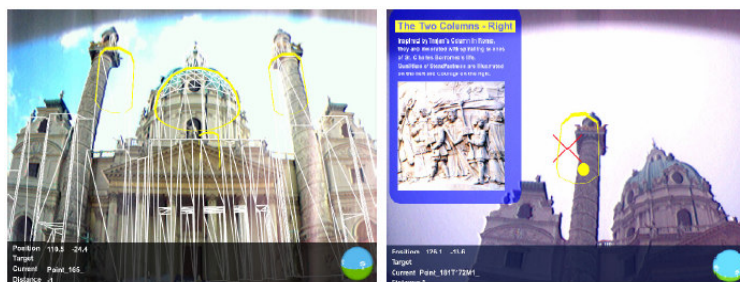


Figura 2.14: Información textual e imágenes de lugares puntuales del edificio [Reitmayr and Schmalstieg, 2004].

Un trabajo similar al presentado por [Kalkusch et al, 2002] para búsquedas de una habitación particular, es mostrado en [Kolsch et al., 2006]. Donde se modifica el algoritmo de Dijkstra para buscar el path más corto y poder mostrar resultados similares a los presentados por [Kalkusch et al, 2002]. El usuario puede generar un camino a un objeto elegido o al centro de un edificio usando la técnica de túnel (puede ver el edificio junto con su estructura interna y elegir ahí el destino). El usuario activa esta función mediante el comando de voz “*set path from here to tunnel*”. La Figura 2.15 muestra el camino que ve el usuario a medida que va ingresando en el edificio. Los autores en [Kolsch et al., 2006] sólo hacen una breve referencia del tema del path ya que el foco principal del paper es el tema del túnel.



Figura 2.15: Path mediante la función de túnel [Kolsch et al., 2006].

2.3.2. Aplicaciones de Hipermedia Física

Dentro del espectro de aplicaciones móviles que se están desarrollando, existe un conjunto en particular, que se caracteriza por combinar información digital y física, es decir, del mundo real. El ejemplo de la introducción es un caso característico de este tipo de aplicaciones. Estas nuevas aplicaciones, desafían a los sistemas de Hipermedia tradicionales y a otros tales como los sistemas de “*open hypermedia*”, “*spatial hypermedia*”, y la Web, en donde la información relevante es puramente digital.

Estas aplicaciones (de Hipermedia Física) se caracterizan por proveer navegación digital y navegación física (el usuario moviéndose por el mundo real). Desde este punto de vista se pueden considerar aplicaciones de hipermedia a las cuales se les incorporan facilidades para la definición y manejo tanto de objetos físicos (objetos que aparecen en el mundo real) como de la navegación física (recorridos que realiza el usuario por el sitio en el cual se encuentra); esto se denomina en [Hansen et al., 2004] como “*physical hypermedia*” (Hipermedia Física).

Las aplicaciones de Hipermedia Física son una forma de aumentar la realidad, agregando información de Hipermedia relacionada a los objetos del mundo real. Este tipo de aplicaciones no son de Realidad Aumentada pura ya que, por un lado, no necesariamente cuentan con las tres características que se definen para las aplicaciones de esta naturaleza (combinar objetos reales y virtuales en un ambiente real, existir interacción en tiempo real y alinear objetos reales y virtuales entre ellos) y por otra parte, la combinación de información real con virtual no siempre se da como en Realidad Aumentada, ya que en Hipermedia Física el usuario recibe información de lo que está viendo pero no necesariamente ve la imagen real (o video) en su dispositivo. Las aplicaciones de Hipermedia Física tampoco muestran siempre objetos virtuales superpuestos con la realidad.

Por último, las aplicaciones de Hipermedia Física cuentan, con una característica particular que no poseen las aplicaciones de realidad aumentada, y es la posibilidad de navegación entre la información que se brinda. Es decir, en Hipermedia se brindan los links navegables que proveen otro concepto que no es considerado en la mayoría

de las aplicaciones de realidad aumentada.

A continuación se detallan distintos proyectos o aplicaciones de Hipermedia Física.

Topos se desarrolló en el proyecto de *WorkSPACE*¹⁹ [Gronbaek et al., 2003]. Su objetivo fue desarrollar un sistema de Hipermedia Física para dar a los usuarios, un ambiente familiar al mundo real en el que trabajan. Organizando las mezclas entre materiales digitales y físicos, mediante colecciones de objetos.

El concepto central en *Topos* es el *workspace* (un ejemplo puede visualizarse en la Figura 2.16) que es el medio principal para agrupar y organizar los materiales y objetos en un espacio 3D.

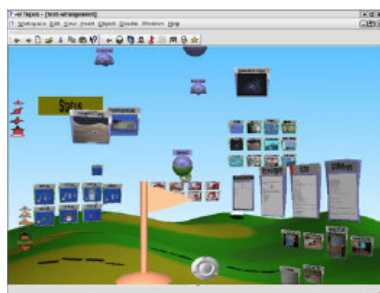


Figura 2.16: WorkSPACE con documentos organizados en varios grupos [Gronbaek et al., 2003].

Topos registra la posición del usuario (mediante *GPS*) para que el mismo pueda etiquetar elementos asociados al lugar actual donde se encuentra. El usuario tiene la posibilidad de realizar notas o tomar fotos relacionadas al lugar donde se encuentra. Además como *Topos* se basa en el trabajo colaborativo, cuando otro usuario está ubicado en ese lugar puede ver tanto los elementos etiquetados como las notas y fotos realizadas por otros usuarios superpuesto a la imagen del mundo real. Esto puede visualizarse en la Figura 2.17. *Topos* también provee la posibilidad de acceder a esa información en forma remota.

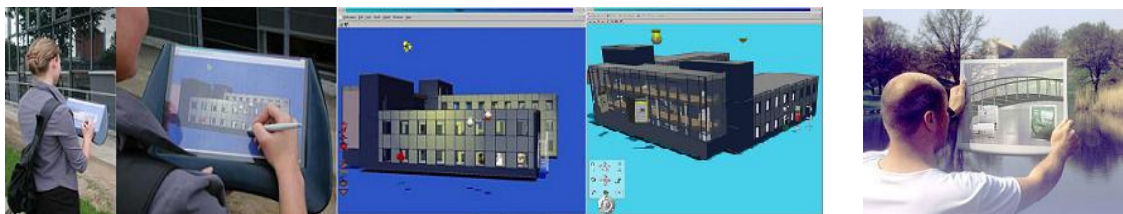


Figura 2.17: Ejemplo del Proyecto Topos de aumentar la realidad con información digital en proyectos arquitectónicos [Bouvin, 2004].

En [Gronbaek et al., 2003] se define Hipermedia Física (*Physical Hypermedia*) como un formalismo para construir aplicaciones de realidad aumentada. Los autores muestran cómo organizar ambientes mixtos entre materiales digitales y físicos, para esto etiquetan (por ejemplo mediante el uso de un código de barra) elementos reales importantes y luego les asocian información digital. De esta manera aumentan los objetos reales con información.

Los autores refinan el trabajo anteriormente mencionado y presentan en [Bouvin et al., 2003] y [Hansen et al., 2004] un framework denominado *HyCon (A framework for Context-Aware Mobile Hypermedia)*, el cual trata de extender Hipermedia con el mundo físico. Dicho framework da soporte a sistemas de Hipermedia context-aware, permite realizar anotaciones, manejo de links y tours guiados asociándolos a ubicaciones. Además brinda los mecanismos clásicos de Hipermedia para navegar,

¹⁹ Página de *workSPACE*: <http://daimi.au.dk/workspace/index.htm>

buscar, hacer anotaciones y tours guiados en el mundo físico, permitiendo a los usuarios realmente unir objetos digitales y/o físicos. A partir de este trabajo, los autores empiezan a manejar el término “*Context Aware Mobile Hypermedia*” para representar aquellos sistemas de Hipermedia Física móviles sensibles al contexto. Los autores van más allá de simplemente aumentar los objetos reales, además proveen anotaciones, links, colecciones o tours guiados dependientes del contexto.

En estos últimos trabajos se diferencian dos tipos de objetos: los digitales y los físicos. Los objetos digitales se definen y utilizan como en las aplicaciones de Hipermedia convencionales mientras que los objetos físicos son aquellos objetos del mundo real en los cuales el usuario se posiciona, y que mediante un mecanismo de código de barra le provee la información digital. La relación entre los objetos digitales y físicos se establece cuando el usuario se posiciona frente a un objeto físico, el sistema es capaz de detectarlo (por ejemplo: mediante la lectura de un código de barra) y le provee la información Hipermedial.

La Figura 2.18 muestra una manera de navegar y buscar información del prototipo *HyCon* [Hansen et al., 2004]. Para ello se ingresan ciertos términos, y el sistema captura la información de contexto. La ubicación actual del usuario se muestra con un punto rojo, las anotaciones con círculos verdes y el lugar elegido para visitar se encuentra remarcado. También se puede observar información digital de la misma y ciertos tours recomendados en forma de flechas.

En la Figura 2.18.a se puede observar como se puede realizar una búsqueda (usando *HyCon*) teniendo en cuenta la posición actual del usuario. En la Figura 2.18.b. se muestra un camino establecido donde los usuarios pueden hacer anotaciones relacionadas a cada punto de interés de dicho camino.

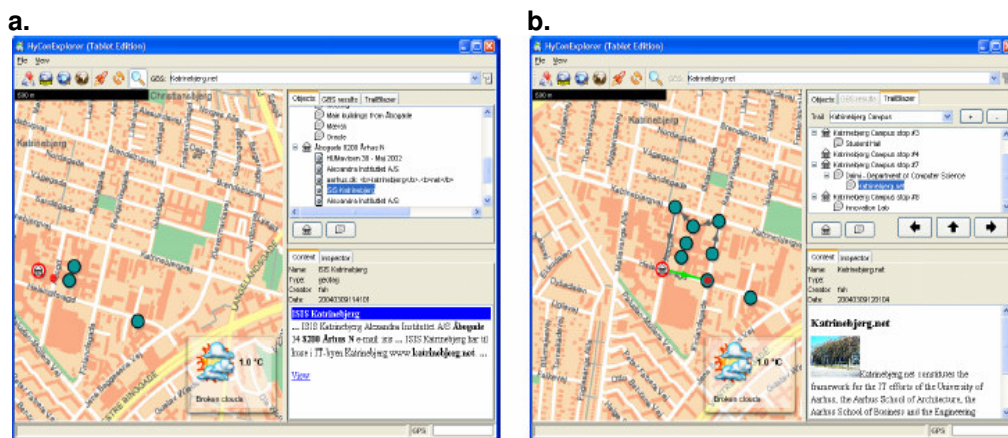


Figura 2.18: Ejemplo de navegación y búsqueda en el prototipo *HyConExplorer* [Hansen et al., 2004].

Los autores utilizan su framework *HyCon* y lo extienden para el proyecto *UrbanWeb*²⁰ (*A Mobile Social Context-Aware Web Infrastructure*). Usando este nuevo framework presentan en [Hansen et al., 2008] el concepto de “*Mobile Urban Drama*”, es un sistema basado en localización que convierte al usuario en el actor principal de una obra y en los distintos lugares éste escucha videos de distintos actores los cuales se relacionan a lugares físicos dentro de la ciudad (la cual actúa como escenario).

En la Figura 2.19 se puede visualiza un usuario usando la aplicación *AudioMove* Hansen et al., 2008], esta aplicación fue desarrollada por los autores en el marco del proyecto *UrbanWeb*. La figura muestra cómo el usuario usa esta aplicación, el mismo va recorriendo la ciudad y va leyendo los distintos códigos de barra que están ubicados en distintos lugares de la misma, como resultado recibe un audio con parte

²⁰ Página de *UrbanWeb*: <http://www.daimi.au.dk/~fah/urbanweb/>

de una historia. Cada audio guía al usuario para que pueda llegar al próximo código de barra y así avanzar en la historia.



Figura 2.19: Ejemplo de Urban Drama.

La extensión del framework *HyCon* también es usada por los autores para proveer aplicaciones educativas como se presentan en [Hansen and Bouvin, 2009]. La aplicación se llama *HasleInteractive*, la cual combina una historia con aspectos de educación móvil. Los alumnos van recorriendo los distintos checkpoints. Cuando los alumnos llegan a cada checkpoint leen, mediante la cámara del celular, los códigos de barras y reciben información específica que deben de completar, según lo que observan del ambiente que los rodea. También hay checkpoints extras que sólo proveen información y audio relacionado, para proporcionar a los alumnos más detalles del misterio que deben resolver y hacer más interesante la recolección de datos. Los alumnos tienen la posibilidad de sacar fotos del lugar y almacenarlas. Luego el docente puede utilizar toda la información recolectada por los alumnos para analizarla en clase.

La conexión que existe entre el objeto digital y físico en [Hansen et al., 2008] y [Hansen and Bouvin, 2009] los autores la denominan “digital-physical links”, este link se establece entre el código de barra y la correspondiente información Hipermedial. Otros autores en [Schwieren and Vossen, 2007] denominan a está relación (link entre el código de barra y la información Hipermedial) como “Physical Hyperlink”.

El proyecto *proXimity* es un sistema que busca aumentar la realidad dando al Hipertexto, y así a la Web, una presencia física en el mundo verdadero. Está basado en trabajos sobre Hipermedia y movilidad en el mundo real e intenta extender la metáfora del link de Hipermedia en el mundo real. Por ejemplo una persona que se encuentra de paseo por una ciudad que no conoce, puede ir obteniendo información en su dispositivo móvil que lo ayude a orientarse e informarse. Aquí surge el concepto de “*walk the Links*” [Harper et al., 2004] o “*caminar los links*”. El principal aporte de *proXimity* es establecer tres componentes para el Hipertexto en la Web: un componente espacial, uno temporal y otro semántico. El componente semántico permite agrupar los links bajo algún criterio. El temporal da la información de la ubicación tanto de la persona como de los objetos del mundo real, y el espacial resuelve las cuestiones de como atravesar las distancias. Este trabajo establece fuertes indicios para suponer que en el futuro, el Hipertexto y el mundo real van a unirse y permiten que el usuario en efecto “*camine los links*”.

El concepto de “*caminar un link*” es representado también por otros autores. En [Millard et al., 2002] en el marco del modelado realizado usando Fundamental Open Hypermedia Model (*FOHM*) los autores especifican la estructura para lo que denominan “*Real-World Link*”. Está especificación representa un link cuyo target y source son objetos del mundo real y en donde se asume que el usuario sigue estos links caminando. En [Millard et al., 2002] se nombra a los links con estas características como “*Traversal Link*”, indicando que son aquellos links donde el usuario tiene que moverse de su ubicación para encontrar el target deseado.

2.4. Resumen

En la Sección 2.1 se presentó una introducción a los conceptos básicos relacionados con el desarrollo dirigido por modelos. Se hizo hincapié en los distintos niveles de modelado (metametamodelo, metamodelo, modelo y instancia del modelo) que se dan en este tipo de desarrollo y como se relacionaban estos niveles.

Además, se mencionó en este capítulo los tres principios básicos relacionados al desarrollo dirigido por modelo como son la representación directa (mediante el uso de lenguajes específicos de dominio), la automatización (transformación entre modelos y generación de código de manera automática) y uso de estándares.

Se analizó en particular un estándar que implementa el desarrollo dirigido por modelo como es *MDA*. Este estándar clasifica los modelos de su ciclo de vida como modelos de requerimientos (*Computation-Independent Model - CIM*), modelos de análisis y diseño (*Platform-Independent Model - PIM*) y modelos de código (*Platform-Specific Model - PSM*). Esta clasificación no solamente es usada por *MDA* sino que además es adoptada por otros, desarrollo dirigidos por modelos. También se mencionó las distintas transformaciones relacionadas con *MDA* como son: *CIM* → *CIM*, *PIM* → *PIM*, *PSM* → *PSM*, *CIM* → *PIM*, *PIM* → *CIM*, *PSM* → *PIM*, *PSM* → *PSM*.

En este capítulo se describió el desarrollo dirigido por modelos en particular para las aplicaciones Web, mencionando tres metodologías como son *OOHDM*, *WebML* y *UWE*. Donde *UWE* se destaca como la metodología que respeta mejor las bases del desarrollo dirigido por modelos, ya que se basa en estándares como son *MOF* y *UML* para la construcción de sus modelos y provee herramientas para la generación automática de aplicaciones Web.

Los conceptos presentados en este capítulo sobre el desarrollo dirigido por modelos son la base para la especificación de nuestro enfoque.

En la Sección 2.2 se presentaron distintos trabajos que enuncian los beneficios de la separación de concerns, como es, por ejemplo, la reducción de la complejidad de una aplicación ya que cada concern se focaliza en solucionar algún tema en particular. Trabajar con separación permite la reusabilidad y simplificar las problemáticas generadas por la evolución del software, ya que cada concern evoluciona de manera independiente. Es fundamental contar con algún mecanismo que componga los distintos concern para lograr una función integrada. El éxito de la separación de concerns está dado por una adecuada elección de los concerns de interés para una aplicación particular.

Además, en la Sección 2.2 se describió como las metodologías *OOHDM* y *UWE* usan separación de concerns. Se mencionó que *OOHDM* usa como parte de la metodología dos niveles de separación de concerns: verticales y horizontales. En tanto, *UWE* cuenta solamente con separación de concerns horizontales, lo cuales están dados por lo modelos que integran su desarrollo dirigido por modelos.

En la Sección 2.3 se analizaron las aplicaciones de realidad aumentada, introduciendo los conceptos básicos y describiendo las tres características específicas que deben cumplir este tipo de aplicaciones como son: combinar objetos reales y virtuales en un ambiente real, tener interacción en tiempo real y contener objetos sintéticos como por ejemplo: texto, objetos 3D, etc. alineados en forma adecuada a la información del ambiente real. Se detallaron algunos ejemplos de aplicaciones de realidad aumentada específicas para dispositivos móviles. Las aplicaciones mostradas están orientadas a guiar al usuario para moverse en el mundo real, alguna de ellas muestra información superpuesta a la imagen de la realidad que está visualizando el usuario en un determinado momento.

Se describieron, además en la Sección 2.3, las aplicaciones de Hipermedia Física (aumenta los objetos del mundo real con información Hipermedial), las cuales pueden no contar con las tres características descritas para las aplicaciones de realidad

aumentada, ya que la combinación de información real con virtual no siempre se da. La Sección 2.3 es la base para luego introducir en el Capítulo 3 los conceptos relacionados con las aplicaciones de Hipermedia Móvil, las cuales son caso particular de aplicación de Hipermedia Física donde se considera la movilidad del usuario y el contexto del mismo.

3. MODELADO DE APLICACIONES DE HIPERMEDIA MÓVIL

En este capítulo se presentan los conceptos básicos relacionados a Hipermedia Móvil (de ahora en adelante HM), mostrando mediante ejemplos cada una de las características propias de este tipo de aplicaciones.

Además, se describe cómo se utiliza la separación de concerns, en particular, en aplicaciones de HM. Se muestra como al usar el concepto separación de concerns en este tipo de aplicaciones se pueden generar links caminables derivados, a partir de la información que está viendo el usuario en un momento dado.

Se describe también, de manera conceptual, el enfoque definido para modelar aplicaciones de HM, mostrando los diferentes modelos que se definen y sus principales características. En el caso del Modelo de Contenido, se detallan las ventajas de la separación de concern como herramienta para el desarrollo de las mismas.

3.1. Aplicaciones de Hipermedia Móvil: el mundo digital y el real

El enfoque de modelado que se presenta en esta tesis está pensado para permitir la especificación de aplicaciones de HM. Nosotros usamos el término HM para describir aquellas aplicaciones de Hipermedia que pueden explorar tanto el mundo real como digital mediante algún dispositivo móvil. Las aplicaciones de HM pueden considerarse un caso particular de aplicación móvil, ya que las primeras tienen la particularidad de contar con los conceptos de link y navegación como servicios centrales para el usuario, mientras que las últimas no necesariamente presentan estas características.

En esta sección se explican las funcionalidades que brinda una aplicación de este tipo en términos de los objetos digitales y los objetos físicos. Las aplicaciones de HM tienen tanto *objetos digitales* (que se comportan de la misma manera que en la Hipermedia tradicional) como *objetos físicos* que representan objetos del mundo real relevantes para el usuario, en términos del lugar en donde se encuentra; de estos objetos se conoce, entre otra información, su ubicación física (posición).

En nuestras aplicaciones, los objetos digitales están relacionados mediante *links digitales*, que definen navegación en el sentido tradicional de Hipermedia, mientras que los objetos físicos están relacionados mediante los *links caminables* (*walking link*) que definen recorridos posibles para el usuario. En la Figura 3.1 se puede apreciar como representamos estos conceptos de manera esquemática (sector izquierdo de la figura) y cómo son visualizados por parte del usuario (sector derecha de la figura).

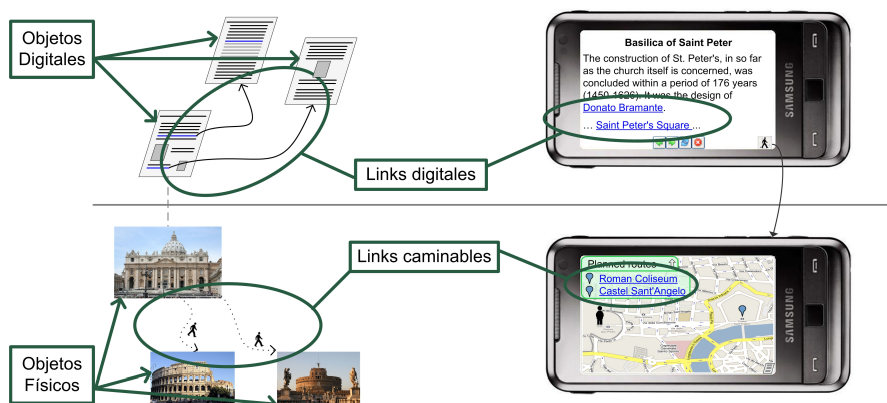


Figura 3.1: Objetos (digitales y físicos) y sus links (digitales y caminables).

Cabe destacar que cada link digital está asociado a un único objeto digital, y permite la navegación a otro objeto digital, al cual llamaremos target. Lo mismo ocurre con el link

caminable, pertenece a un único objeto físico y permite establecer una relación con otro objeto físico (target). Esta característica permite que cada vez que el usuario es sensado por un objeto físico, se despliegue, además de la información de ese objeto, los links caminables que tiene asociados.

Si el usuario selecciona un link digital, el sistema le brinda información digital del objeto target. Vamos a referirnos a la acción de seleccionar un link digital y recibir la información del mismo como *navegación digital*.

Cuando el usuario selecciona un link caminable, indica la intención de caminar hacia el target, en este momento se considera que se empieza la *navegación en el mundo real (Real-World Navigation)* y ésta finaliza recién cuando el usuario es detectado por el destino elegido. Como resultado de seleccionar un link caminable, el sistema le brinda al usuario un camino para que pueda llegar al target (físico) deseado asistiéndolo en este trayecto. La diferencia principal entre la navegación digital y la navegación en el mundo real, es que la primera es atómica y la segunda no. La navegación en el mundo real demanda tiempo y depende del usuario, muchas veces además influyen las condiciones ambientales. En la navegación en el mundo real, es importante tener en cuenta los cambios de idea que puede tener el usuario (por ejemplo, cancelar un recorrido), si el usuario se pierde en el medio del camino, etc. También pueden influir las preferencias que cada usuario posee (que determinan el camino elegido, los objetos intermedios que atraviesa, etc.).

Respecto de Hipermedia, HM no solamente incorpora el concepto de navegación en el mundo real, sino que también tiene la particularidad de brindar información al usuario, cuando el mismo se encuentra parado frente a un objeto del mundo real determinado, es decir, hay información digital que se activa automáticamente sólo por el hecho de que el usuario fue detectado (o sensado) por el objeto. A nivel de sistema esto implica una navegación implícita a la información correspondiente al objeto. En este punto, dependiendo de la naturaleza de la aplicación deben analizarse, por ejemplo, cómo es el mecanismo de feedback para la notificación de la llegada del usuario a un nuevo objeto aumentado, qué información proveerle en ese momento, si se mantiene o no información de contexto sobre los lugares ya visitados, etc.

Un objeto físico puede estar relacionado a un objeto digital, en este caso éste el primero se denomina *contraparte física del objeto digital*; este mecanismo de asociación es el que permite que el objeto del mundo real “conozca” su información digital. También pueden darse los casos de objetos digitales sin contraparte física y objetos físicos sin contraparte digital. Los distintos tipos de objetos mencionados se pueden apreciar en la Figura 3.2. De ahora en adelante nos vamos a referir a aquellos objetos físicos que tienen contraparte digital como Puntos de Interés.

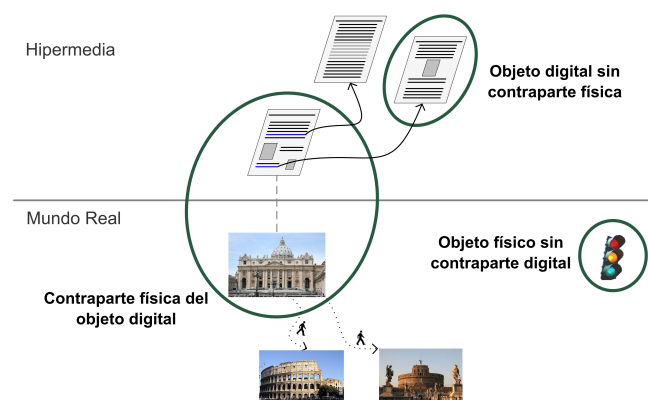


Figura 3.2: Objetos que aparecen en HM.

Veamos un ejemplo donde un usuario (desde su dispositivo móvil) usa una aplicación de HM. Supongamos (Figura 3.3) que el usuario está frente a la *Basílica de San Pedro*, como este objeto es un Punto de Interés, el usuario recibe información digital y física del mismo. El usuario puede cambiar de la vista con información digital (o Hipermedial) a la vista con información física (mapa) utilizando un botón del menú disponible (lo mismo sucede cuando está en la información física y quiere cambiar a la información digital).

El usuario puede navegar digitalmente mediante los links digitales, de la misma manera que lo hace en una aplicación de Hipermedia tradicional, produciendo cambios en la pantalla con información digital a medida que los diferentes objetos digitales se van mostrando. La navegación digital no altera la pantalla con información física.

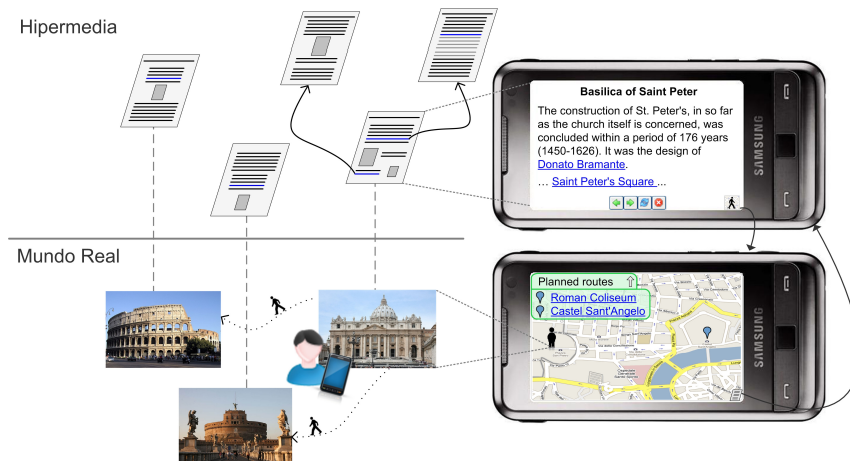


Figura 3.3: Información que recibe el usuario cuando es sensado por la *Basílica de San Pedro*.

En la pantalla con información física, a su vez, aparecen los links caminables. Si se selecciona un link caminable, el sistema muestra un mapa indicando el camino que debe realizar para llegar al target elegido.

En la Figura 3.4 se puede apreciar (suponiendo que el usuario eligió ir al *Coliseo*) que la vista con información física ahora tiene un camino establecido. Se puede notar en la Figura 3.4 que anteriormente el usuario había navegado digitalmente a la *Plaza de San Pedro*, cómo se visualiza en la vista con información digital.

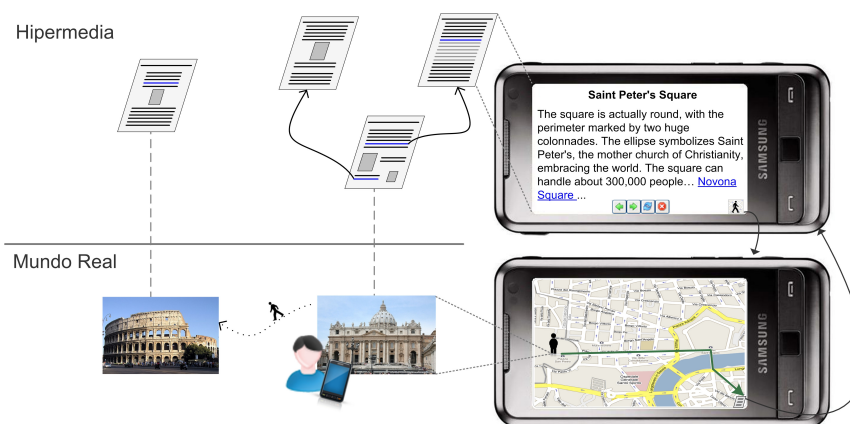


Figura 3.4: Inicio de la navegación en el mundo real.

Si mientras realiza el recorrido de un camino dado por el sistema (como resultado de haber seleccionado un link caminable), el usuario es detectado por un Punto de Interés (registrado en el sistema) puede ver información digital del lugar. Ésta situación se puede observar en la Figura 3.5 (el usuario es sensado por el *Panteón*). Esto lo habilita para navegar por los links digitales sin perder de vista que tiene pautado un

camino. Cabe aclarar que la navegación digital, mientras se está navegando en el mundo real, no afecta el camino pautado.

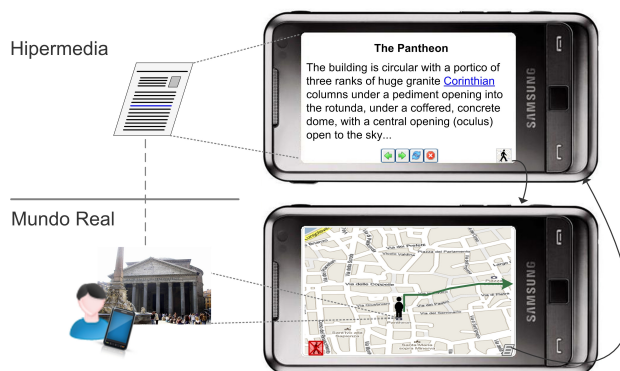


Figura 3.5: El usuario pasa frente a un nuevo objeto físico mientras navega en el mundo real.

Supongamos que el usuario camina siguiendo el recorrido pautado por el sistema, en algún momento es sensado por *el objeto target*. En este momento se considera que la navegación en el mundo real fue completada correctamente, entonces, se actualiza tanto la información digital como física con los links digitales y caminables correspondientes (y el usuario pasa a visualizar una pantalla similar a la Figura 3.3).

La principal diferencia, como se menciona anteriormente, entre la navegación digital y la navegación en el mundo real, es que la primera es atómica y la segunda no. En la navegación en el mundo real, hasta que el usuario no llegue físicamente al destino, esta navegación no se completa. Mientras el usuario realiza su camino puede perderse del camino pautado o puede cancelarlo. Es decir, en estos casos, se necesitan mecanismos que asistan al usuario para poder llegar al destino elegido.

El cálculo del camino está fuera del alcance de esta tesis. Sólo a modo informativo, se puede decir que el cálculo del camino puede ser simple como buscar el camino más corto o complejo como buscar un camino con la mayor cantidad de Puntos de Interés intermedios. Este cálculo se puede realizar usando, por ejemplo, la *API de Google Directions*²¹.

Manejar todos los aspectos referidos a la movilidad del usuario y la asistencia que éste necesita hace más complejo el modelado de estas aplicaciones, para las cuales no son suficientes las metodologías de modelado existentes para Hipermedia tradicional ya que las mismas no consideran los aspectos de movilidad como parte del modelado. Las metodologías de modelado de Hipermedia existentes, no soportan los conceptos de nodos físicos y su contraparte digital, por lo tanto tampoco es posible representar la navegación en el mundo real, ni el modelo del usuario necesario para poder representar la movilidad del mismo.

3.2. Utilización de Separación de Concerns en Hipermedia Móvil

Las aplicaciones de HM se caracterizan por manejar información de distinto tipo, por un lado la información propia del dominio y también información relacionada al mundo real, principalmente relacionada con posiciones de Puntos de Interés. La información propia del dominio, a su vez, puede contener información de distinta naturaleza como sucede en las aplicaciones de Hipermedia tradicional. Por ejemplo, en la aplicación

²¹ Página de *Google Directions API*:
<http://code.google.com/intl/es-ES/apis/maps/documentation/directions/>

turística presentada en la Sección 1.1 se puede tener como parte del dominio información turística, histórica, arquitectural, etc.

La información relacionada al mundo real abarca todo lo referente a la posición de los lugares que puede visitar el usuario y la información de los links caminables.

Se puede pensar que una aplicación de Hipermedia tradicional se puede convertir en una aplicación de HM si se le agrega la información relacionada al mundo real.

El concepto de separación de concern nos va a dar un marco para la modelización de las aplicaciones en el nivel conceptual. El enfoque propuesto se basa en la definición de concerns digitales y un concern físico. Los concerns digitales son propios del dominio de la aplicación que se este modelando, es decir, estos concerns van a surgir a partir de la naturaleza de la aplicación. Cada concern digital representa información sobre un aspecto particular del dominio la aplicación. Un objeto puede estar definido en más de un concerns, por ejemplo, un monumento puede estar definido en el concern histórico (especificando la información sobre su histórica) y en el arquitectural (detallando la información sobre su arquitectura).

El concern físico, va a representar información de los objetos físicos en el mundo real. En este concern cada objeto tiene definida (como información básica) su posición y los links caminables a otros objetos físicos alcanzables desde él.

Para el desarrollo de estas aplicaciones, al menos se debe definir un concern digital y el concern físico. Mientras que puede haber varios concerns digitales, el concern físico es único y representa todos los objetos físicos del sistema, sean Puntos de Interés o no.

En la Figura 3.6 se puede apreciar tres concerns asociados a la *Basilica de San Pedro*, dos digitales (histórico y arquitectural) y el concern físico. Cabe destacar que cada concern tiene información y links propios, los concerns digitales tienen links digitales mientras que el concern físico tiene links caminables.

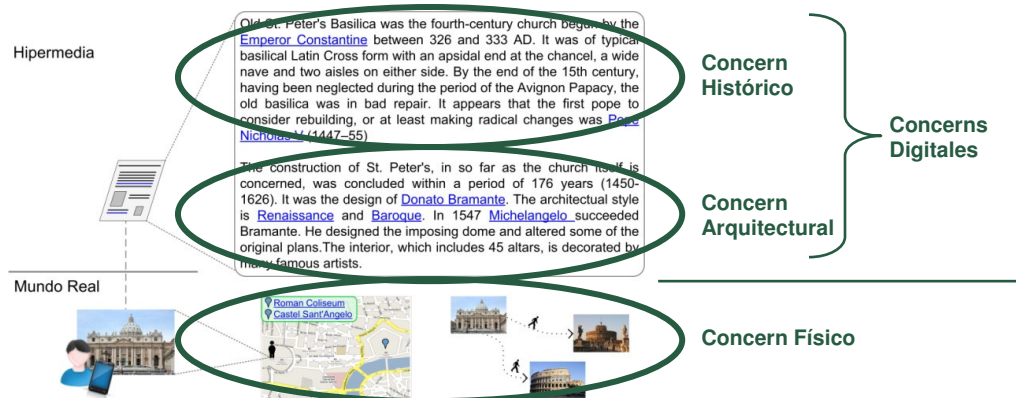


Figura 3.6: Diferente información asociada a la *Basilica de San Pedro*.

Desde el punto de vista del usuario, se puede pensar que cada concern digital representa una vista distinta del mismo Punto de Interés. El usuario puede cambiar de concern digital para recibir información diferente del mismo Punto de Interés. Cuando el usuario cambia de concern se le actualiza la información digital que está visualizando como así también los links digitales disponibles, que se ajustan al concern correspondiente.

Mientras el usuario no se mueve del lugar donde se encuentra parado la información relativa a su posición no cambia, ni tampoco los links caminables ya que estos pertenecen al objeto físico que lo senso.

Desde el punto de vista de la información que visualiza, el usuario, puede haber seleccionado un concern o, en el caso de no haberlo hecho, se le muestra la

información del concern definido como “Core”.

El usuario puede cambiar de concern en cualquier momento produciendo un cambio de vista automático que no afecta la vista con información física (Figura 3.7).

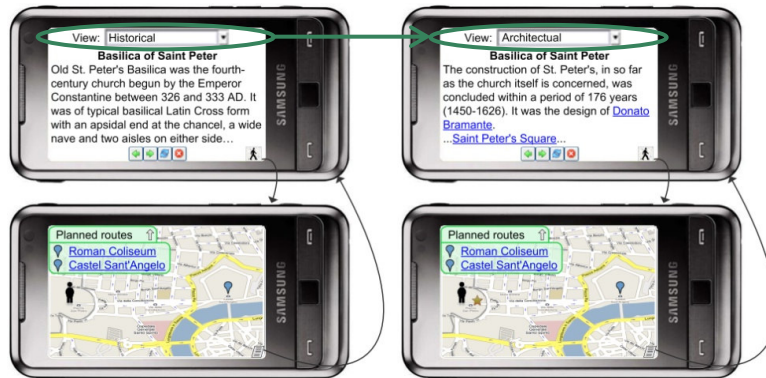


Figura 3.7: Visualización de los distintos concerns de la *Basílica de San Pedro*.

Cuando un usuario está viendo un determinado concern digital asociado a un Punto de Interés y selecciona un link digital, navega dentro del mismo concern. Es decir, el objeto target del link digital se visualiza en el mismo concern del objeto digital que contiene dicho link.

Cuando el usuario decide caminar a otro objeto físico, la información que visualiza al completar el recorrido, es la del concern digital en el que se encuentra. Por otro lado el usuario puede cambiar el concern digital en el que se encuentra, y, si mientras lo hace está realizando un recorrido, esto no afecta la navegación en el mundo real.

De la misma manera, cuando el usuario está caminando en el mundo real, si en el camino es sensado por un Punto de Interés intermedio, se le muestra información digital de este punto, en el concern digital en el que se encuentra.

En la Figura 3.8 se puede apreciar como el usuario en su camino, es sensado por el *Panteón*. En ese momento, recibe información digital del *Panteón* (en el concern que venía navegando, en este caso, el arquitectural), pudiendo navegar digitalmente (por los links digitales que provee el *Panteón*) o cambiar de concern, sin afectar el camino que tiene pautado.

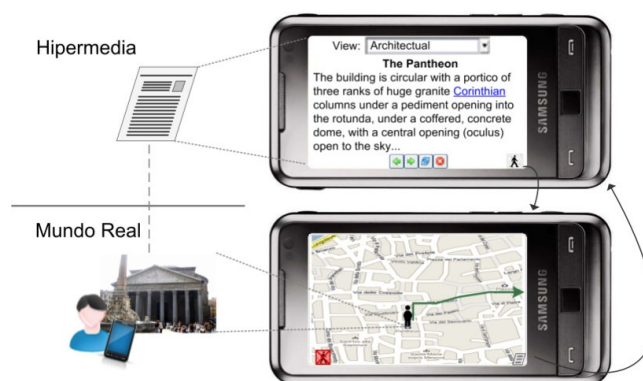


Figura 3.8: Punto de Interés intermedio a la navegación en el mundo real.

3.2.1. Generación de links caminables derivados

Hasta ahora vimos que las acciones realizadas en los concerns digitales no afectaban al concern físico y viceversa. Veamos cómo se puede enriquecer el nivel físico a partir del concern digital. Supongamos que el usuario está frente a un Punto de Interés y que

está navegando digitalmente por la información disponible. En la vista digital puede aparecer otro Punto de Interés que está relacionado con el actual (por ejemplo, porque corresponden al mismo período histórico). Al ser un Punto de Interés, es decir, que tiene contraparte física, la aplicación asume que el usuario puede querer visitar ese sitio, aún cuando no se haya definido un link caminable entre el Punto de Interés actual y el nuevo, entonces genera un link (caminable) derivable automáticamente. De esta forma, si el usuario cambia a la vista con información física, ese Punto de Interés le aparece como sugerencia entre las rutas posibles.

Estos links caminables derivados son temporales y se actualizan automáticamente cada vez que el usuario modifica la vista con información digital.

De esta manera se enriquece el concern físico con información relativa al concern digital que se está visualizando. Es decir, además de tener los links caminables relacionados a un objeto físico, tenemos links caminables derivados a partir de la información digital que está viendo el usuario en un determinado momento.

A partir de este momento llamaremos a los links que asocian objetos físicos como links caminables predefinidos y los links que se derivan a partir de la información de un concern digital como links caminables derivados. Los links caminables predefinidos son visualizados por el usuario cuando sea sensado por el Punto de Interés y son establecidos por el diseñador de la aplicación de HM. Los links caminables derivados son calculados dinámicamente mediante una función que analiza, por ejemplo, la información digital actual que está viendo el usuario en un momento dado.

La Figura 3.9 muestra cómo le aparecen los links caminables derivados, suponiendo que el usuario está viendo el concern (digital) arquitectural de la *Basílica de San Pedro*. Los links caminables predefinidos asociados al objeto físico que representa a la *Basílica de San Pedro* se indican en la pantalla como *Planned Routes*, mientras que los links caminables derivados se indican como *Suggestions*. Se puede apreciar, que dado que en el concern arquitectural aparece un link digital a la *Plaza de San Pedro*, y, como dicha plaza tiene contraparte física, se deriva un link caminable a ese lugar.

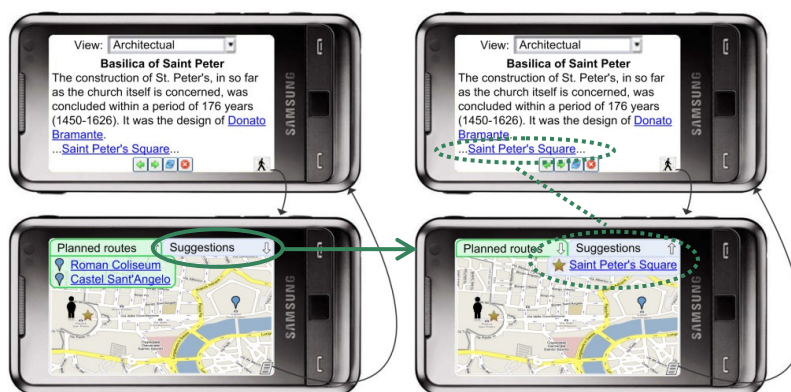


Figura 3.9: Links caminables predefinidos y links caminables derivados.

Los links caminables derivados se recalculan cada vez que el usuario cambia de concern digital, navega digitalmente o al ser sensado por un Punto de Interés. En la Figura 3.10 se muestra el resultado de haber navegado digitalmente desde la *Basílica de San Pedro* a la *Plaza de San Pedro* y cómo esta navegación digital afecta a los links caminables derivados, generando un nuevo link caminable derivado a la *Plaza Novona*.

Se puede apreciar que el link caminable derivado a la *Plaza de San Pedro* deja de ser una sugerencia para el usuario, ya que al cambiar la información digital que está visualizando, se actualizan los links caminables derivados.

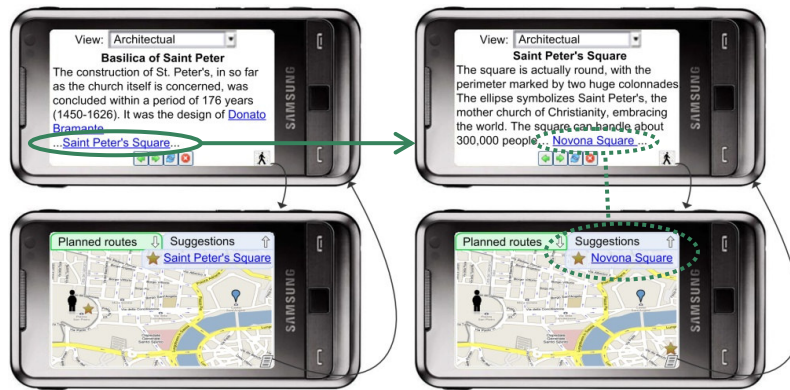


Figura 3.10: Actualización de los caminables derivados cuando el usuario navega digitalmente.

En esta sección se presentó en términos generales, el comportamiento que suponemos básico en una aplicación de HM. Esto significa, la posibilidad de que el usuario acceda a información tanto digital, como del espacio en el que se está moviendo. Desde el punto de vista de la información digital disponible, ésta está organizada para que el usuario pueda acceder a la misma por temas (concerns), y desde el punto de vista físico que pueda servirle de guía en su recorrido

Todas las características presentadas, deben de ser consideradas a la hora de diseñar este tipo de aplicaciones. Actualmente algunas metodologías existentes de Hipermedia consideran la separación de concerns como parte de sus modelos. Nuestro enfoque considera los conceptos de separación de concerns como parte del modelado, teniendo en cuenta tanto los concerns digitales, el concern físico y las relaciones entre ellos.

En la siguiente sección se presenta nuestro enfoque para modelar aplicaciones de HM y cómo tratamos la separación de concerns incorporando todas las cuestiones referidas al concern físico.

3.3. Enfoque para modelar aplicaciones de Hipermedia Móvil

En [Turlone et al., 2006] se presenta una metodología abstracta para desarrollar aplicaciones Web adaptativas. Esta metodología muestra el ciclo de vida abstracto que debe tener estas aplicaciones. Este ciclo de vida se pueden observar en la Figura 3.11, el cual empieza con una etapa de especificación de requerimientos, luego una serie de etapas iterativas que abarcan el diseño de datos, diseño navegacional y diseño arquitectural para luego plasmar todo a una implementación. Como últimos pasos del ciclo de vida se tienen las etapas de mantenimiento y el testeo para luego comenzar un nuevo inicio del ciclo logrando mejorar la aplicación.

Para la parte de especificación de requerimientos se realiza el análisis como en cualquier aplicación. Para las etapas de diseño de datos y diseño navegacional se debe usar alguna metodología que permita especificar de manera conceptual la aplicación. El diseño de la arquitectura, es un aspecto muy amplio que puede abarcar, de forma conceptual, la definición parcial o total de la estructura interna que permite establecer el funcionamiento (o parte del funcionamiento) de una aplicación. En cuanto a la implementación debe realizarse para algún lenguaje en particular. El mantenimiento y testeo permiten hacer una iteración en el ciclo permitiendo mejorar la aplicación.

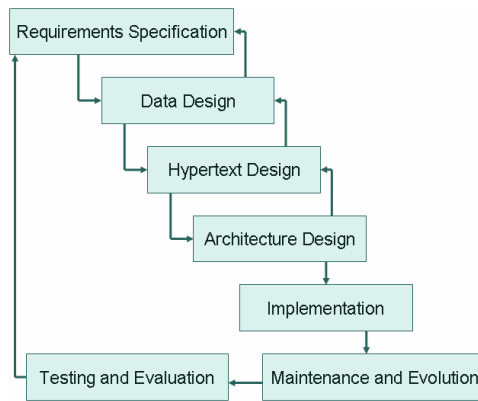


Figura 3.11: Etapas de la metodología presentada en [Torlone et al., 2006].

El ciclo de vida presentado en [Torlone et al., 2006] se ajusta adecuadamente a las características de las aplicaciones de HM que queremos representar, las cuales necesitan adaptación en diferentes aspectos como son: la ubicación del usuario, el perfil del usuario, el tipo de dispositivo (del usuario), etc. Por esta razón se toma como base para introducir el enfoque presentado en esta tesis.

En esta tesis no se profundiza en las etapas de requerimientos, mantenimiento y testeo, las cuales son tomadas como generales a cualquier aplicación. Es decir, nuestro enfoque se focaliza en las etapas de diseño de datos, diseño navegacional, diseño arquitectural e implementación. Como nuestro enfoque está basado en el desarrollo dirigido por modelos, para cada una de estas etapas (diseño de datos, diseño navegacional, diseño arquitectural e implementación) tenemos modelos específicos. Dentro del diseño de datos tenemos dos modelos asociados el de *Contenido* y el *Unificado*. En cuanto al diseño Hipermedial tenemos asociado el *Modelo Navegacional*. Mientras que asociado al diseño de la arquitectura tenemos el *Modelo de Presentación* y el *Modelo del Usuario*, estos modelos cubren de manera parcial la estructura interna de una aplicación (en futuras evoluciones de nuestro enfoque, se abarcará en forma total todas las cuestiones relacionadas a la estructura interna de una aplicación). La Figura 3.12 muestra para cada etapa de la metodología presentada en [Torlone et al., 2006] la relación que existe con los modelos de nuestro enfoque.

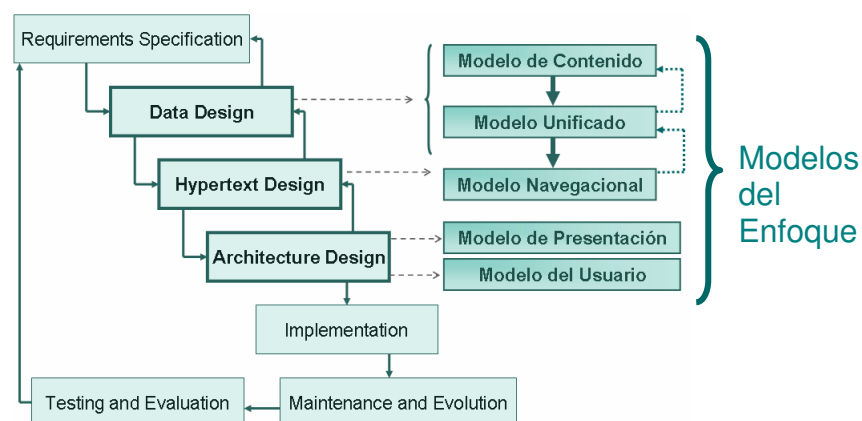


Figura 3.12: Relación entre la etapas propuestas por [Torlone et al., 2006] y los modelos de nuestro enfoque.

Además de los modelos mencionados, se hace hincapié en la necesidad de contar con transformaciones entre los mismos de manera de derivar el modelo siguiente a partir del anterior en forma semi-automática.

Se definió una transformación para derivar el *Modelo Unificado* a partir del de

Contenido y otra para derivar el *Unificado* al *Navegacional*. Se puede apreciar en la Figura 3.12 flechas punteadas entre estos modelos, que son para reflejar la posibilidad de hacer reingeniería, es decir reflejar cambios de un modelo en otro. En esta tesis sólo nos focalizaremos en mostrar las dos transformaciones mencionadas (del Modelo de Contenido al Modelo Unificado y del Modelo Unificado al Modelo Navegacional).

A continuación se describen las características de cada uno de nuestros modelos (Modelo de Contenido, Modelo Unificado, Modelo Navegacional, Modelo de Presentación y Modelo del Usuario) como así también cómo se realizan las transformaciones para pasar del Modelo de Contenido al Modelo Unificado y del Modelo Unificado al Modelo Navegacional.

3.3.1. Modelo de Contenido

El Modelo de Contenido define los conceptos relevantes del dominio de la aplicación y las relaciones entre estos conceptos mediante un diagrama de clases de *UML*. En este nivel, es donde se utiliza el concepto de *Separación de Concerns* para modelar diferentes aspectos de la aplicación. Esto nos permite reducir la complejidad del dominio y contar con las ventajas propias de este enfoque.

Las clases se modelan dentro de un concern determinado, la información que contiene cada concern es independiente de los demás. Es decir, cada concern esta auto-definido con sus propias clases y las relaciones entre las mismas.

En el Modelo de Contenido se van a diferenciar dos tipos de concerns, los concerns digitales (relacionados a los concerns de Hipermedia, es decir propios del dominio) y el concern físico (que representa la información del mundo real). El Modelo de Contenido de una aplicación de HM cuenta con tantos concerns digitales según el dominio lo requiera y además el concern físico. En una aplicación turística se puede modelar, por ejemplo, el concern turístico, arquitectónico, histórico y físico.

En el concern físico del Modelo de Contenido se especifican aquellas clases que se quieren representar físicamente. Las clases de este concern conocen su ubicación dentro del ambiente que se está representando (ubicación dentro de una ciudad, edificio, etc.). En este concern aparecen las clases sin ninguna relación ya que a lo sumo se tienen las relaciones geográficas implícitas de ubicación como contenido en o cerca de, etc.

Dado que los objetos físicos siempre tienen la misma ubicación, independientemente de la aplicación o el dominio que se esté representando, el concern físico puede ser compartido por diferentes aplicaciones.

Una clase de un determinado concern puede: tener su contrapartida en todos los demás concerns, tener su contrapartida en algunos concerns y en otros no o estar representada solamente en ese concern. Por ejemplo, un edificio de una ciudad puede estar representado tanto en el concerns arquitectónico, histórico como físico.

Nuestro Modelo de Contenido puede ser representado mediante un grafo²², ya que es un diagrama de clases de *UML*, y según la especificación [UML Specification], estos diagramas pueden ser vistos como grafos dirigidos [Bang-Jensen and Gutin, 2007], donde las clases son los vértices y las relaciones son las aristas.

El Modelo de Contenido se puede definir como un grafo integrado por sus clases y sus relaciones. Para simplificar la notación, las clases del grafo no se especifican detallando su estructura interna como es: nombre, atributos, métodos, etc. La

²² En [Bang-Jensen and Gutin, 2007] se puede encontrar más nivel de detalles de las estructuras de grafos y sus operaciones.

definición de las relaciones se establece en base a lo definido por [Ali et al., 2005], donde se menciona que el tipo de las relaciones pueden ser: asociación, agregación, composición, generalización o especialización. En la Definición 3.1 se puede apreciar la representación del Modelo de Contenido como un grafo.

```
ContentModel = (ClassesContentModel, RelationshipsContentModel)
ClassesContentModel = {Class1, ..., Classn}
RelationshipsContentModel = {<Classi, Type, Classj>:
    (Type ∈ {association, agregation, composition,
              generalisation, specialisation})
    and (Classi, Classj ∈ ClassesContentModel)}
```

Definición 3.1: Grafo que representa el Modelo de Contenido.

Si sólo se consideran en el grafo (del Modelo de Contenido) las clases y relaciones, se pierde la información respecto de los concerns definidos en el Modelo de Contenido y las clases que contiene en cada concern.

Los concerns se pueden definir como subgrafos [Bang-Jensen and Gutin, 2007] del Modelo de Contenido y cada uno de estos subgrafos contiene sus clases y relaciones. La Definición 3.2 especifica los subgrafos que representan los concerns del Modelo de Contenido. Se puede apreciar que el conjunto de todos los concerns (del Modelo de Contenido) se denomina $Concerns_{ContentModel}$, el cual contiene un conjunto de concerns digitales ($CDS_{ContentModel}$) y un concern físico ($CF_{ContentModel}$). Las clases y relaciones de cada concern (subgrafo) se definen como subconjuntos de $Classes_{ContentModel}$ y $Relationships_{ContentModel}$ respectivamente, respetando así la definición de subgrafo [Bang-Jensen and Gutin, 2007].

```
ConcernsContentModel = DigitalCsContentModel U {PhysicalCContentModel}
DigitalCsContentModel = {DigitalC1-ContentModel, ..., DigitalCn-ContentModel}
DigitalCi-ContentModel = (ClassesDCi-ContentModel, RelationshipsDCi-ContentModel)
    DigitalCi-ContentModel es un subgrafo de ContentModel, por lo tanto:
        ClassesDCi-ContentModel ⊆ ClassesContentModel y
        RelationshipsDCi-ContentModel ⊆ RelationshipsContentModel
PhysicalCContentModel = (ClassesPC-ContentModel, RelationshipsPC-ContentModel)
    PhysicalCContentModel es un subgrafo de ContentModel, por lo tanto:
        ClassesPC-ContentModel ⊆ ClassesContentModel y
        RelationshipsPC-ContentModel ⊆ RelationshipsContentModel
```

Definición 3.2: Subgrafos que representan los concerns del Modelo de Contenido.

Se puede apreciar en la Definición 3.2, que las clases $Classes_{CDi-ContentModel}$ y $Classes_{CF-ContentModel}$ se definen como un subconjunto de $Classes_{ContentModel}$. Esto no es suficiente si se quiere indicar que cada una de las clases pertenezca a un único concern. Para contemplar esta restricción se especificó la Definición 3.3.

```
ClassesDCi-ContentModel = {ClassDCi-ContentModel :
    (ClassDCi-ContentModel ∈ ClassesContentModel) and
    (ClassDCi-ContentModel ∉ ClassesDCj-ContentModel, i≠j) and
    (ClassDCi-ContentModel ∉ ClassesPC-ContentModel, i=1..n)}
ClassesPC-ContentModel = {ClassPC-ContentModel :
    (ClassPC-ContentModel ∈ ClassesContentModel) and
    (ClassPC-ContentModel ∉ ClassesDCi-ContentModel, i=1..n)}
```

Definición 3.3: Clases de los concerns digitales y físico.

De esta manera con las Definiciones 3.1, 3.2 y 3.3, se logro la especificación del grafo

que representa el Modelo de Contenido (con sus clases y relaciones), como así también la definición de los subgrafos que representan los concerns de dicho modelo.

Veamos un ejemplo particular del Modelo de Contenido, la Figura 3.13 muestra cómo representar (usando este modelo) una aplicación turística con tres concerns digitales representados (turístico, arquitectónico e histórico), y el concern físico.

Para detallar e introducir los modelos asociados a nuestro enfoque se eligió una notación simple²³ que permita la fácil comprensión de los conceptos. En el Capítulo 5 se muestra tanto la notación como los modelos concretos del enfoque.

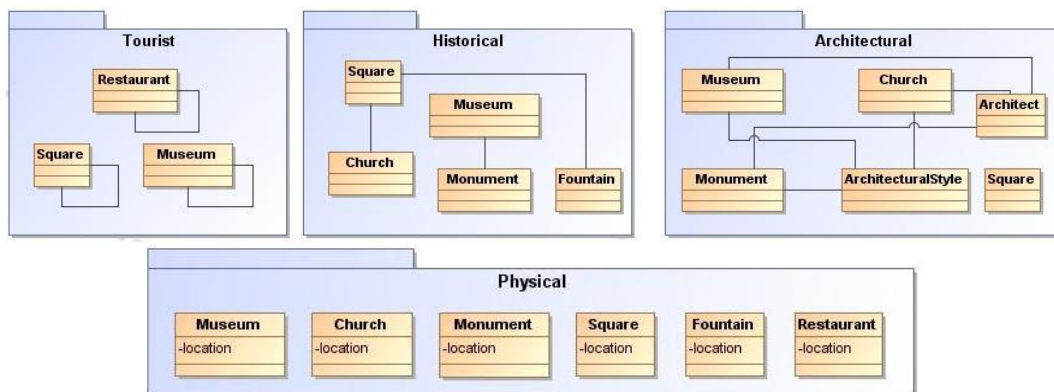


Figura 3.13: Esquema del Modelo de Contenido para HM.

Se puede ver en la Figura 3.13 como en algunos concerns las clases tienen correspondencias entre sí y en otros no, mientras que las relaciones de cada concern son independientes del resto de los concerns. Se puede observar que en el concern turístico hay una clase que no está representada en los otros concerns digitales como es *Restaurant*, esto se debe a que esta clase posee información sólo en este concern particular. Lo mismo ocurre para las clases *Architect* y *ArchitecturalStyle* del concern arquitectural que sólo tienen sentido en dicho concern. En el concern histórico pasa lo mismo con *Fountain*. Se puede apreciar que hay clases que aparecen en los tres concerns, en particular, *Square* y *Museum*. Mientras que hay otras que aparecen en algunos concerns como son *Church* y *Monument*, las cuales aparecen en el concern histórico y arquitectural.

En el concern físico las clases que se detallan tienen como propiedad fundamental la ubicación, indicada como `location`. También se puede apreciar que hay clases que no tienen representación en el concern físico como son *Architect* y *ArchitecturalStyle*. Es decir, estas clases no tienen una representación en el mundo real.

3.3.2. Modelo Unificado

En este modelo se relaciona la información de todos los concerns (del Modelo de Contenido), obteniendo un único modelo. Este modelo se deriva del Modelo de Contenido y la unificación se basa en la detección de las correspondencias entre las clases de los diferentes concerns. Para derivar la unificación se debe elegir uno de los concerns digitales como *Core*, según el dominio que se quiera representar. La información de los demás concerns aparece como roles [Cabot and Raventós, 2006] de las clases *Core*.

La composición de los concerns no se ve afectada según la elección del concern que actúa como *Core*, sino que se cambia el foco del concern principal de la aplicación.

Si una clase no está representada en el concern *Core* pero aparece en otros concerns,

²³ La notación simple usada en la Sección 3.3 se basa en la notación de OOHDM presentada en [Nanard et al., 2008] para separación de concerns en Hipermedia.

se decide cuál de los concerns en los que aparece la clase es el más conveniente para que actúe como *Core* de la misma y las demás clases son roles de la clase elegida. Nunca el *Core* es el concern físico, al excluir este concern como *Core* se asegura consistencia con la Hipermedia de base. Si se necesita usar la aplicación de HM como una aplicación de Hipermedia tradicional basta con sacar el concern físico. De otra manera hay que reestructurar todo el modelo.

En la unificación de los concerns a un modelo único no se deben agregar ni sacar información ya sean concerns, clases, atributos o relaciones que hayan sido definidas en el Modelo de Contenido. El Modelo Unificado sirve para logra relacionar las clases representadas en los distintos concerns, las cuales modelan la misma clase de objeto desde diferentes puntos de vista.

El Modelo Unificado también puede ser representado como un grafo, el cual agrega al Modelo de Contenido la información respecto de las relaciones de rol que se establecen entre las clases *Core* y sus roles. La representación del Modelo Unificado como grafo se puede apreciar en la Definición 3.4.

```

UnifiedModel = (ClassesUnifiedModel, RelationshipsUnifiedModel, RoleOfUnifiedModel)
ClassesUnifiedModel = {Class1-UnifiedModel, ..., Classn-UnifiedModel}
RelationshipsUnifiedModel = {<Classi, Type, Classj>:
    (Type ∈ {association, agregation, composition,
              generalisation, specialization})
    and (Classi, Classj ∈ ClassesUnifiedModel)}
RoleOfUnifiedModel = {<Class1, RoleOf, Classk>:
    (Class1, Classk ∈ ClassesUnifiedModel) and
    (Class1 ∈ Cj and Classk ∈ Ci, i≠j and (Ci, Cj ∈ ConcernsUnifiedModel))}

```

Definición 3.4: Grafo que representa el Modelo Unificado.

Se puede apreciar en la Definición 3.4, que la especificación de $RoleOf_{UnifiedModel}$ requiere de la definición de los concerns ($Concerns_{UnifiedModel}$). Esto se debe a que esta relación se establece entre clases de diferentes concerns. Los concerns del Modelo Unificado ($Concerns_{UnifiedModel}$) se pueden representar como subgrafos como se puede apreciar en la Definición 3.5. Estos subgrafos se representan de la misma manera que los concerns del Modelo de Contenido (Definiciones 3.2 y 3.3).

```

ConcernsUnifiedModel = DigitalCsUnifiedModel U {PhysicalCUnifiedModel}
DigitalCsUnifiedModel = {DigitalC1-UnifiedModel, ..., DigitalCn-UnifiedModel}
DigitalCi-UnifiedModel = (ClassesDCi-UnifiedModel, RelationshipsDCi-UnifiedModel)
    DigitalCi-UnifiedModel es un subgrafo de UnifiedModel, por lo tanto:
        ClassesDCi-UnifiedModel ⊆ ClassesUnifiedModel y
        RelationshipsDCi-UnifiedModel ⊆ RelationshipsUnifiedModel
PhysicalCUnifiedModel = (ClassesPC-UnifiedModel, RelationshipsPC-UnifiedModel)
    PhysicalCUnifiedModel es un subgrafo de UnifiedModel, por lo tanto:
        ClassesPC-UnifiedModel ⊆ ClassesUnifiedModel y
        RelationshipsPC-UnifiedModel ⊆ RelationshipsUnifiedModel
ClassesDCi-UnifiedModel = {ClassDCi-UnifiedModel :
    (ClassDCi-UnifiedModel ∈ ClassesUnifiedModel) and
    (ClassDCi-UnifiedModel ∉ ClassesDCj-UnifiedModel, i≠j) and
    (ClassDCi-UnifiedModel ∉ ClassesPC-UnifiedModel, i=1..n)}
ClassesPC-UnifiedModel = {ClassPC-UnifiedModel :
    (ClassPC-UnifiedModel ∈ ClassesContentModel) and
    (ClassPC-UnifiedModel ∉ ClassesDCi-UnifiedModel, i=1..n)}

```

Definición 3.5: Subgrafos que representan los concerns del Modelo Unificado.

Como se menciona anteriormente, el Modelo Unificado se deriva a partir del Modelo de Contenido. Al grafo que representa el Modelo de Contenido se le puede aplicar reglas de transformación (de grafos) para obtener así el grafo que representa al Modelo Unificado.

En [Varró et al., 2002] se describe una forma para definir reglas de transformación para grafos en general. En [Kuske et al., 2009] se amplían estas reglas de transformación mostrando cómo usarlas para grafos que representan diagramas de clases de *UML*.

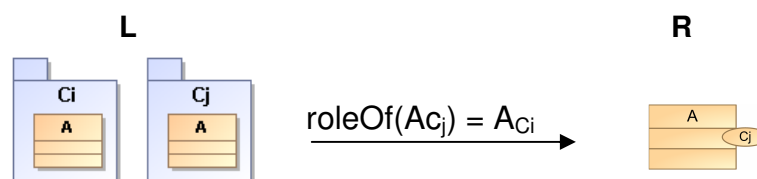
Usando como bases las ideas presentadas por [Varró et al., 2002] y [Kuske et al., 2009], a continuación, definiremos la regla de transformación necesaria para obtener el grafo que representa un Modelo Unificado a partir de un Modelo de Contenido.

Según [Varró et al., 2002] una regla de transformación se define con un grafo izquierdo (*L*), un grafo derecho (*R*) y eventualmente una condición necesaria que se debe cumplir para aplicar la regla. La regla (de transformación) se aplica a un grafo origen obteniendo un grafo resultante.

El grafo *L* sirve para identificar la estructura que se quiere reemplazar del grafo origen (en nuestro caso, el Modelo de Contenido), cada ocurrencia del grafo *L* se reemplaza por el grafo *R* obteniendo así el grafo resultante de la transformación (en nuestro caso, el Modelo Unificado). El reemplazo de las ocurrencias del grafo *L* por el grafo *R* sólo se realiza si se cumple, en el caso de estar especificada, la condición indicada en la regla.

La regla de transformación para generar el grafo del Modelo Unificado a partir del grafo del Modelo de Contenido ($\text{ContentModel} \Rightarrow_{R3.1} \text{UnifiedModel}$), sólo necesita definir cuando agregar las relaciones de rol entre las clases *Core* y sus contrapartes en los demás concerns. El resto del Modelo de Contenido pasa igual al Modelo Unificado.

La Regla de Transformación 3.1, muestra visualmente los grafos *L* y *R* definidos para esta regla. La condición para aplicar la regla es que la clase *A* del concern *Cj* sea un rol de la clase *A* del concerns *Ci* ($\text{roleOf}(A_{Cj}) = A_{Ci}$). Depende de cuales son las clases *Core* elegidas, si se cumple o no esta condición, y sólo hay reemplazo de *L* por *R* cuando dicha condición se cumple.



Regla de Transformación 3.1: Establecer la relación de rol, entre una clase (*Core*) y sus contrapartes en los demás concerns.

Al Modelo de Contenido presentado en la Figura 3.13, le aplicaremos la Regla de Transformación 3.1, logrando como resultado obtener un Modelo Unificado. A continuación se muestra como a partir del Modelo de Contenido (Figura 3.13) y eligiendo diferentes concerns como *Core*, se pueden derivar diferentes Modelos Unificados. Esto es posible porque la Regla de Transformación 3.1 establece la condición que una de las clases sea rol de otra (que está en otro concerns). Por lo tanto, si varía el *Core* elegido, al aplicar la transformación se obtienen diferentes Modelos Unificados (cada uno con un *Core* en particular).

Si se toma el concern *Tourist* como *Core* entonces el Modelo Unificado queda conformado como se muestra en la Figura 3.14. Para las clases que no están

representadas en el concern turístico se debe elegir cuál es el concern *Core* de la misma, esto sucede con las clases *Church* y *Monument*. En la Figura 3.14.a estas clases tienen como *Core* el concern histórico mientras que en la Figura 3.14.b tienen como *Core* el concern arquitectural.

Hay otras clases que sólo están representadas en un único concern digital, en este caso se representan con ese concern como *Core*, como es el caso de las clases *ArchitecturalStyle* y *Architect*.

Las clases que están representadas en un único concern digital y en el concern físico, como son *Fountain* y *Monument*, se toma como *Core* de las mismas el único concern digital.

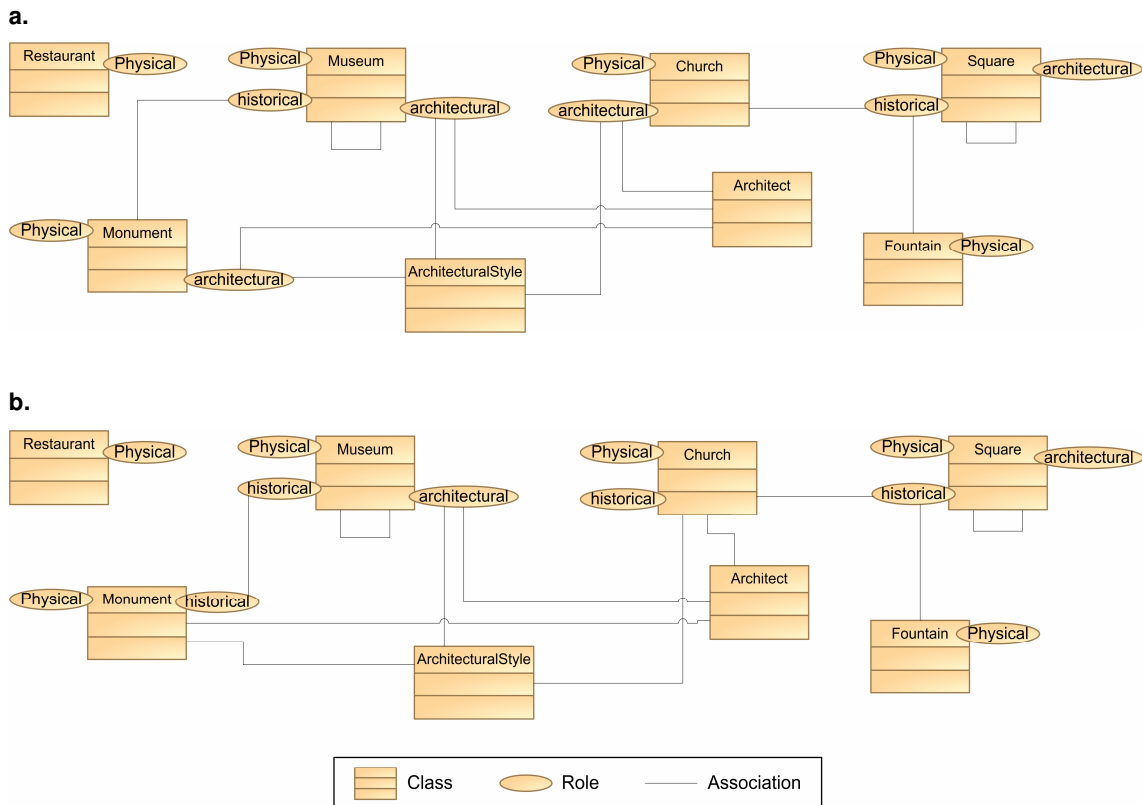


Figura 3.14: Modelo Unificado tomando como *Core* el concern turístico.

El Modelo Unificado resultante de la transformación depende de cuál fue el *Core* elegido para cada clase. Esto se puede visualizar en la Figura 3.14 donde a partir del mismo Modelo de Contenido (Figura 3.13) se obtuvieron dos Modelos Unificados diferentes, ya que al variar el *Core* de alguna de las clases la Regla de Transformación 3.1 da diferente resultado.

Hay diferentes criterios que se pueden usar para elegir cuál es el *Core* de una clase si esa clase no existe en el concern que se elige como *Core* de la aplicación. Una posibilidad es que sea al azar entre los demás concerns digitales que tiene la clase. Otra alternativa es tener priorizados los demás concerns y de esta manera poder elegir el concern *Core* de la clase según el nivel de prioridad de los mismos.

En el caso de tomar para el Modelo Unificado el concern histórico como *Core* hay un sólo modelo resultante al aplicar la Regla de Transformación 3.1, como se puede visualizar en la Figura 3.15.

Las clases *ArchitecturalStyle* y *Architect* son las únicas clases que no tiene una representación en el concern histórico, pero como estas clases aparecen en un único concern no hay otra opción de *Core* posible que el concern arquitectural.

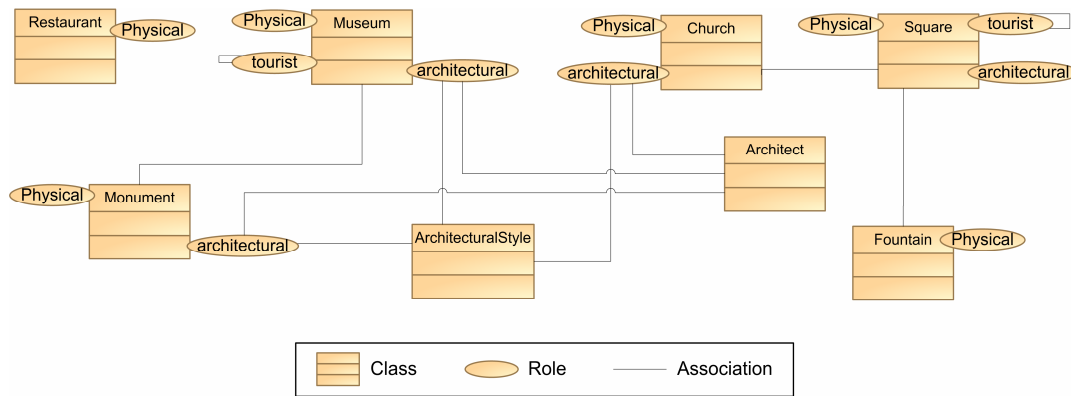


Figura 3.15: Modelo Unificado tomando como Core el concern histórico.

Se puede observar que en ninguno de los Modelos Unificados presentados se pierde información, sino que cambia el foco de cual es el concern *Core* de la aplicación. Es decir, cambia el punto de entrada para cada clase. Como se pudo observar también en los diferentes Modelo Unificados presentados, el concern físico siempre se pasa como un rol de la clase *Core*.

Se pueden tener distintos Modelos Unificados a partir del mismo Modelo de Contenido. Permitiendo especificar según el *Core* elegido cuál va a ser el foco de la aplicación.

3.3.3. Modelo Navegacional

Para realizar el Modelo Navegacional de una aplicación de HM nos focalizamos en el modelado de los nodos (digitales y físicos) y los links (digitales y caminables predefinidos).

En [Parunak, 1989] se especifica como Hipermedia se puede representar como un grafo, donde los nodos son los vértices y los links las aristas. La definición detallada por [Parunak, 1989], no nos es suficiente para representar nuestro Modelo Navegacional, ya que nosotros necesitamos identificar nodos digitales y físicos, como así también links digitales y caminables predefinidos. Además, necesitamos representar la relación de rol entre cada nodo *Core* y sus roles. La Definición 3.7 muestra la definición del Modelo Navegacional (*NavigationModel*) como un grafo que cuenta con nodos (los cuales pueden ser digitales o físicos), links (tanto sean digitales como caminables predefinidos) y roles²⁴ (entre los nodos *Core* y sus contrapartes en los demás concerns).

```

NavigationModel = (Nodes, Links, RoleOfNavigationModel)
Nodes = DigitalNodes U PhysicalNodes
Links = DigitalLinks U WalkingLinks
DigitalNodes = {DN1, ..., DNn}
PhysicalNodes = {PN1, ..., PNn}
DigitalLinks = {<DNi, DNj>: DNi, DNj ∈ DigitalNodes}
WalkingLinks = {<PNk, PNl>: PNk, PNl ∈ PhysicalNodes}
RoleOfNavigationModel = {<N1, RoleOf, Nk>: (N1, Nk ∈ Nodes) and
(N1 ∈ Cj and Nk ∈ Ci, i≠j and (Ci, Cj ∈ ConcernsNavigationModel))}

```

Definición 3.6: Grafo que representa el Modelo Navegacional.

²⁴ En [Challiol et al., 2007] definimos la función que permite brindar la contraparte física de un nodo, esto se puede ver como un rol acotado a la contraparte física. En esta tesis ampliamos ese concepto para que la relación se establezca entre un nodo y todas sus contrapartes en los distintos concerns.

Como se puede apreciar en la Definición 3.6, la especificación de la relación de rol ($RoleOf_{NavigationModel}$) se establece en base a los concerns del Modelo Navegacional ($Concerns_{NavigationModel}$). Esto se debe a que esta relación se establece entre nodos de diferentes concerns. Los concerns de este modelo también se pueden representar como subgrafos como se muestra en la Definición 3.7.

$$Concerns_{NavigationModel} = DCS_{NavigationModel} \cup \{PC_{NavigationModel}\}$$

$$DCS_{NavigationModel} = \{DC_{1-NavigationModel}, \dots, DC_{n-NavigationModel}\}$$

$$DC_{i-NavigationModel} = (DigitalNodes_{CDi}, DigitalLinks_{CDi}),$$

$DC_{i-NavigationModel}$ es un subgrafo de $NavigationModel$, por lo tanto:

$$DigitalNodes_{CDi} \subseteq DigitalNodes \text{ y}$$

$$DigitalLinks_{CDi} \subseteq DigitalLinks$$

$$PC_{NavigationModel} = (PhysicalNodes, WalkingLinks)$$

$PC_{NavigationModel}$ es un subgrafo de $NavigationModel$

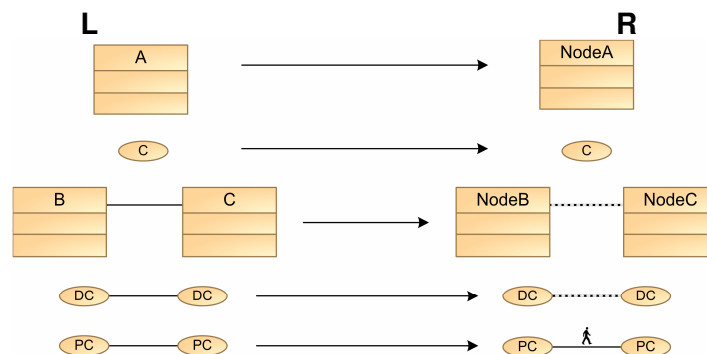
$$DigitalNodes_{DCi} = \{Node_{DCi} : (Node_{DCi} \in DigitalNodes) \text{ and}$$

$$(Node_{DCi} \notin DigitalNodes_{DCj}, i \neq j)\}$$

Definición 3.7: Subgrafos que representan los concerns del Modelo Navegacional.

El Modelo Navegacional es considerado una vista del Modelo Unificado. Es decir, a partir del Modelo Unificado se puede derivar el Modelo Navegacional. Los nodos se crean a partir de las clases del Modelo Unificado. Un nodo puede reflejar la información de una clase o agrupar información de varias clases. Los roles de las clases del Modelo Unificado pasan como roles de los nodos. Cada nodo o rol define que atributos debe mostrar. Las relaciones del Modelo Unificado pasan como links. Las relaciones entre las clases de los concerns digitales pasan al Modelo Navegacional como links digitales mientras que las relaciones entre los roles físicos pasan como links caminables predefinidos.

Veamos como especificar la regla de transformación (de grafos) necesaria para obtener un Modelo Navegacional a partir de un Modelo Unificado ($UnifiedModel \Rightarrow_{R3.2} NavigationModel$). La Regla de Transformación 3.2 muestra visualmente los grafos L y R definidos para esta regla. En este caso no hay una condición establecida que se deba cumplir para aplicar la regla, esto quiere decir que cada ocurrencia de L se reemplaza automáticamente con su correspondencia en R . Se puede apreciar (en la Regla de Transformación 3.2) que las clases pasan a nodos, los roles siguen siendo roles (en el modelo resultante, este caso el Modelo Navegacional), las relaciones entre clases o roles digitales pasan como links digitales mientras que las relaciones de los roles físicos pasa como links caminables predefinidos.



Regla de Transformación 3.2: Pasar clases a nodos, roles de las clases como roles de los nodos y las relaciones a links digitales o caminables predefinidos.

Si la Regla de Transformación 3.2 se aplica a un Modelo Unificado se obtiene como

resultado un Modelo Navegacional. Supongamos que tomamos el Modelo Unificado que tiene el concern histórico como *Core* (como se mostró en la Figura 3.15), el Modelo Navegacional que se deriva (aplicando la Regla de Transformación 3.2) se puede visualizar en la Figura 3.16. Donde cada clase paso a ser un nodo, los roles de las clases pasan a ser roles de los nodos y las relaciones pasan a ser links (tanto entre los nodos como entre roles). En este caso particular, sólo se crearon links digitales, ya que el Modelo Unificado no tenía relaciones entre roles físicos.

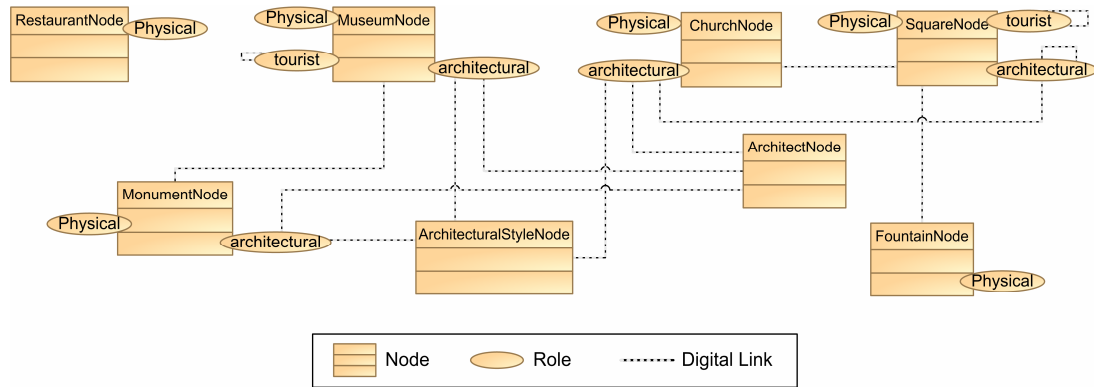


Figura 3.16: Modelo Navegacional a partir del Modelo Unificado con Core Histórico.

El diseñador puede refinar el Modelo Navegacional (resultante de la transformación) ajustándolo a las necesidades de cada aplicación. En la etapa de refinamiento es donde se debe elegir qué clases quedan efectivamente como nodos, es decir, es información relevante para el usuario. El diseñador puede crear nuevos links según lo crea conveniente para cada aplicación particular. Es decir, se pueden crear links digitales o caminables predefinidos que no tengan su correspondencia en el Modelo Unificado. Los links caminables (predefinidos o derivados) van a expresar que hay un camino entre dos objetos físicos (al menos una forma de llegar desde uno a otro).

Cada rol físico va a tener links caminables predefinidos especificados por el diseñador de la aplicación. Los links caminables derivados se calculan mediante alguna función de contexto, por ejemplo la que considera que información digital está visualizando el usuario (como se explicó en la Sección 3.2.1); esta función se debe especificar para cada rol físico.

A diferencia del Modelo Unificado en este modelo puede haber clases propias del modelo de dominio que el diseñador sabe que no deben ser representados como nodos de la HM, por ejemplo, clases que representan algoritmos. Es decir, el diseñador puede eliminar tanto nodos como links del Modelo Navegacional obtenidos a partir de un Modelo Unificado, como así también agregar los nodos y links que crea necesarios.

Un Modelo Navegacional refinado, a partir del modelo presentado en la Figura 3.16, es visualizado en la Figura 3.17. En la etapa de refinamiento es donde se debe elegir que clases quedan efectivamente como nodos, podemos ver que la clase *Restaurant* se eliminó como nodo en el Modelo Navegacional refinado, ya que se decidió que no sea información relevante para el usuario. Estas decisiones deben ser tomadas por el diseñador de la aplicación.

Se puede apreciar en la figura que en el modelo refinado se agregan links caminables predefinidos entre los roles físicos de algunos nodos (marcados en la Figura 3.17 como *Added Walking Link*). Se crea un link caminable del *MuseumNode* al *MonumentNode* y otro al *ChurchNode*, un link caminable del *ChurchNode* al *MonumentNode* y por último un link caminable del *MonumentNode* a sí mismo. Estos links caminables predefinidos son los que luego el usuario visualiza en su dispositivo como *Planned Routes* cuando sea sensado por ese objeto del mundo real. Estos nuevos links no tienen su correspondencia en el Modelo Unificado.

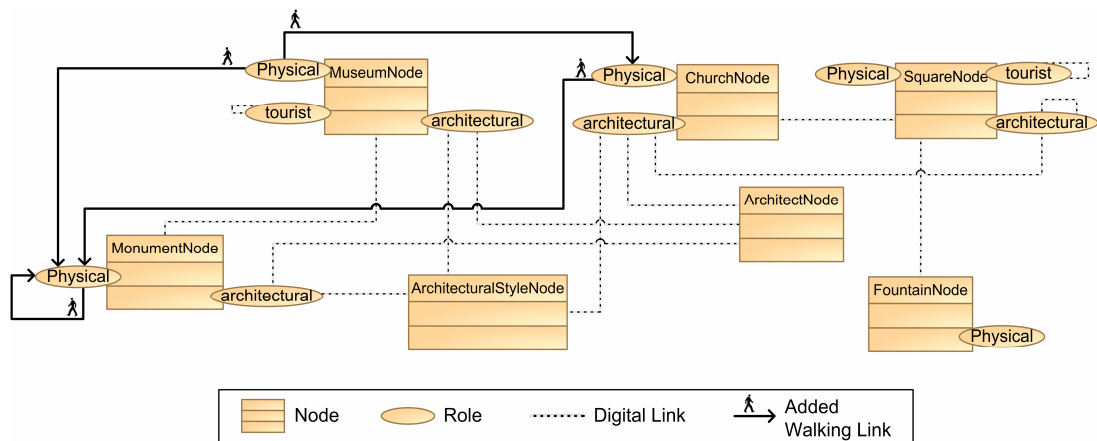


Figura 3.17: Modelo Navegacional refinado.

Cabe destacar que en el Modelo Navegacional (Figura 3.17) no se especificaron los links caminables derivados, ya que estos surgen dinámicamente según la información que el usuario está visualizando en un momento dado. Sin embargo, si se puede definir en este modelo la función que calcula los links derivados. Es decir, cada rol físico puede tener definido la función para el cálculo de los links caminables derivados.

En resumen, primero se logra un Modelo Navegacional a partir del Modelo Unificado y luego este es refinado por el diseñador. En la instancia de refinamiento se eligen que clases (o roles) se dejan como nodos (o roles) y cuales son eliminadas del Modelo Navegacional. Lo mismo ocurre con los links. Se puede derivar tantos Modelos Navegacionales como sean necesarios a partir del mismo Modelo Unificado, por ejemplo uno para cada perfil de usuario, para que los mismos puedan reflejar información de navegación acorde a cada perfil.

3.3.4. Modelo de Presentación y Modelo del Usuario

El Modelo de Presentación se define para un Modelo Navegacional determinado, donde se establece cómo visualiza el usuario cada uno de los nodos. Todas las instancias de un nodo se van a mostrar de la misma forma (cuando se usa un Modelo de Presentación particular).

Se puede definir más de un Modelo de Presentación asociado a un mismo Modelo Navegacional. De esta manera con un mismo Modelo Navegacional se pueden lograr diferentes visualizaciones asociadas al mismo. El Modelo de Presentación está muy ligado a la implementación que se elija para desarrollar la aplicación, por esta razón a nivel conceptual no se puede dar más nivel de detalle. En la Sección 5.7 se detalla un posible Modelo de Presentación.

Algo similar pasa con el Modelo del Usuario. Este modelo debe contener al menos la información referente a que nodo digital está viendo el usuario como así también en que nodo físico se encuentra parado el usuario. Se debe guardar información referente a la navegación del usuario tanto a nivel digital como en el mundo real. El Modelo del Usuario debe conocer una instancia particular de un Modelo Navegacional y cuál es la presentación que debe recibir (el usuario) para cada nodo. Además, registrar la navegación del usuario tanto digitalmente como en el mundo real.

Este modelo puede contener además información del perfil del usuario como así también alguna otra información del contexto necesaria para la aplicación. Una posible implementación de Modelo del Usuario es presentada en la Sección 5.8.

3.4. Resumen

En la Sección 3.1 se detallaron los conceptos básicos de las aplicaciones de HM, como son los objetos digitales y físicos como así también como se clasifican los links entre los mimos (links digitales y links caminables respectivamente). Se especificó la diferencia entre navegar digitalmente (cuando se selecciona un link digital) y navegar en el mundo real (cuando se selecciona un link caminable), esta última navegación implica que el usuario camine en el mundo real para obtener información del destino (target del link caminable).

Además, en la Sección 3.2 se describió el comportamiento de este tipo de aplicaciones cuando se considera el concepto de separación de concerns. Se mostró que se podían tener links caminables derivados a partir de la información digital que el usuario estaba visualizando, estos links se calculan dinámicamente mediante una función definida por el diseñador.

En la Sección 3.3 se describió de manera conceptual nuestro enfoque para modelar aplicaciones de HM mostrando las características principales de cada uno de nuestros modelos y las relaciones que existen entre los mismos. Se mostró cómo expresar el Modelo de Contenido, Modelo Unificado y Modelo Navegacional en términos de grafo. Además, se especificaron dos reglas de transformación entre grafos, una que permite obtener a partir del Modelo de Contenido el Modelo Unificado y la otra regla que permite obtener a partir del Modelo Unificado el Modelo Navegacional.

El Modelo de Contenido, el Modelo Unificado y el Modelo Navegacional son independientes de la implementación, pueden ser definidos y luego usados en diferentes arquitecturas o lenguajes. Por lo tanto, estos modelos son considerados *Plataforma-Independent Model (PIM)*.

En el enfoque, el Modelo de Presentación y el Modelo del Usuario están ligados con la implementación que se utilice para tener funcionando una aplicación. Estos modelos son considerados *Plataforma-Specific Model (PSM)*.

En la Figura 3.18 se pueden apreciar los modelos del enfoque clasificados en *Plataforma-Independent Model (PIM)* y *Plataforma-Specific Model (PSM)*.



Figura 3.18: Modelos de nuestro enfoque clasificados en PIM y PSM.

En el Capítulo 5 se especifican cada uno de los modelos del enfoque detallándolos de manera concreta, en base a la extensión de *UWE* para aplicación de HM que se presenta en el Capítulo 4.

4. EXTENSIÓN DE UWE PARA REPRESENTAR APLICACIONES DE HIPERMEDIA MÓVIL

En la Sección 3.3 se describió un enfoque para crear aplicaciones de HM en forma conceptual, especificando cada uno de los aspectos de nuestros modelos y los elementos involucrados en cada uno. En este capítulo se presenta nuestro enfoque desde un punto de vista más concreto. Partiendo de la base que una aplicación de este tipo es una aplicación de Hipermedia donde el usuario puede explorar tanto el mundo real como digital mediante algún dispositivo móvil. Esto motiva a elegir, como base, una metodología de Hipermedia ya existente y extenderla con los conceptos propios de HM.

Como se mencionó en la Sección 2.1, en [Schwinger et al., 2008] se hace una comparación considerando diferentes aspectos de distintas metodologías existentes para Hipermedia. Según el análisis presentado por [Schwinger et al., 2008], se detalla que *UWE* (UML-based Web Engenniering) [Koch et al., 2008] es la metodología que mejor se ajusta a las bases del desarrollo dirigido por modelos para aplicaciones de Hipermedia, ya que se basa en estándares como son *MOF* y *UML* para la construcción de sus modelos, define transformaciones entre modelos y provee herramientas para la generación automática de aplicaciones Web. Es decir, *UWE* cumple los tres puntos básicos (analizados en la Sección 2.1) del desarrollo dirigido por modelo: la representación directa (mediante el uso *MOF* y *UML*), la automatización (mediante el uso de herramientas y la definición de transformaciones entre modelos) y el uso de estándares. Estas características, hicieron que se la eligiera como metodología base para extenderla incorporando los conceptos específicos de HM.

A *UWE* no le es suficiente *UML* para la especificación de sus elementos, por lo tanto define un metamodelo con sus elementos. Para mantenerse dentro del uso de estándares, *UWE* está definido como una extensión conservativa de *UML*. Es decir, los elementos del metamodelo de *UML* no se modifican, sino que se usan perfiles de *UML* para agregar la semántica de los elementos necesarios por *UWE*. Esta característica, presenta como ventaja que cualquier herramienta *UML* soporta el modelado de *UWE*. La extensión de *UWE* que se presenta en esta tesis también va a ser una extensión conservativa, es decir, se define un perfil para incorporar los elementos específicos de HM.

En la Sección 4.1 se describe el metamodelo especificando los elementos de los modelos que son independientes de la plataforma (Modelo de Contenido, Modelo Unificado y Modelo Navegacional). En la Sección 4.2 se especifica un perfil de *UML* que detalla la representación de cada elemento definido en el metamodelo definido en la Sección 4.1, para lograr así tener una extensión conservativa de *UWE*.

4.1. Metamodelo para aplicaciones de Hipermedia Móvil

En un enfoque de desarrollo dirigido por modelos, el metamodelo juega un rol fundamental, ya que es donde se definen los elementos del lenguaje de modelado junto a sus relaciones. Además, es la base para definir las transformaciones entre modelos y permitir hacer chequeos sobre el contenido de cada modelo.

A partir del enfoque presentado en la Sección 3.3, veamos ahora cómo los conceptos presentados para cada modelo pueden definirse a nivel de metamodelo. De *UWE* se toma el Modelo de Contenido y el Modelo Navegacional como base para extenderlos con los elementos específicos de HM.

El Modelo de Contenido de *UWE* no define ningún elemento a nivel de metamodelo, ya que los conceptos de este modelo se pueden representar con los elementos de los diagramas de clases *UML*. En cuanto al Modelo Navegacional de *UWE* agrega a nivel de metamodelo todos los elementos relacionados con Hipermedia. Estos elementos de metamodelo son usados por el enfoque para representar la información digital de nuestro Modelo Navegacional.

A continuación se detallan los elementos específicos de HM que se definieron a nivel de metamodelo y que no están representados por *UWE*. Para hacer un mejor seguimiento iremos mostrando los elementos necesarios a nivel de metamodelo para definir nuestros Modelos de Contenido, Unificado y Navegacional.

Como se mencionó anteriormente *UWE* no agrega ningún elemento a nivel de metamodelo para su Modelo de Contenido. Pero en nuestro Modelo de Contenido surge el concepto de concern, el cual no está representado ni en *UML* ni en *UWE*. El elemento de *UML* que utilizamos para representar el concepto de concern, es *Package* [UML Superstructure], subclasificándolo para representar los concerns digitales y físicos como se puede ver en el Figura 4.1, cada una de estos paquetes contiene los elementos de los diagramas de clases de *UML*.

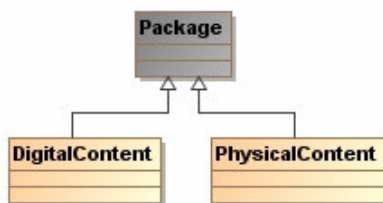


Figura 4.1: Metamodelo de los concerns para el Modelo de Contenido.

En el Modelo de Contenido (como se presentó en la Sección 3.3) las clases del concern físico deben definir la propiedad de ubicación. Este concepto no está representado ni en *UML* ni en *UWE*, por lo tanto es un concepto que define el enfoque a nivel de metamodelo.

Las aplicaciones móviles existentes modelan el concepto de ubicación de diferentes maneras, según las necesidades de cada aplicación móvil. Como este tema excede el foco de esta tesis, el metamodelo presentado para la propiedad de ubicación es muy simple considerando sólo ubicaciones geométricas [Leonhardt, 1998], simbólicas [Leonhardt, 1998] y etiquetadas. Sin embargo, este metamodelo se puede extender según las necesidades que surjan en futuras evoluciones del enfoque.

Para representar una propiedad que representa el concepto de ubicación, subclasificamos la clase *Property* [UML Superstructure] para darle la semántica adecuada a la propiedad de las clases. Como puede observarse en la Figura 4.2 se crearon tres subclases de *Property*.

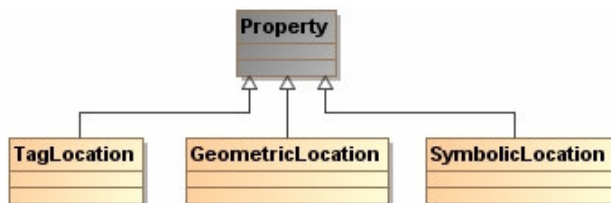


Figura 4.2: Metamodelo de la propiedad de ubicación.

La subclase *TagLocation* representa una etiqueta que identifica al objeto del mundo real. De esta manera, se pueden identificar objetos usando códigos de barra 1D o 2D. Esta clase de propiedad va a estar asociada a aquellos objetos del mundo real que se identifican, por ejemplo, mediante un código de barra. La subclase *GeometricLocation* especifica una coordenada geométrica. Mientras que la subclase *SymbolicLocation* al

objeto del mundo real ubica mediante un valor simbólico, por ejemplo “*La Basílica de San Pedro*”.

Con la definición a nivel de metamodelo de los concerns (digitales y físico) y la propiedad de ubicación tenemos todos los elementos necesarios para la representación de nuestro Modelo de Contenido.

Analicemos ahora el Modelo Unificado, éste conserva todos los elementos del Modelo de Contenido pero le agrega el concepto de rol dependiendo de cuál es el concern seleccionado como *Core* de la aplicación. Por lo tanto, para el Modelo Unificado se usan los elementos definidos a nivel de metamodelo del Modelo de Contenido y se agrega el concepto de rol (Figura 4.3). Este concepto no está representado ni en *UML* ni en *UWE*, entonces nosotros lo definimos a nivel de metamodelo.

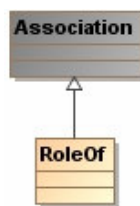


Figura 4.3: Metamodelo del rol.

El concepto de rol, se agrega siguiendo el enfoque de [Cabot and Raventós, 2006], que define el rol como una asociación denominada *RoleOf*. Esta asociación se usa para relacionar la clase *Core* con las otras clases que son roles de la misma. De esta manera se tienen todos los elementos a nivel de metamodelo para el Modelo Unificado.

Como vimos en la Sección 3.3, el Modelo Navegacional contiene como elementos nodos y links, en particular nodos y links digitales y, nodos físicos y links caminables predefinidos. Como *UWE* define elementos a nivel de metamodelo para representar los conceptos de las aplicaciones de Hipermedia, estos conceptos se usan en el enfoque en forma completa sin ser modificados para representar los nodos y links digitales. El metamodelo de *UWE* para su Modelo Navegacional se muestra en la Figura 4.4.

UWE posee varios elementos definidos a nivel de metamodelo para el Modelo Navegacional. Por simplicidad, en los ejemplos presentados en esta tesis nos focalizaremos principalmente en el uso de *NavigationClass* y *NavigationLink*. Pero todo este metamodelo se puede utilizar en nuestro Modelo Navegacional para modelar los conceptos que contienen los concerns digitales.

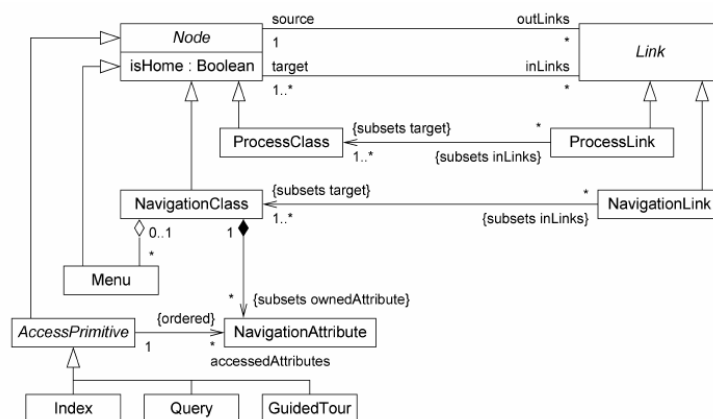


Figura 4.4: Metamodelo de *UWE* para su Modelo Navegacional [Koch et al., 2008].

UWE representa los nodos como subclases de *Class* [UML Superstructure] denominándola *NavigationClass* y los links como subclases de *Association* [UML Superstructure] denominándolos *NavigationLink*. Esto se puede visualizar en la Figura 4.5.

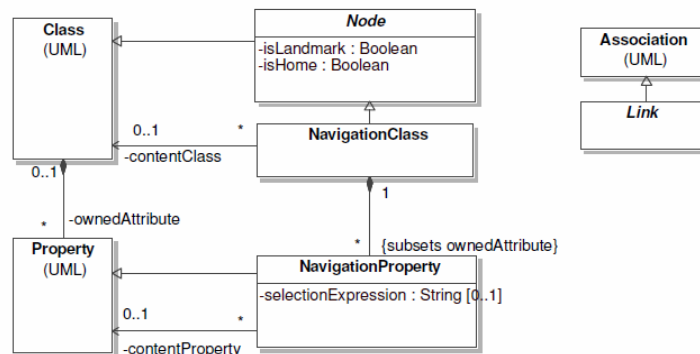


Figura 4.5: Relación entre los elementos del Metamodelo de *UWE* y *UML* [Kroib and Koch, 2008].

En cuanto a los nodos físicos y los links caminables predefinidos se definen específicamente para el enfoque. Para la definición del nodo físico se subclasifico la clase *Class* [UML Superstructure] mientras que para la definición del link caminable predefinido se subclasifico de *Association* [UML Superstructure]. Estas nuevas subclases se pueden apreciar en la Figura 4.6. Se puede visualizar que se denomina al nodo físico como *PhysicalNode* y al link caminable predefinido como *WalkingLink*.

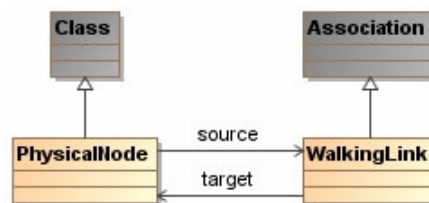


Figura 4.6: Metamodelo del nodo físico y link caminable.

En la Sección 3.3 se mencionó que había dos tipos de links caminables: los predefinidos, para los cuales definimos la asociación *WalkingLink*, y los derivados que son calculados dinámicamente. Para poder expresar a nivel de metamodelo la operación dinámica que calcula los links caminables derivados creamos dos²⁵ subclases de *Operation* [UML Superstructure] como se puede observar en la Figura 4.7. Una de las subclases se llama *AllHavePhysicalCounterparts*, esta operación toma todos los links digitales (del nodo digital) que está viendo el usuario en un momento dado y si estos tienen targets con contrapartes físicas, deriva links caminables a estos targets. La otra operación llamada *OnlyNearPhysicalCounterparts* es un poco más específica que la anterior ya que toma como links derivado sólo a aquellos targets digitales (de los links digitales del nodo digital que está viendo el usuario) que tienen contrapartes físicas y están dentro de un rango de alcance desde la ubicación actual del usuario.

Las operaciones *AllHavePhysicalCounterparts* y *OnlyNearPhysicalCounterparts* son explicadas con más nivel de detalle en los Capítulos 5 y 6.

De esta manera se provee la posibilidad que los nodos físicos tengan operaciones con la semántica de derivación de links caminables.

²⁵ En futuras evoluciones del enfoque se agregarán más operaciones de links caminables derivados para que el diseñador cuente con más opciones a la hora de modelar.

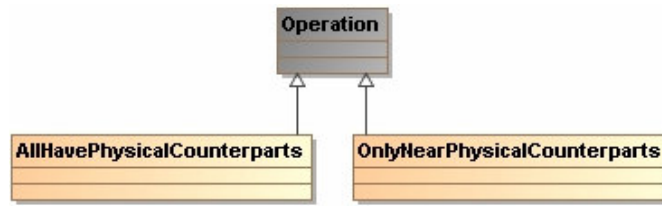


Figura 4.7: Metamodelo de las operaciones de links caminable derivados.

Dado que los elementos internos de los concerns de Modelo de Contenido o Unificado no tienen la misma semántica que los elementos internos del Modelo Navegacional; es decir, los elementos del Modelo de Contenido o Unificado son clases y relaciones mientras que los elementos del Modelo Navegacional son nodos y links, se crean nuevas subclase de *Package* que nos permiten reflejar esta diferencia. Estas subclases se pueden visualizar en la Figura 4.8. Se puede apreciar que el concern navegacional digital se denomina *DigitalNavigation*, este contiene los elementos del metamodelo definido por *UWE* como son, por ejemplo, *NavigationalClass* y *NaviationLink*. Mientras que el concern navegacional físico (denominado *PhysicalNavigation*) contiene a los nodos físicos (*PhysicalNode*) y los links caminables predefinidos (*WalkingLink*).

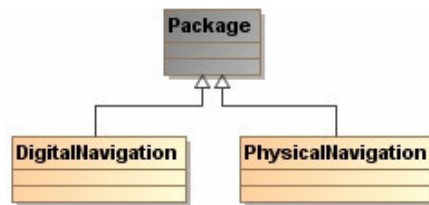


Figura 4.8: Metamodelo de los concerns para el Modelo Navegacional.

El Modelo Navegacional también cuenta con el concepto de rol, para el cual se usa el metamodelo definido para el Modelo Unificado, es decir la asociación *RoleOf* (Figura 4.3). El concepto de rol es algo genérico que se puede aplicar tanto a clases como a nodos sin tener la necesidad de crear un nuevo elemento a nivel de metamodelo. Lo mismo ocurre con el metamodelo de la propiedad de ubicación. Los nodos físicos (*PhysicalNode*) usan el metamodelo definido en la Figura 4.2 para especificar la propiedad de ubicación.

En la Figura 4.9 se puede apreciar la relación entre el metamodelo para HM y nuestros modelos (de Contenido, Unificado y Navegacional).

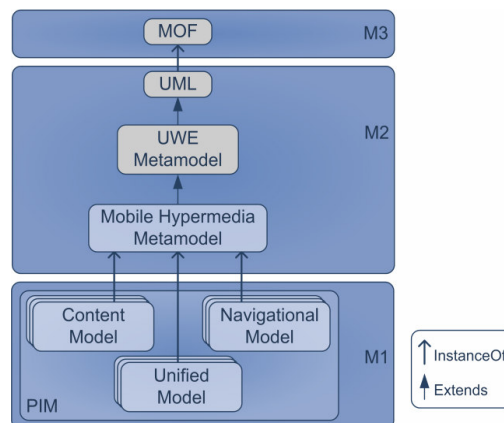


Figura 4.9: Relación entre los diferentes niveles de modelado considerando la definición de un metamodelo para aplicaciones de HM.

Se puede visualizar que la Figura 4.9 está detallada en base a los niveles de modelados relacionados con el desarrollo dirigido por modelos (descritos en la Sección 2.1). Donde el metamodelo definido para HM está especificado en un nivel M2 y los modelos del enfoque (Modelo de Contenido, Modelo Unificado y Modelo Navegacional) están en un nivel M1.

Se puede apreciar que nuestro metamodelo se define como una especificación de *UWE*, es decir utiliza todos los elementos de su metamodelo y agrega los elementos específicos de HM como se describió anteriormente. El único modelo que usa elementos del metamodelo de *UWE* es el Modelo Navegacional; esto sucede porque aprovecha los elementos de *UWE* para reasentar los elementos de los concerns digitales.

4.2. Perfil de *UML* para aplicaciones de Hipermedia Móvil

UML cuenta con el concepto de perfil [Fuentes and Vallecillo, 2004] que es un mecanismo que provee *UML* para extender su semántica permitiendo expresar los conceptos específicos de un dominio determinado. Para nuestro caso son las aplicaciones de HM. Este es el mecanismo que se utiliza para definir una extensión conservativa de *UML*.

Un Perfil según [Fuentes and Vallecillo, 2004] se define en un paquete *UML*, estereotipado como *Profile* (`<<profile>>`). Tres son los mecanismos que se utilizan para definir perfiles: estereotipos (stereotypes), restricciones (constraints), y valores etiquetados (tagged values).

Los estereotipos son un mecanismo de extensibilidad, el cual permite extender la semántica de los elementos de *UML* sin agregar elementos nuevos a nivel de metamodelo. El estereotipo tiene que definirse sobre un determinado elemento de *UML* como pueden ser *Package*, *Class*, *Association*, *Property*, etc.

Al estereotipo se le pueden agregar restricciones sobre el modelado y además definirles valores etiquetados que permiten definir valores asociados al estereotipo.

UWE define un perfil de *UML* para representar los elementos de su metamodelo. El enfoque define un perfil propio que cuenta con los estereotipos que representan al metamodelo presentado en la Sección 4.1. El nombre definido para nuestro perfil es *Mobile UWE* (el cual contiene los estereotipos que representan el metamodelo definido en la Sección 4.1).

La relación entre el perfil de *UWE* y nuestro perfil se puede visualizar en la Figura 4.10, donde nuestro perfil importa el perfil de *UWE*.

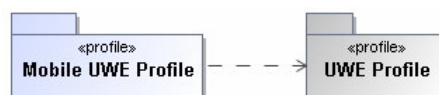


Figura 4.10: Perfil de HM.

Cada uno de los conceptos del metamodelo (de HM) presentados en la Sección 4.1 es especificado como estereotipo de distintos elementos de *UML*. Esto se realiza creando un elemento de *UML* denominado *Stereotype* [UML Superstructure]. El estereotipo se define para extender la semántica de algún elemento de *UML*. Para pasar los elementos del metamodelo a estereotipos se utilizaron las siguientes reglas:

- Los paquetes del metamodelo se transforman en estereotipos de *Package*. El nombre del paquete se va a corresponder con el nombre del estereotipo.
- Las clases del metamodelo se transforman como estereotipos de *Class*.

El nombre de la clase se va a corresponder con el nombre del estereotipo.

- Las asociaciones, propiedades u operaciones del metamodelo son pasadas a estereotipos de *Association*, *Property*, *Operation* respectivamente. El nombre de la asociación, propiedad u operación se va a corresponder con el nombre del estereotipo.

Veamos ahora cómo queda representado cada elemento del metamodelo como un estereotipo. La Figura 4.11 muestra los estereotipos que representan los concerns (del Modelo de Contenido y del Unificado). Por un lado, los concerns digitales y por otro lado el concern físico. Los dos elementos son estereotipos del elemento de *UML Package*. Tanto el Modelo de Contenido como el Modelo Unificado representan los concerns digitales mediante el elemento *Package* los cuales contienen el estereotipo `<<digitalContent>>` mientras que el concern físico se especifica con el elemento *Package* con el estereotipo `<<physicalContent>>`.

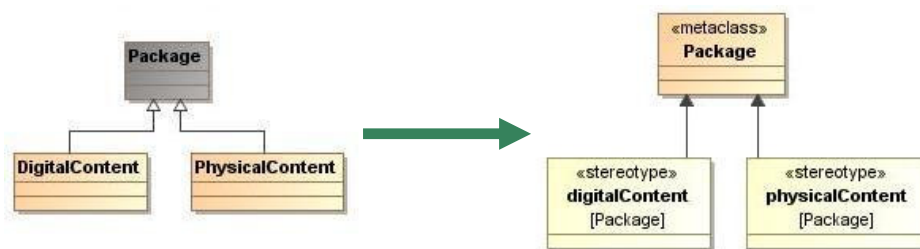


Figura 4.11: Estereotipos para representar los concerns del Modelo de Contenido y Unificado.

Para la propiedad de ubicación se habían definido tres subclases de *Property*, por lo tanto se definieron tres estereotipos del elemento *Property* como se puede visualizar en la Figura 4.12.

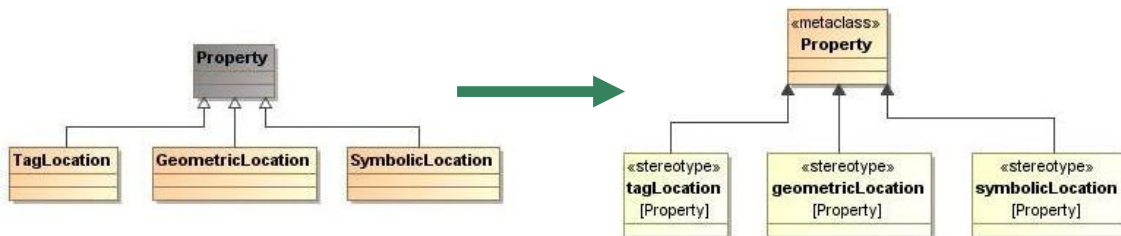


Figura 4.12: Estereotipos para representar la propiedad de ubicación.

Los modelos (de Contenido, Unificado y Navegacional) representan la propiedad y le agrega el estereotipo `<<tagLocation>>`, `<<geometricLocation>>` o `<<symbolicLocation>>` según corresponda, acorde a la ubicación que se quiera representar.

Como se mencionó en la Sección 4.1, el modelo de ubicación presentado para esta tesis es simple y puede evolucionar. Por tal motivo se eligió poner los estereotipos creados para representar la propiedad de ubicación en un perfil nuevo denominado *Location*. De esta manera, el perfil *Location* puede evolucionar de manera independiente al perfil *Mobile UWE*. Más aún, se puede utilizar en lugar de este perfil presentado algún otro perfil de ubicación ya existente para otra aplicación. El nuevo perfil agregado se puede visualizar en la Figura 4.13, se puede observar que no tiene relación con el perfil *Mobile UWE*.

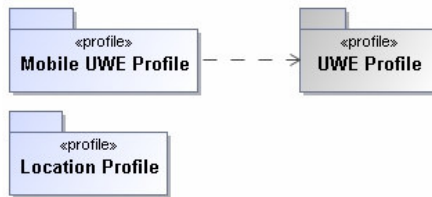


Figura 4.13: Perfil *Mobile UWE* y *Location*.

En cuanto a la asociación que representa el rol queda especificada como un estereotipo de *Association* como se puede visualizar en la Figura 4.14.

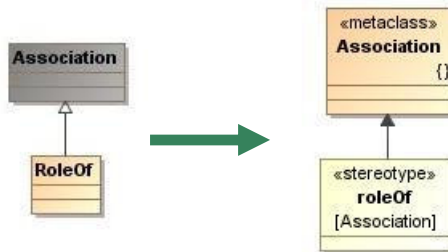


Figura 4.14: Estereotipo para representar la asociación de rol.

Tanto el Modelo Unificado como el Modelo Navegacional definen el rol como una asociación estereotipada como `<<roleOf>>`. Esta asociación debe ser direccional para que se puede determinar cuál es el elemento *Core*.

Como se mencionó en la Sección 4.1 se usa de *UWE* los elementos para la representación del nodo y link digitales. *UWE* tiene, a nivel de metamodelo, los elementos *NavigationClass* y *NavigationLink*, los cuales son representados en el perfil de *UWE* con el estereotipo de clase `<<navigationClass>>`²⁶ y el estereotipo de asociación `<<navigationLink>>`²⁶.

En la Figura 4.15 se puede apreciar los estereotipos definidos por *UWE* para su Modelo Navegacional en el perfil de *UWE* versión 1.8²⁷. Estos estereotipos se usan para representar los elementos de los concerns digitales en nuestro Modelo Navegacional.

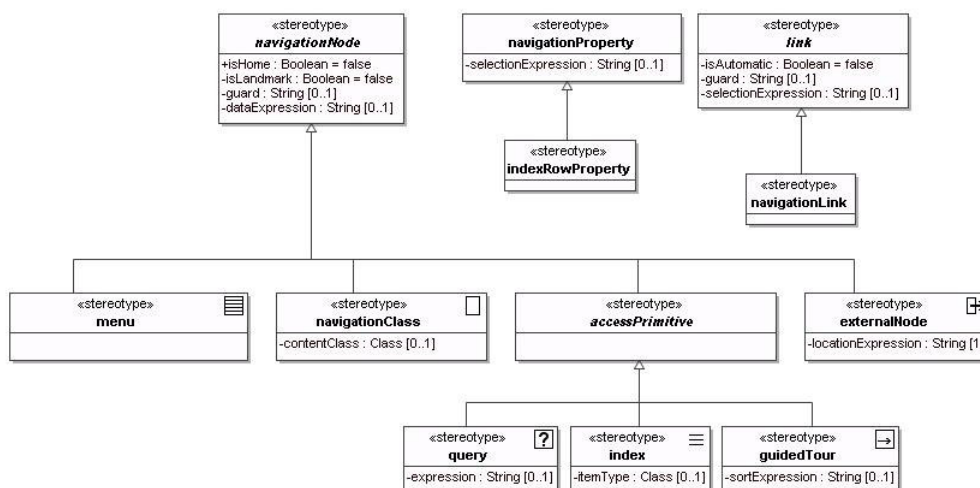


Figura 4.15: Estereotipos para el Modelo Navegacional en el perfil de *UWE* versión 1.8.

²⁶ Estereotipo definido en el perfil de *UWE*.

²⁷ Página del perfil *UWE* 1.8: http://uwe.pst.ifi.lmu.de/download/UWE_Profile_v1.8.zip

En los ejemplos presentados en esta tesis sólo se usan algunos de estos estereotipos, pero el enfoque soporta el uso de todos los estereotipos de la Figura 4.15.

En cuanto a cada nodo físico y link caminable predefinido pasan a estereotipos como se muestra en la Figura 4.16. Donde el nodo físico es un estereotipo de *Class* y es usado en el Modelo Navegacional como una clase con el estereotipo `<<physicalNode>>`. El link caminable predefinido se define como un estereotipo de *Association* y se usa en el Modelo Navegacional como una asociación estereotipada como `<<walkingLink>>`.

Se puede apreciar en la Figura 4.16 que estos dos estereotipos tienen dos logos asociados, estos son dibujados cuando se especifican en el Modelo Navegacional nodos físicos y links caminables predefinidos.

Como se puede apreciar en la Figura 4.16, el estereotipo `<<physicalNode>>` tiene un valor etiquetado llamado `contentPhysicalClass`, el cual fue creado para mantener consistencia con *UWE*, que permite especificar la clase origen de la cual se creó el nodo físico. *UWE* define para el estereotipo `<<navigacionClass>>` el valor etiquetado `contentClass` para representar la clase origen de la cual se derivó el nodo. En [Ruiz-González et al., 2009] se indica que este valor etiquetado es usado para sincronizar modelos de *UWE* y permitir hacer reingeniería.

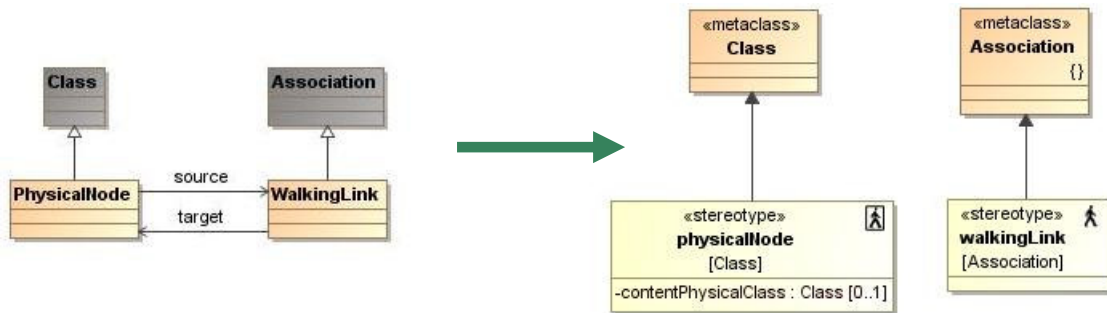


Figura 4.16: Estereotipos para representar el nodo físico y los links caminables predefinidos.

En cuanto a las operaciones para calcular los links caminables derivados se van a definir dos estereotipos de *Operation* como se ve en la Figura 4.17. Se puede observar que cada estereotipo define valores etiquetados, los cuales están relacionados a los parámetros que tienen estas funciones.

Se definen como valores etiquetados para que el diseñador tenga en cuenta que valores toma dicha operación como parámetro. Ambos estereotipos definen las etiquetas `core` (representa el concern *Core* de la aplicación), `concern` (representa el concern digital que está navegando el usuario) y `currentNode` (representa el nodo digital actual).

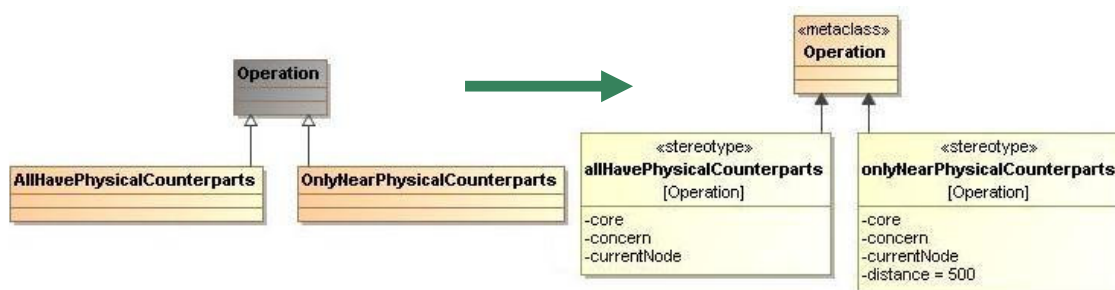


Figura 4.17: Estereotipos para representar las operaciones de links derivados.

El estereotipo `<<onlyNearPhysicalCounterparts>>` define además el valor etiquetado `distance`, que representa la medida del rango de alcance para filtrar los links caminables derivados. Es decir, este estereotipo considera sólo los links

caminables derivados cuyos targets (objetos físicos) están a lo sumo a la distancia establecida con `distance`. El valor por default que toma este valor etiquetado, es de 500 metros. Este valor puede ajustarse a las necesidades de cada aplicación según lo establezca el diseñador cuando define una operación con este estereotipo.

Los concerns (digitales y físico) del Modelo Navegacional pasan como estereotipos del elemento *Package* como se visualiza en la Figura 4.18. Estos estereotipos son especificados en el Modelo Navegacional mediante el uso de elementos *Package* estereotipados como `<<digitalNavigation>>` o `<<physicalNavigation>>` según corresponda.

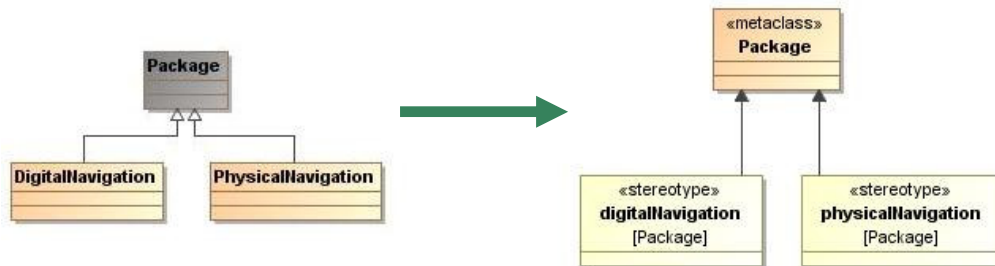


Figura 4.18: Estereotipos para representar los concerns navegacionales.

En la Figura 4.19 se puede apreciar la relación entre los diferentes perfiles definidos (*Mobile UWE* y *Location*) y nuestros modelos (de Contenido, Unificado y Navegacional). En el nivel M2 se encuentran todos los perfiles de UML. Cada uno de nuestros modelos usa elementos de UML y, los perfiles *Mobile UWE* y *Location*. En particular, el Modelo Navegacional usa elementos del perfil de UWE.

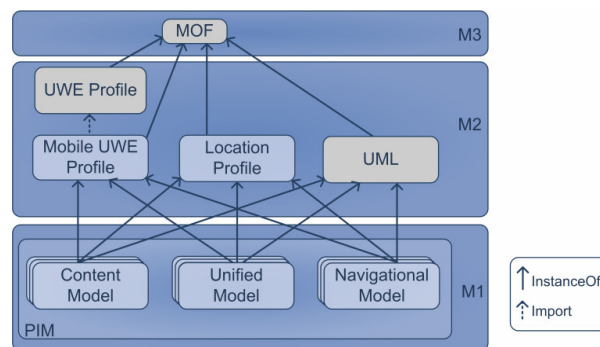


Figura 4.19: Relación entre los perfiles definidos y nuestros modelos.

Para que una aplicación no tenga que importar estos dos perfiles (*Mobile UWE* y *Location*) por separado se creó un perfil que permite agrupar todos los perfiles necesarios para las aplicaciones de HM. Este nuevo perfil se denomina *Mobile Hypermedia Profile*, e importa los perfiles necesarios para este tipo de aplicaciones. Como se ve en la Figura 4.20, este perfil importa el perfil de *Mobile UWE* y *Location*. En el Capítulo 5 se muestra que este perfil (*Mobile Hypermedia*) además importa los perfiles relacionados con los Modelos de Presentación y del Usuario.

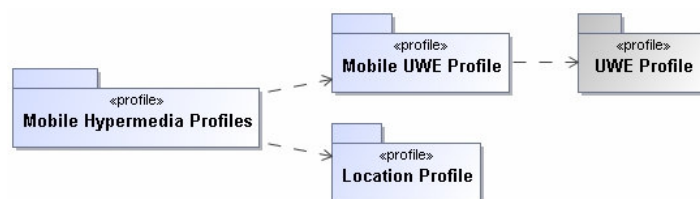


Figura 4.20: Perfil Mobile Hypermedia.

4.3. Resumen

Se mostró en este capítulo como realizar la extensión de *UWE* para modelar los conceptos propios de las aplicaciones de HM. En particular, se mostró en detalle el metamodelo especificado para aplicaciones de HM y cómo cada uno de estos elementos puede ser representado en un perfil de *UML*, logrando que nuestra extensión sea conservativa respecto de *UWE*.

En la Figura 4.21 se presentan todos los estereotipos definidos, tanto para el perfil *Mobile UWE* como para el perfil *Location*, brindando de esta manera una vista general de los elementos definidos. Se puede visualizar que el perfil *Mobile Hypermedia* importa los perfiles *Mobile UWE* y *Location*. El perfil *Mobile UWE* importa el perfil de *UWE*.

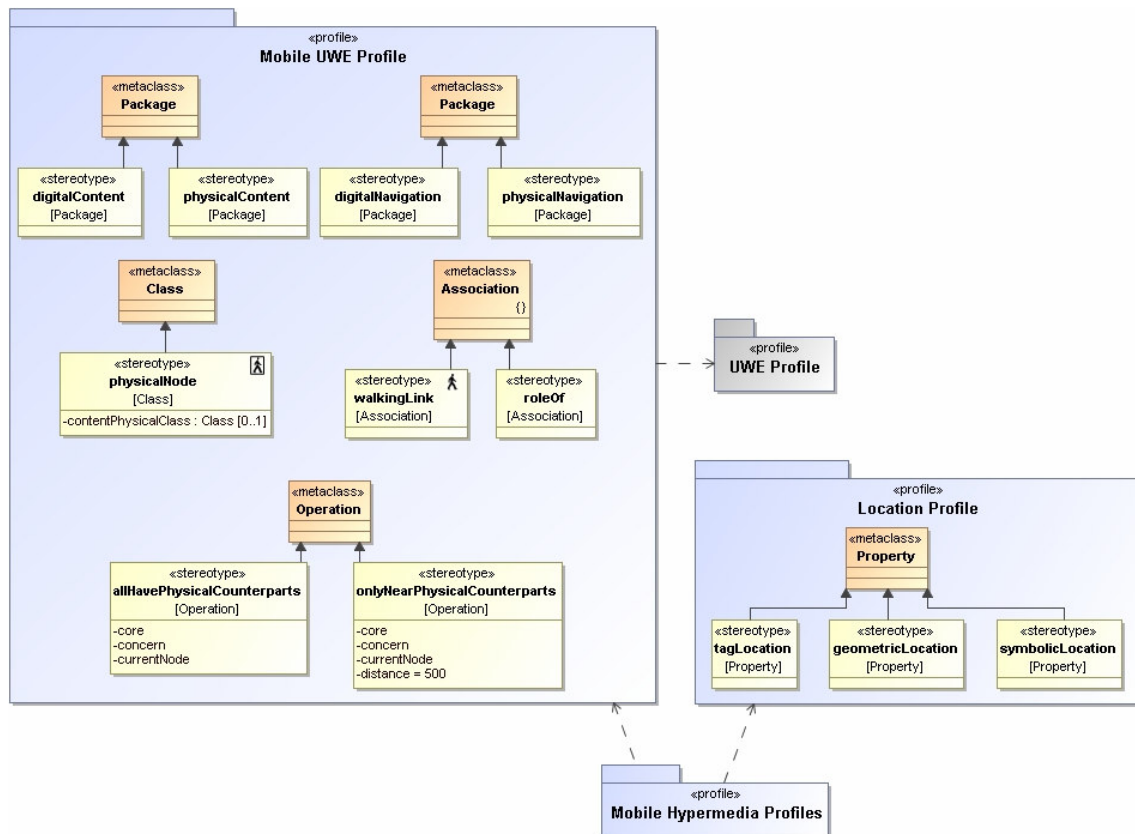


Figura 4.21: Estereotipos definidos para el perfil *Mobile UWE* y *Location*.

En la Tabla 4.1 se muestra un resumen del nombre de cada uno de los estereotipos junto al elemento *UML* al cual se le define el estereotipo. En la tercera columna se detalla en que modelos del enfoque son usados cada uno de estos estereotipos.

Tabla 4.1: Estereotipos definidos por el perfil *Mobile UWE* y *Location* para representar los conceptos de HM.

Estereotipo	Elemento de UML	Modelos de nuestro enfoque en los que se usa el estereotipo
digitalContent	Package	Modelo de Contenido Modelo Unificado
physicalContent	Package	Modelo de Contenido Modelo Unificado

Estereotipo	Elemento de UML	Modelos de nuestro enfoque en los que se usa el estereotipo
tagLocation	Property	Modelo de Contenido Modelo Unificado Modelo Navegacional
geometricLocation	Property	Modelo de Contenido Modelo Unificado Modelo Navegacional
symbolicLocation	Property	Modelo de Contenido Modelo Unificado Modelo Navegacional
roleOf	Association	Modelo Unificado Modelo Navegacional
digitalNavigation	Package	Modelo Navegacional
physicalNavigation	Package	Modelo Navegacional
physcialNode	Class	Modelo Navegacional
walkingLink	Association	Modelo Navegacional
allHavePhysicalCounterparts	Operation	Modelo Navegacional
onlyNearPhysicalCounterparts	Operation	Modelo Navegacional

En el Capítulo 5 se muestra cómo cada uno de estos estereotipos (definidos en nuestro perfil *Mobile Hypermedia*) es usado en cada uno de los modelos del enfoque para poder así representar los conceptos propios de las aplicaciones de HM.

5. ESPECIFICACION DE MODELOS DEL PROPUESTOS

En este capítulo se detalla cada uno de los modelos de nuestro enfoque, indicando qué elementos de nuestro perfil se utilizan en cada uno. Primero se presentan los modelos que son independientes de la plataforma (*Plataform-Independent Model - PIM*): el Modelo de Contenido, el Modelo Unificado, el Modelo Navegacional y el Modelo de Instancias Navegacional. Se especifican las dos transformaciones de modelo a modelo, una para transformar el Modelo de Contenido en Unificado y otra para transformar el Modelo Unificado en Navegacional. Estas dos transformaciones son clasificadas como $PIM \rightarrow PIM$, por el tipo de modelos que están involucrados.

Luego se detallan los modelos que son dependientes de la plataforma (*Plataform-Specific Model - PSM*) como son el Modelo de Presentación y el Modelo del Usuario. Para estos se define una posible representación para cada uno, para la cual se especifican elementos a nivel de perfil *UML* para seguir trabajando con una extensión conservativa de *UWE*.

5.1. Modelo de Contenido

Como se describió en la Sección 3.3.1 el Modelo de Contenido define los conceptos relevantes del dominio de la aplicación y las relaciones que existen entre los mismos. Todos los modelos del enfoque se realizan usando diagramas de clases de *UML*. El Modelo de Contenido tiene definido los concerns digitales y físico acorde a la aplicación que se quiera modelar, cada uno de los concerns define sus clases y las asociaciones entre las mismas.

La Figura 5.1 muestra un Modelo de Contenido para el ejemplo del turista, se muestra que se modelaron dos concerns digitales (el histórico y el arquitectural) y el concern físico. Se puede apreciar que hay clases que tienen su contrapartida en todos los concerns como son *Square*, *Monument*, *Church* y *Museum*. Mientras que otras clases tienen representación sólo en algunos de los concerns, como es el caso de la clase *Fountain*, que tiene sólo representación en el concern arquitectural y físico. También puede observarse que las clases *Architect* y *ArchitecturalStyle* sólo tienen representación en el concern arquitectural. Es decir, estas dos clases no tienen representación en el mundo real.

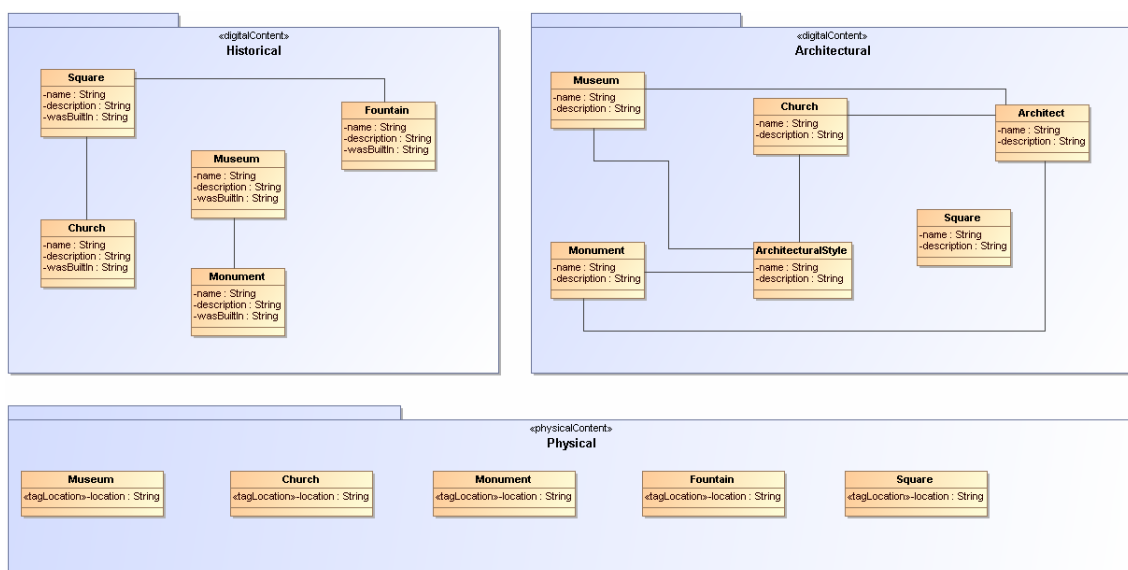


Figura 5.1: Ejemplo de un Modelo de Contenido para la aplicación del Turista.

Se puede apreciar en la Figura 5.1 que los concerns digitales se definen usando el estereotipo `<<digitalContent>>` mientras que el concern físico usa el estereotipo `<<physicalContent>>`. Dentro de cada concern, las clases definen sus propiedades acordes a la naturaleza de la misma, por ejemplo para el concern histórico las clases definen la propiedad `wasBuiltIn`. Cada clase puede tener propiedades propias que otras clases pueden no definir.

En cuanto a las propiedades de ubicación en la Figura 5.1 (que tienen las clases del concern físico) están estereotipadas como `<<tagLocation>>`. Esto quiere decir que los objetos de estas clases van a tener una representación en el mundo real identificados mediante una etiqueta.

Para el ejemplo de la Figura 5.1, las asociaciones entre clases se establecieron en forma bidireccional a modo de simplificar la visualización del modelo, y hacer foco sólo en los detalles importantes (del enfoque). Sin embargo, en este modelo se pueden especificar tanto asociaciones bidireccionales como unidireccionales detallando tanto el nombre como la cardinalidad de las mismas.

Cabe destacar que en el Modelo de Contenido se puede usar cualquier elemento de los diagramas de clases de *UML*: generalización, agregación, composición, etc. Es decir, en este modelo el diseñador puede ser tan específico y descriptivo como sea necesario para la aplicación. Además de poder usar los estereotipos definidos por el enfoque los cuales se listan en la Tabla 5.1.

Tabla 5.1: Estereotipos definidos por el perfil *Mobile UWE* que se usan en el Modelo de Contenido.

Stereotype	UML Element
<code>digitalContent</code>	Package
<code>physicalContent</code>	Package
<code>tagLocation</code>	Property
<code>geometricLocation</code>	Property
<code>symbolicLocation</code>	Property

5.2. Modelo Unificado

El Modelo Unificado (como se definió en la Sección 3.3.2) permite unificar todas las clases que aparecen en el Modelo de Contenido independientemente del concern donde estén representadas. Este modelo especifica un concern como *Core* de la aplicación, las clases de los demás concerns aparecen como roles de las clases del concern *Core*. Si hay clases que no están especificadas en el concern *Core* pero aparecen en otros concerns digitales, se elige uno de estos como *Core* para esta clase.

Cuando se genera el Modelo Unificado, a partir del Modelo de Contenido, no se debe agregar ni sacar información, sino que se logra relacionar las clases por la relación de rol.

La Figura 5.2 muestra un Modelo Unificado generado a partir del Modelo de Contenido presentado en la Figura 5.1, donde se eligió el concern histórico como *Core*. Se puede apreciar que se mantiene la información del Modelo de Contenido y se agregan las relaciones estereotipada como `<<roleOf>>`. En el caso de las clases *Architect* y *ArchitecturalStyle* al no tener representación en el concern histórico y solamente aparecer en el concern arquitectural, este concern pasa a ser el *Core* de dichas clases. El resto de las clases aparecen en el concern histórico, por lo tanto, éste es el concern *Core* de las mismas.

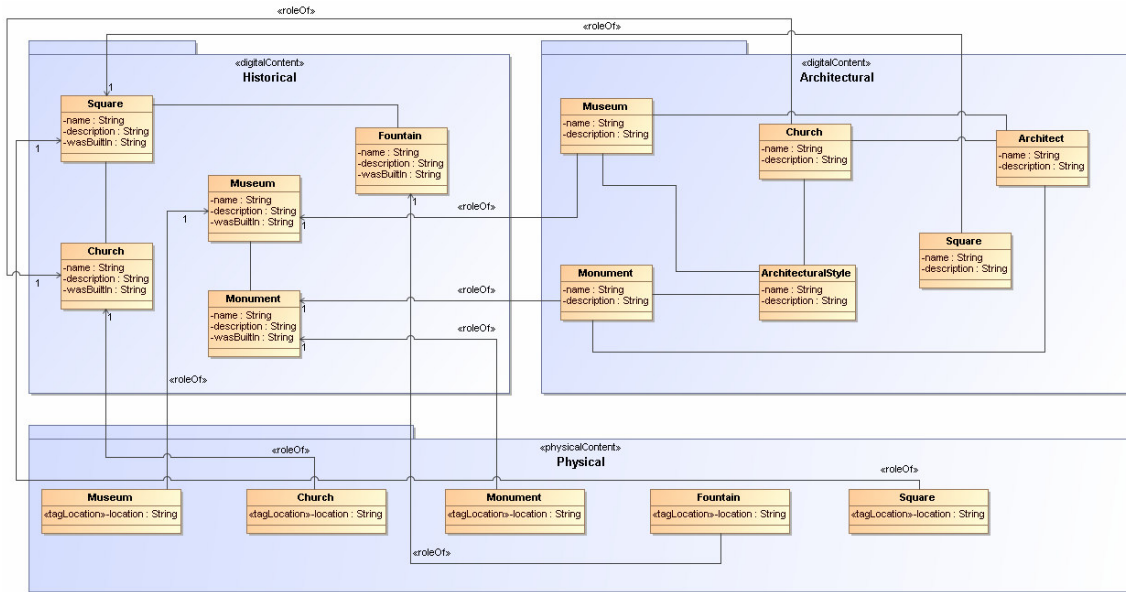


Figura 5.2: Modelo Unificado para la aplicación del Turista (Core Histórico).

Veamos ahora en la Figura 5.3, el Modelo Unificado que se genera (a partir del Modelo de Contenido presentado en la Figura 5.1) si se eligiera el concern arquitectural como Core. Se puede apreciar que hay clases que no tienen representación en el concern elegido como Core (arquitectural) como es la clase *Fountain*. Para esta clase se elige como Core de la misma la representación que existe en el concern histórico ya que es el único concern digital disponible para esta clase. El resto de las clases tienen su representación en el concern arquitectural, por lo tanto, éste es el Core de las mismas.

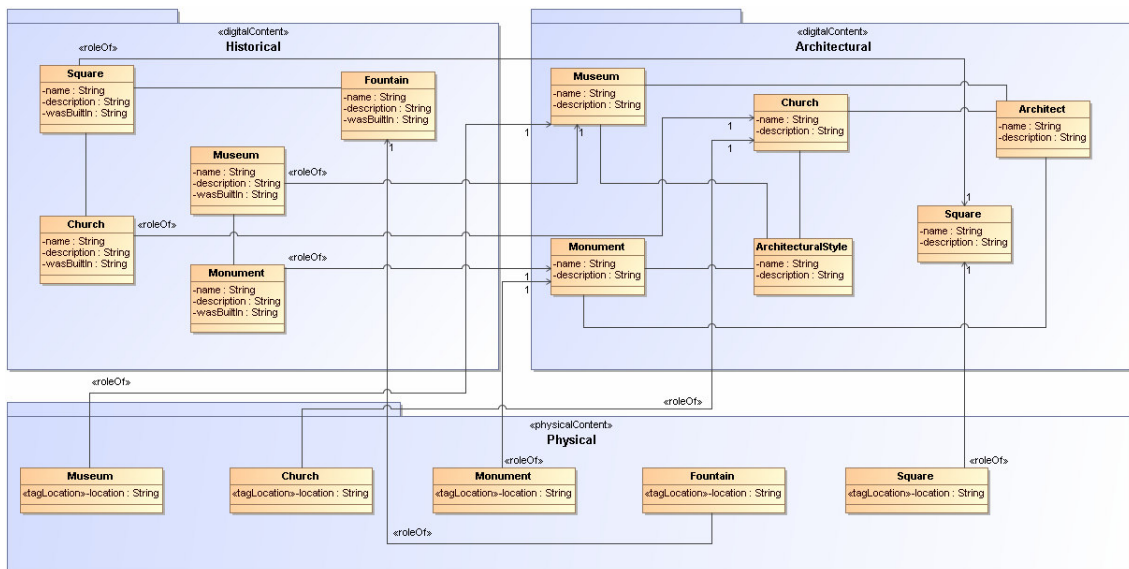


Figura 5.3: Modelo Unificado para la aplicación del Turista (Core Arquitectural).

Se puede observar en las Figuras 5.2 y 5.3 que para la representación de las relaciones de roles se usó una asociación direccional estereotipada como `<<roleOf>>`. La dirección de la asociación permite identificar cuál de las dos clases es Core. Las asociaciones estereotipadas como `<<roleOf>>` se establecen entre clase de distintos paquetes que tienen igual nombre.

En el caso que las clases no tuvieran igual nombre en los diferentes paquetes el diseñador también puede agregar la relación de rol entre las mismas. Esta situación la

puede determinar únicamente el diseñador ya que es el que conoce la interpretación del modelo acorde al dominio de la aplicación que se está modelando.

Para el Modelo Unificado se usan los estereotipos expresados en la Tabla 5.2. Se mantiene el uso de los estereotipos usados en el Modelo de Contenido y se agrega el uso del estereotipo `<<roleOf>>`.

Tabla 5.2: Estereotipos definidos por el perfil *Mobile UWE* que se usan en el Modelo Unificado.

Stereotype	UML Element
<code>roleOf</code>	Association
<code>digitalContent</code>	Package
<code>physicalContent</code>	Package
<code>tagLocation</code>	Property
<code>geometricLocation</code>	Property
<code>symbolicLocation</code>	Property

A partir de observar las Figuras 5.2 y 5.3 se puede apreciar que no importa como se realiza la composición de los concerns, la información disponible no se ve afectada. Es decir, la elección del concern que actúa como *Core* permite cambiar el foco de la aplicación pero se mantiene disponible la misma información. Se puede apreciar en ambas figuras que no se agregó ni sacó información, sólo se agregaron las asociaciones que representan al rol.

5.3. Transformación del Modelo de Contenido al Modelo Unificado

En la Sección 3.3.2 se describió una regla de transformación de grafos para obtener un Modelo Unificado a partir de un Modelo de Contenido. En esta sección se detalla cómo cada uno de los elementos del Modelo de Contenido se transforma al Modelo Unificado, en término de los elementos de un diagrama de clases *UML* y los estereotipos del enfoque involucrados.

Para realizar la transformación del Modelo de Contenido al Modelo Unificado se utilizan las siguientes reglas:

- Los paquetes del Modelo de Contenido pasan como paquetes en el Modelo Unificado con el mismo nombre y estereotipo (`<<digitalContent>>` o `<<physicalContent>>`).
- Las clases del Modelo de Contenido pasan como clases en el Modelo Unificado con el mismo nombre dentro del paquete correspondiente. En las clases del Modelo Unificado se crean las mismas propiedades que la clase origen (del Modelo de Contenido), manteniendo los nombres y tipo de las mismas.
- En el caso de la propiedad de ubicación, se crea dicha propiedad en la clase correspondiente (del Modelo Unificado) con el estereotipo que tiene la propiedad origen (`<<tagLocation>>`, `<<geometricLocation>>` o `<<symbolicLocation>>`) en el Modelo de Contenido.
- Las asociaciones entre dos clases del Modelo de Contenido, pasan a ser asociaciones entre las clases correspondientes del Modelo Unificado. Las asociaciones creadas en el Modelo Unificado mantienen el nombre y cardinalidades de las asociaciones origen.
- Se debe elegir un concern digital como *Core* de la aplicación. Según este

concern elegido se realizan los siguientes pasos:

- Las clases que están en el concern *Core* no se modifican. Si estas clases tienen su correspondencia (mediante mismo nombre) en algún otro concern, estas clases (que están en los otros concerns) agregan una asociación estereotipada como `<<roleOf>>` a la clase *Core*.
- Si una clase no tiene representación en el concern *Core* se elige uno de los concerns digitales en los que aparece la clase y se toma este como *Core* para dicha clase. Las otras clases que aparecen en los otros concerns que no son el concern *Core* elegido para esta clase, agregan una asociación estereotipada como `<<roleOf>>` a la clase *Core*.

La transformación puede ser más específica y tener en cuenta sólo algunos de los concerns. Es decir, crear un Modelo Unificado sólo de los concerns de interés. Esto permite que en la unificación, sólo queden representados los concerns de interés, filtrando aquellos concerns que, por ejemplo, son sólo aplicativos y, por lo tanto, no formarán parte de la navegación de la aplicación. Se pueden elegir cuales de los concerns digitales son de interés pero al menos el Modelo Unificado debe tener un concern digital el cual es el *Core* de la aplicación. No se puede filtrar el concern físico, este concern siempre se pasa al Modelo Unificado.

En el caso de no considerar todos los concerns en la transformación, las reglas anteriormente descritas sólo se aplican considerando los concerns de interés. Es decir, se tiene en cuenta sólo las clases que aparecen en los concerns de interés para hacer la transformación.

En el caso que las clases no tuvieran igual nombre pero el diseñador sabe que éstas son roles de otras que aparecen en otros concerns. El diseñador puede agregar manualmente la asociación que representa al rol. De esta manera se está refinando el Modelo Unificado. Sólo el diseñador tiene el conocimiento para determinar si dos clases con distintos nombre pueden relacionarse mediante la asociación de rol.

Una vez realizada la transformación, el diseñador puede a lo sumo agregar alguna asociación de rol que este faltando pero no puede sacar ni agregar otros elementos, es decir, no puede agregar concerns, clases, propiedades u otras asociaciones.


Puede pasar que a partir de un mismo Modelo de Contenido se realicen distintas transformaciones considerando en las mismas distintos concerns de interés. Creando de esta manera distintos Modelos Unificados para el mismo Modelo de Contenido.



5.4. Modelo Navegacional

El Modelo Navegacional representa cuáles son los nodos y links de la aplicación que se está modelando (como se definió en la Sección 3.3.3). A continuación se mencionan los elementos que se utilizan en este modelo describiendo los estereotipos.

Para este modelo se utiliza, a nivel de concerns digitales, el estereotipo `<<digitalNavigation>>` mientras que para el concern físico se utiliza el estereotipo `<<physicalNavigation>>`.

Dentro de los concerns digitales se utilizan los estereotipos que define *UWE* para su Modelo Navegacional, para los ejemplos presentados en esta tesis se usa mayormente los estereotipos `<<navigationClass>>` y `<<navigationLink>>`. Estos estereotipos sirven para representar los nodos y links digitales respectivamente. El estereotipo `<<navigationClass>>` agrega no solamente el estereotipo a la clase sino

que también tiene asociado el logo . El logo permite una rápida identificación del elemento cuando éste se encuentra dentro de un modelo.

En el caso del concern físico, los elementos que se usan para representar al nodo físico y link caminable (predefinido) son nuestros estereotipos `<<physicalNode>>` y `<<walkingLink>>` respectivamente. Cada uno de estos define un logo particular, el estereotipo `<<physicalNode>>` define el logo , mientras que el estereotipo `<<walkingLink>>` define el logo . Los logos están relacionados a la movilidad que debe realizar el usuario tanto para visualizar el nodo físico como para seguir un link caminable.

En el Modelo Navegacional se va a usar también el estereotipo relacionado a la asociación que representa el rol (asociación estereotipada como `<<roleOf>>`). Esta asociación se utiliza en este modelo para establecer cuál es el nodo del concern *Core* y cuáles son los nodos que son roles de este nodo.

También se usan los estereotipos relacionados a la propiedad de ubicación como son: `<<tagLocation>>`, `<<geometricLocation>>` o `<<symbolicLocation>>`. Estos estereotipos se usan sólo en las propiedades de los nodos físicos (clases estereotipadas como `<<physicalNode>>`).

En la Figura 5.4 se puede apreciar el Modelo Navegacional correspondiente al Modelo Unificado que tiene como *Core* el concern Histórico (Figura 5.2). En este Modelo Navegacional se muestra el uso de todos los estereotipos mencionados anteriormente.

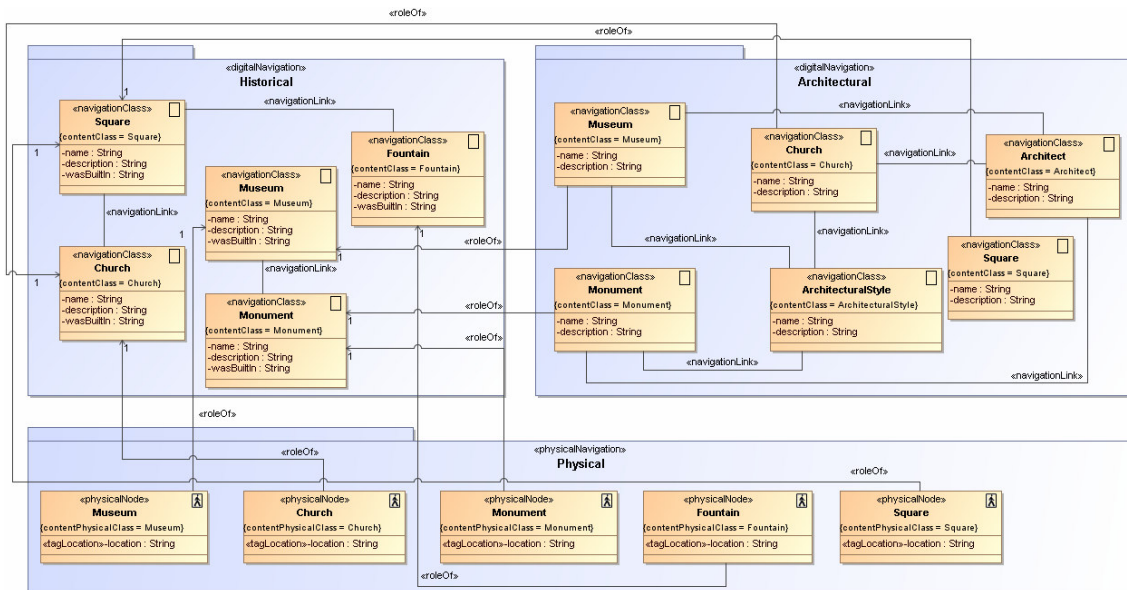


Figura 5.4: Modelo Navegacional para la aplicación del Turista (*Core* Histórico).

Se puede ver cómo cada nodo ya sea digital o físico define las propiedades que contendrá, indicando el nombre de las mismas y su tipo. Para este ejemplo particular (Figura 5.4) cada nodo tiene las mismas propiedades que la/s clase/s correspondiente/s en el Modelo Unificado, manteniendo el nombre y tipo de las mismas.

Observando la Figura 5.4 se puede ver que las clases del concern físico no tienen links caminables predefinidos, esto se debe a que en el Modelo Unificado de la Figura 5.2 no había relaciones entre las clases del concern físico.

Se puede apreciar también (en la Figura 5.4) que las clases agregan un valor

etiquetado denominado `contentClass`²⁸ o `contentPhysicalClass`²⁹, estos valores especifican la clase origen del Modelo Unificado del cual fueron derivadas. El valor etiquetado `contentClass` se asocia al estereotipo `<<navigationClass>>`, mientras que el valor etiquetado `contentPhysicalClass` se asocia al estereotipo `<<physicalNode>>`.

En la Tabla 5.3 se resumen los estereotipos usados en el Modelo Navegacional, no solamente indicando el elemento de *UML* al que se corresponde el estereotipo sino también se detalla si es un estereotipo provisto por *UWE* o es un estereotipo del enfoque.

Tabla 5.3: Estereotipos definidos por el perfil *Mobile UWE* que se usan en el Modelo Navegacional.

Stereotype	UML Element	Provisto
<code>digitalNavigation</code>	Package	Nuestro enfoque
<code>physicalNavigation</code>	Package	Nuestro enfoque
<code>navigationClass</code>	Class	UWE
<code>physicalNode</code>	Class	Nuestro enfoque
<code>navigationLink</code>	Association	UWE
<code>walkingLink</code>	Association	Nuestro enfoque
<code>roleOf</code>	Association	Nuestro enfoque
<code>tagLocation</code>	Property	Nuestro enfoque
<code>geometricLocation</code>	Property	Nuestro enfoque
<code>symbolicLocation</code>	Property	Nuestro enfoque

5.5. Transformación del Modelo Unificado al Modelo Navegacional

En la Sección 3.3.3 se describió una regla de transformación de grafos para obtener un Modelo Navegacional a partir de un Modelo Unificado. En esta sección se detalla cómo cada uno de los elementos del Modelo Unificado se transforma al Modelo Navegacional, en término de los elementos de un diagrama de clases *UML* y los estereotipos del enfoque involucrados.

Para realizar la transformación del Modelo Unificado al Modelo Navegacional se utilizan las siguientes reglas:

- Los paquetes del Modelo Unificado son incluidos en el Modelo Navegacional como paquetes con el mismo nombre. Si el paquete origen (Modelo Unificado) tiene el estereotipo `<<digitalContent>>`, el paquete creado en el Modelo Navegacional especifica el estereotipo `<<digitalNavigation>>`. En el caso de que el paquete origen tenga el estereotipo `<<physicalContent>>` en el Modelo Navegacional se agrega con el estereotipo `<<physicalNavigation>>`.
- Las clases del Modelo Unificado se incluyen en el Modelo Navegacional como clases con el mismo nombre. Para representar que estas clases son nodos en el Modelo Navegacional se le agrega el estereotipo correspondiente según sea un nodo digital o físico. Si la clase origen está contenida en un paquete con el estereotipo `<<digitalContent>>`, la clase creada en el Modelo Navegacional agrega el estereotipo

²⁸ Valor etiquetado definido en *UWE* para poder ser usado para la sincronización de modelos.

²⁹ Valor creado por nuestro enfoque para mantener consistencia con *UWE*.

<<navigationClass>>³⁰. Si la clase origen está contenida en un paquete cuyo estereotipo es <<physicalContent>>, la clase creada (en el Modelo Navegacional) se define con el estereotipo <<physicalNode>>.

Cada una de las clases que se crean en el Modelo Navegacional, tiene las mismas propiedades que la clase origen (del Modelo Unificado), manteniendo los nombres y tipo de las mismas.

Si la clase del Modelo Navegacional se especifica con el estereotipo <<navigationClass>>, se setea el valor etiquetado `contentClass`³¹ con la clase origen del Modelo Unificado. Si la clase del Modelo Navegacional se especifica con el estereotipo <<physicalNode>>, se setea el valor etiquetado `contentPhysicalClass` con la clase origen del Modelo Unificado.

- Las asociaciones entre clases, dentro de un paquete del Modelo Unificado, se incluyen en el Modelo Navegacional como asociaciones (entre las clases correspondientes) con el mismo nombre. Para representar que estas clases especifican links en el Modelo Navegacional hay que definirle el estereotipo correspondiente, indicando si son links digitales o links caminables predefinidos. Si el paquete que contiene la asociación origen (en el Modelo Unificado) tiene el estereotipo <<digitalContent>> a la asociación del Modelo Navegacional se le agrega el estereotipo <<navigationLink>>³⁰. En el caso que la asociación origen del Modelo Unificado esté contenida en un paquete con el estereotipo <<physicalContent>>, a la asociación (del Modelo Navegacional) se le agrega el estereotipo <<walkingLink>>.

Estas asociaciones creadas en el Modelo Navegacional mantienen los nombres y las cardinalidades de las asociaciones origen (del Modelo Unificado).

Las relaciones entre clases de diferentes paquetes del Modelo Unificado se incluyen en el Modelo Navegacional como asociaciones (entre las clases correspondientes). Si la asociación en el Modelo Unificado tiene el estereotipo <<roleOf>>, en el Modelo Navegacional también se le agrega a la asociación creada este estereotipo. En el caso que la asociación (del Modelo Unificado) no tenga el estereotipo <<roleOf>>, se le agrega a la asociación del Modelo Navegacional el estereotipo <<walkingLink>>.

Estas asociaciones creadas (en el Modelo Navegacional) mantienen los nombres y las cardinalidades de las asociaciones del Modelo Unificado.

- En el caso de la propiedad de ubicación, se crea dicha propiedad en la clase correspondiente del Modelo Navegacional, con el estereotipo <<tagLocation>>, <<geometricLocation>> o <<symbolicLocation>> que tiene la propiedad en el Modelo Unificado.

Las reglas anteriormente descritas permiten hacer la transformación del Modelo Unificado al Modelo Navegacional. Para esta transformación no se puede elegir que concerns pasar y cuales no, se transforman todos los concerns del Modelo Unificado. Es decir, el filtrado de los concerns de interés se realiza cuando se hace la transformación del Modelo de Contenido al Modelo Unificado, pero no se puede hacer la elección de qué concerns pasar al Modelo Navegacional cuando se realiza esta transformación.

Una vez realizada la transformación correspondiente, este modelo puede ser modificado por el diseñador. Es decir, se puede sacar y agregar elementos ya sean

³⁰ Estereotipo definido por *UWE*.

³¹ Valor etiquetado definido en *UWE*.

clases, propiedades o asociaciones. De esta manera se logra refinar el Modelo Navegacional para ajustarlo a las necesidades de cada aplicación, un ejemplo de esto se puede ver en la Figura 5.5.

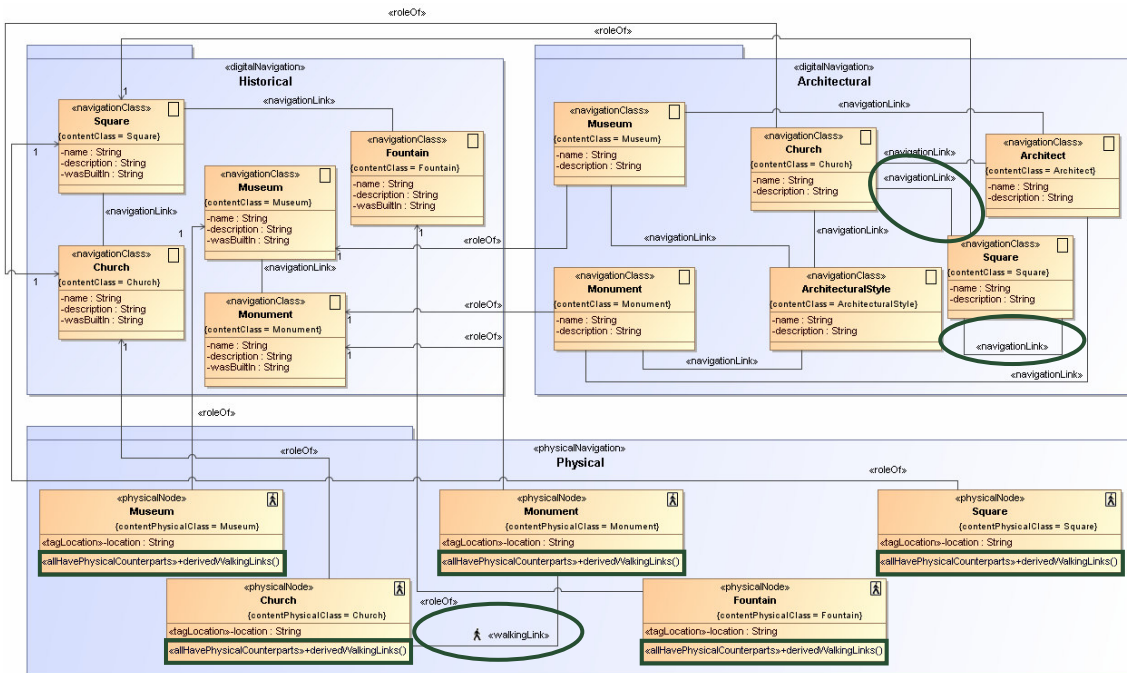


Figura 5.5: Modelo Navegacional refinado.

En la etapa de refinamiento se pueden, por ejemplo, agregar links caminables predefinidos como muestra la Figura 5.5 entre el nodo físico *Church* y *Monument*. Se puede observar que también se agregaron dos links digitales dentro del concern arquitectural, uno del nodo *Church* al nodo *Square* y otro del nodo *Square* al nodo *Square*.

En la etapa de refinamiento es donde se define la operación que calcula los links caminables derivados si los hubiera. Como se puede apreciar en la Figura 5.5, todos los nodos físicos definen la operación *derivedWalkingLinks*, la cual está estereotipada como `<<allHavePhysicalCounterparts>>`. De esta manera se está indicando cómo se calculan los links caminables derivados de cada nodo físico.

Puede haber nodos físicos que no definan una operación para calcular los links caminables derivados, en dicho caso cuando el usuario sea sentido por el objeto del mundo real (que se corresponde con este nodo físico), no se le deriva ningún link caminable. También puede pasar que, dentro de un Modelo Navegacional, el diseñador decida que algunos nodos físicos tengan definida la operación de links caminables derivados estereotipada como `<<allHavePhysicalCounterparts>>` o como `<<onlyNearPhysicalCounterparts>>`.

De esta manera el usuario refina el Modelo Navegacional para que se ajuste a las necesidades de la aplicación que está modelando.

En la Tabla 5.4 se presentan los estereotipos relacionados a la etapa de refinamiento, donde se definen las operaciones de links caminables derivados para cada nodo físico.

Tabla 5.4: Estereotipos adicionales incorporados al perfil *Mobile UWE* que se usan en el Modelo Navegacional.

Stereotype	UML Element	Provisto
<code>allHavePhysicalCounterparts</code>	Operation	Nuestro enfoque
<code>onlyNearPhysicalCounterparts</code>	Operation	Nuestro enfoque

5.6. Instanciación del Modelo Navegacional

En este modelo se definen las instancias particulares de un Modelo Navegacional que actúa como base de la instanciación. Los elementos de este modelo son objetos, los cuales son instancias (objetos) de los nodos que aparecen en el Modelo Navegacional. Las asociaciones entre estos objetos, están determinadas acordes a los links definidos en el Modelo Navegacional.

En el Modelo de Instancias Navegacional se representan los mismos concerns que aparecen en el Modelo Navegacional de base. Dentro de cada concern se crean instancias de los nodos y se completan las propiedades asociadas a cada nodo, con valores particulares según corresponda.

Dentro de cada concern del Modelo de Instancias Navegacional, solamente se pueden instanciar aquellos nodos que aparecen en el concern correspondiente en el Modelo Navegacional. Un mismo nodo se puede instanciar las veces que sea necesario según la aplicación que se esté modelando. También puede pasar que para un mismo Modelo Navegacional el diseñador cree distintos Modelos de Instancias Navegacional, donde para cada uno provea distinta información.

En este modelo, las instancias creadas tienen los estereotipos correspondientes al nodo que instancian, ya sea `<<navigationClass>>` o `<<physicalNode>>`. Lo mismo ocurre con las asociaciones, van a tener el estereotipo definido en el Modelo Navegacional, es decir `<<navigationLink>>`, `<<walkingLink>>` o `<<roleOf>>`.

Lo mismo ocurre a nivel de concerns, estos van a tener el estereotipo `<<digitalNavigation>>` o `<<physicalNavigation>>` según corresponda y además tienen el mismo nombre que aparece en el Modelo Navegacional.

Cada objeto instanciado debe tener un nombre determinado y la clase de nodo que representa.

En el Modelo de Instancias Navegacional, no surgen estereotipos nuevos ya que se usa el Modelo Navegacional como base y al instanciar los elementos del mismo estos ya tienen sus estereotipos definidos. En la Figura 5.6 se puede apreciar un Modelo de Instancias Navegacional basado en el ejemplo presentado en la Sección 3.2.

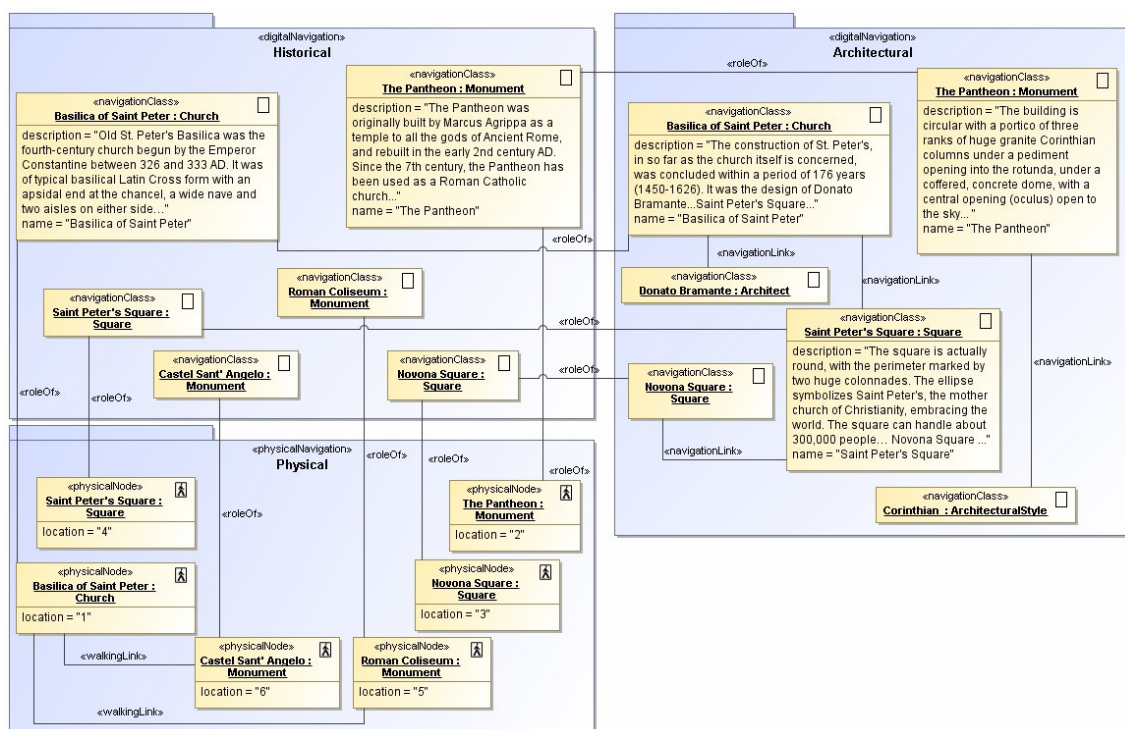


Figura 5.6: Modelo de Instancia Navegacional.

Se puede apreciar en la Figura 5.6 que se instancia *Church* con los datos de la *Basílica de San Pedro* en los tres concerns (histórico, arquitectural y físico). En cuanto a *Square* es instanciado también en los tres concern con los datos de *La Plaza de San Pedro* y *La Plaza Novona*. Por otro lado, *Monument* se instancia con *Castillo San Ángel*, el *Coliseo Romano* y el *Panteón*. Para completar el ejemplo presentado en la Sección 3.2 se instancia *Architect* con el arquitecto *Donato Bramante* y *ArchitecturalStyle* con *Corintio*. Por simplicidad, sólo se completaron algunos valores de sus atributos de las instancias.

Se puede apreciar en la Figura 5.6 la asociación entre las instancias *Core* y el resto de las instancias que aparecen en los otros concerns. Además la definición de los links digitales y links caminables (predefinidos) se corresponden con los visualizados en el ejemplo presentado en la Sección 3.2. En cuanto a la propiedad de ubicación como en el Modelo Navegacional estaban todas definidas como `<<tagLocation>>`, a nivel de instancias se le especificó un número como valor de esta propiedad. Dicho valor especifica el valor con el cual se etiqueta dicho objeto del mundo real, para luego, por ejemplo, ser recuperado mediante un código de barras.

Veamos cómo es la definición de una operación particular para derivar los links caminables. Como se menciona en el refinamiento del Modelo Navegacional se podía definir la operación `derivedWalkingLinks` con distintos estereotipos. Para el ejemplo que venimos presentando (Figura 5.6), el Modelo Navegacional de base define esta operación estereotipada como `<<allHavePhysicalCounterparts>>` (Figura 5.5). Esto quiere decir que la operación se calcula considerando los siguientes parámetros:

- `core`, representa al concern *Core* de la aplicación.
- `concern`, es el concern digital actual que está visualizando el usuario.
- `currentNode`, es el nodo digital actual que está visualizando el usuario.

Una operación definida con este estereotipo (`<<allHavePhysicalCounterparts>>`) realiza el siguiente cálculo: si el `currentNode` tiene links digitales asociados y el target de estos tienen su contraparte física, entonces se puede derivar un link caminable a esa contraparte física.

Cabe aclarar que la operación para derivar links caminables, se ejecuta en forma dinámica para cada usuario, acorde a qué está visualizando digitalmente en un momento dato. Cuando cambia por algún motivo el contenido digital, esta operación se debe volver a ejecutar. Los resultados de ejecutar esta operación se ven recién cuando el usuario utiliza la aplicación, a nivel de modelo sólo se la puede definir en los nodos físicos.

Analicemos cómo visualiza el usuario los links caminables derivados si la aplicación tiene un Modelo de Instancias Navegacional como el que se presentó en la Figura 5.6. Para focalizarnos en como funciona la operación de los links caminables derivados, se va a recortar el Modelo de Instancias Navegacional para que sólo muestre la información necesaria. En la Figura 5.7 se puede apreciar como el usuario visualiza el link caminable derivado que se genera desde el concern arquitectural de la *Basílica de San Pedro*. Veamos cómo funciona la operación `derivedWalkingLinks`, los valores que toman los parámetros de esta función son:

- `core` = el concern histórico
- `concern` = el concern arquitectural
- `currentNode` = nodo de la *Basílica de San Pedro* (concern arquitectural)

El nodo *Basílica de San Pedro* (`currentNode`) tiene dos links, uno a *Donato Bramante* y otro a la *Plaza de San Pedro*. El nodo *Donato Bramante* no tiene contraparte física por lo tanto no se puede derivar un link caminable. Pero la *Plaza de San Pedro* si tiene su contraparte física entonces se puede crear un link caminable derivado hacia este

lugar. El resultado que emite la operación `derivedWalkingLinks` para este caso es un link caminable derivado a la *Plaza de San Pedro*. Esta situación el usuario la visualiza mediante la opción *Suggestions* en su vista con información física.

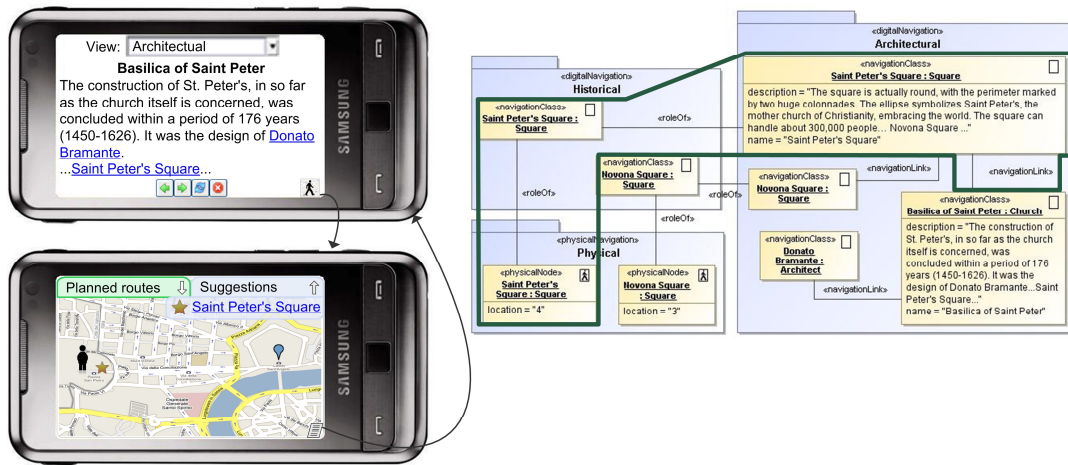


Figura 5.7: Link caminable derivado desde el nodo de la *Basilica de San Pedro*.

Se debe controlar que el nodo físico, donde se encuentra parado el usuario, no tenga un link caminable prefijado cuyo destino sea alguno de los links caminables derivados. En este caso el link caminable derivado no es incluido como *Suggestions*, ya que el usuario tiene esta alternativa como *Planned Routes*. Además, se debe controlar que el link caminable no tenga como target el objeto físico donde se encuentra ubicado el usuario, ya que no tiene sentido brindarle un link caminable al lugar donde se encuentra ubicado actualmente.

Veamos ahora que pasa cuando el usuario (visualizando la información digital de la Figura 5.7) elige el link digital a la *Plaza de San Pedro*, no sólo se le actualiza la pantalla con información arquitectural de dicho nodo sino que además se deben recalculer los links caminables derivados. Es decir, se vuelve a calcular la operación `derivedWalkingLinks` donde el parámetro que varía es `currentNode`, tomando como valor el nodo de la *Plaza de San Pedro* del concern arquitectural. Este nodo tiene un sólo link digital al nodo de la *Plaza Novona*, el cual posee contraparte física. Por lo tanto, se puede derivar un link caminable, y éste es el resultado que devuelve la operación `derivedWalkingLinks`. En la Figura 5.8 se puede visualizar como se le actualizaron los links caminables derivados listados como *Suggestions*.

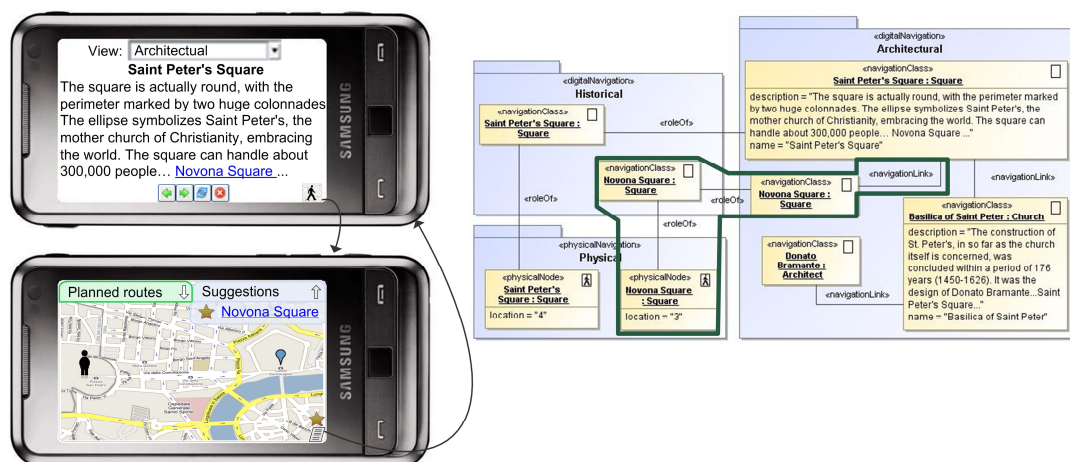


Figura 5.8: Ejemplo de Link caminable derivado desde el concern arquitectural de la *Plaza de San Pedro*.

Cada vez que el usuario navega digitalmente se ejecuta la operación `derivedWalkingLinks` con los parámetros actualizados, determinando así los nuevos links caminables derivados acorde al nodo digital al cual se navegó. Esta operación también se ejecuta cuando el usuario cambia de concern digital o cuando es sentido por un Punto de Interés.

5.7. Modelo de Presentación

Hasta el momento los modelos que se venían presentando no involucraban ninguna relación con la forma de implementarlos. Sin embargo el Modelo de Presentación está ligado íntegramente con la implementación final que tenga la aplicación. Por esta razón, pueden existir varios Modelos de Presentación asociados a un mismo Modelo Navegacional. En rasgos generales este modelo representa la visualización que tiene cada nodo del Modelo Navegacional.

Para esta tesis se presenta un Modelo de Presentación particular a fin de mostrar el enfoque en forma completa. Se necesita representar el concepto de concerns (digitales y físico) los cuales contengan presentaciones de nodos. Para esto se debe tener un estereotipo particular indicando si se agrupan presentaciones de nodos digitales o físicos. Por lo tanto, creamos dos estereotipos de *Package* [UML Superstructure] uno denominado `<<digitalLayout>>` y otro `<<physicalLayout>>` como se puede visualizar en la Figura 5.9.

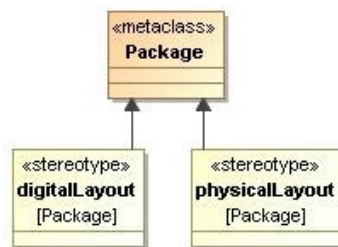


Figura 5.9: Estereotipos para los concerns del Modelo de Presentación.

Con estos dos estereotipos (`<<digitalLayout>>` y `<<physicalLayout>>`) se pueden representar los concerns necesarios para la visualización. Cada nodo va a tener definido una presentación particular: para poder representar esta semántica se crearon dos estereotipos de clases, un estereotipo `<<navigationClassLayout>>` para representar la presentación de los nodos digitales, y otro estereotipo `<<physicalNodeLayout>>` para representar la presentación de los nodos físicos, como se puede visualizar en la Figura 5.10.

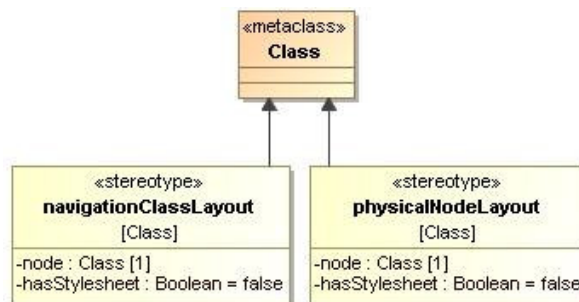


Figura 5.10: Estereotipos para las clases del Modelo de Presentación.

Estos dos estereotipos (`<<navigationClassLayout>>` y `<<physicalNodeLayout>>`) tienen definidos dos valores etiquetados. Uno de los valores etiquetados se denomina `node`, el cual se utiliza para definir el nodo al que se le está definiendo la presentación.

El otro valor etiquetado asociado a estos estereotipos, se denomina `hasStyleSheet`, el cual representa un valor `boolean` indicando si se definió o no un estilo de presentación asociado al nodo. Inicialmente, cuando se define la presentación asociada al nodo el valor etiquetado (`hasStyleSheet`) es falso y cuando se define una presentación determinada este valor pasa a ser verdadero.

En la Figura 5.11 se puede ver un Modelo de Presentación donde sólo se detalló la clase de presentación que se corresponde a cada nodo navegacional (presentado en la Figura 5.5). Se puede ver que el valor etiquetado `node` de todas las clases tiene un valor definido, mientras que el valor etiquetado `hasStyleSheet` tiene para todas las clases valor falso, ya que las clases todavía no tienen un estilo definido. También se puede observar en este Modelo de Presentación, el uso de los estereotipos definidos tanto a nivel de concerns (paquetes) como a nivel de presentación de nodo (clases). Se puede apreciar en la Figura 5.11 que cada nodo tiene su presentación asociada, esto permite que, si un Punto de Interés está representado en más de un concern digital, para cada concern pueda tener una visualización diferente del mismo. De esta manera si un nodo tiene propiedades particulares en el Modelo de Presentación, se expresa cómo se debe visualizar cada una.

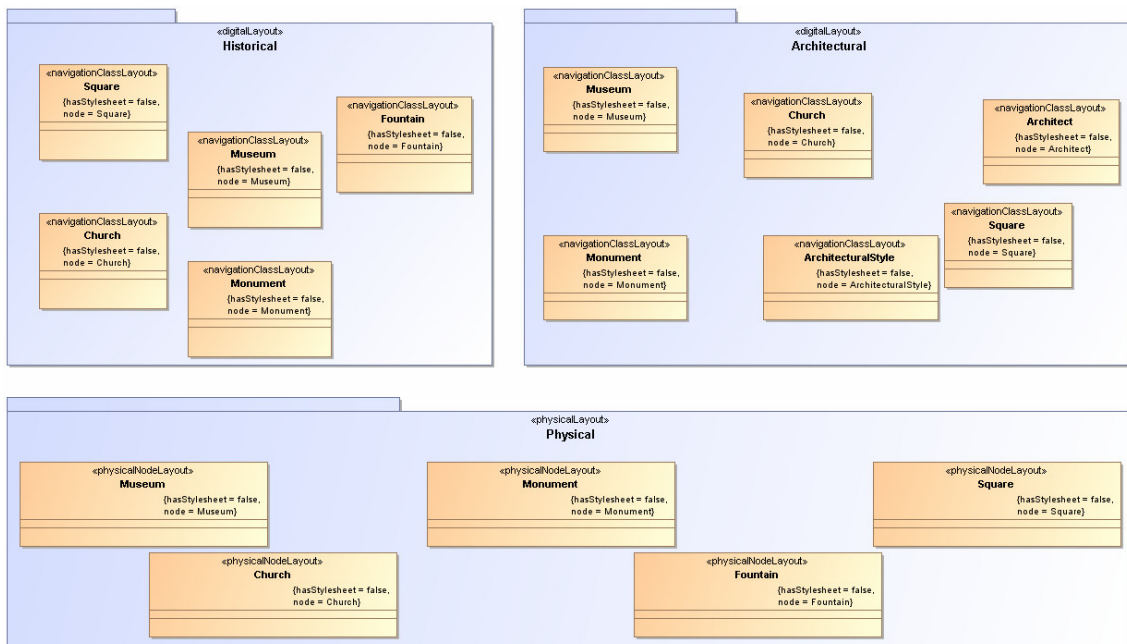


Figura 5.11: Modelo de Presentación.

Es muy probable que en la mayoría de las aplicaciones, la visualización de los nodos digitales vaya a diferir de los nodos físicos. Hay que tener en cuenta que todas las instancias de un mismo nodo se van a mostrar de la misma manera, ya que la presentación está asociada al nodo. Si se quieren lograr presentaciones distintas de un mismo nodo, se debe definir distintos Modelos de Presentación.

Para el Modelo de Presentación particular especificado en esta tesis se eligió definir la presentación asociada a cada nodo mediante un archivo *XSLT*³² (*Extensible Stylesheet Language Transformation*) que define cómo se muestran los datos. Es decir, a cada clase de presentación se le asocia un archivo *XSLT*³³ que define cómo se muestra el nodo.

³² Página de la W3C que define XSLT: <http://www.w3.org/TR/xslt>.

³³ Asociada a cada archivo XSLT se tiene un archivo DTD (*Document Type Definition*), el cual define la estructura que tienen los elementos que toma el template como parámetro.

*XSLT es un estándar definido por la W3C (World Wide Web Consortium)*³⁴ que se puede ver como una hoja de estilo. Tiene parámetros que se deben de ingresar para luego derivar la salida (presentación de los datos ingresados como parámetros).

En nuestro caso, al *XSLT* se le pasan como parámetro los datos del nodo para generar como salida la visualización de los mismos, según como esté definida la estructura interna del archivo *XSLT*.

La clase de presentación debe indicar para cada uno de los parámetros del *XSLT*, con qué valor del nodo se debe completar. De esta manera utilizando esta especificación del Modelo de Presentación se puede mostrar cada nodo según cómo se haya especificado el archivo *XSLT* correspondiente.

A modo de ejemplificar la presentación de un nodo usando *XSLT* nos focalizamos en un nodo particular del modelo navegación, por ejemplo, el nodo *Church* del concern arquitectural.

Supongamos que se define un archivo *XSLT* que tiene tres parámetros nombrados como *nodeName*, *descriptionNode* y *link*. Estos tres parámetros deben de ser definidos como propiedades de la clase de presentación del nodo como se muestra en la Figura 5.12. Para cada una de estas propiedades, se debe especificar un *String* que representa el nombre de la propiedad del nodo de donde se toman el dato para “enviar” como parámetro.

En la figura se puede ver cómo el parámetro *nodeName* se completa con el valor que tenga definido el nodo en la propiedad *name*; por otro lado *descriptionNode* se especifica con el valor que tenga el nodo en la propiedad *description*. El último parámetro es *link*, el cual toma como valor todas las relaciones definidas con el estereotipo `<<navigationLink>>`. Las definiciones de las clases de presentación se realizan todas de esta manera.

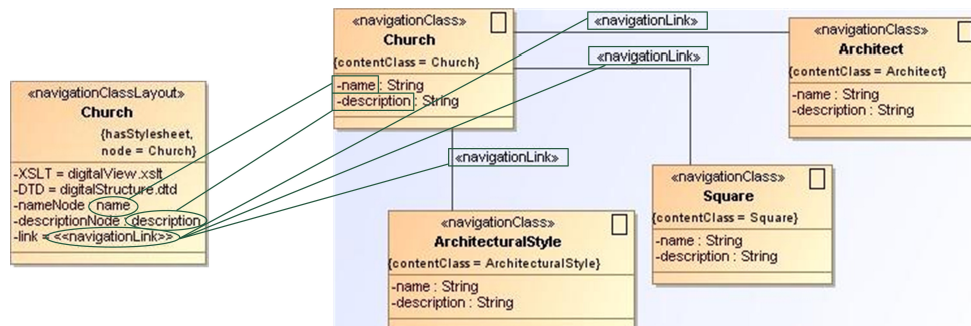


Figura 5.12: Relación entre la presentación y el nodo *Church* del concern arquitectural.

Si un nodo tiene más propiedades que otros nodos, esto está contemplado por la clase de presentación, la cual tendrá un archivo *XSLT* con la cantidad de parámetros adecuados. Lo mismo pasa a nivel de nombre de propiedades, como la clase de presentación define para cada parámetro con qué propiedad del nodo se debe completar, esto permite dinamismo a nivel de la presentación asociada a cada nodo. Permitiendo variar y adecuar la presentación a las características de cada nodo.

En la Figura 5.13 se puede apreciar la clase de presentación asociada al nodo *Church* del concern histórico. Comparando las Figuras 5.12 y 5.13 se puede distinguir que la clase de presentación de la Figura 5.13 tiene que definir un parámetro más respecto del nodo *Church* del concern arquitectural, ya que el mismo contiene más información. También se puede apreciar que cambia el *XSLT* asociado a la clase de presentación.

³⁴ Página de la W3C: <http://www.w3.org>.

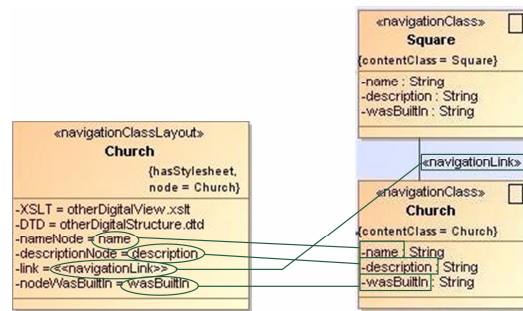


Figura 5.13: Relación entre la presentación y el nodo *Church* del concern histórico.

El archivo *XSLT* va a definir cómo se va a mostrar la información asociada al nodo ya sea a nivel de propiedades como de links. El *XSLT* asociado puede ser tan complejo como sea necesario según la aplicación que se este modelando. Para los nodos físicos, el diseñador debe especificar un *XSLT* que considere, por ejemplo, la visualización de los links caminables predefinidos como así también los links caminables derivados (siempre que el nodo tenga definida la operación que los calcula).

Los estereotipos definidos para este modelo son agregados a un nuevo perfil denominado *Layout* (ver Figura 5.14), de esta manera este perfil puede extenderse de manera independiente del resto de los perfiles que se presentaron en la Sección 4.2.

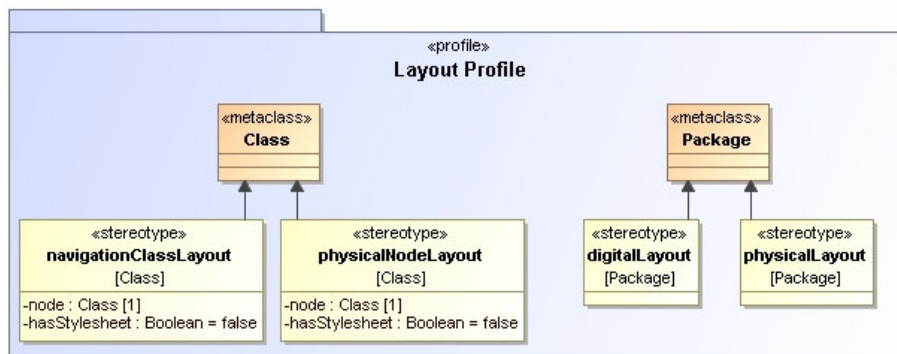


Figura 5.14: Perfil *Layout* y los estereotipos que contiene.

El nuevo perfil *Layout* es importado por el perfil *Mobile Hypermedia* para lograr un único perfil, que contenga los elementos necesarios para poder crear todos los modelos del enfoque. Con esta importación el perfil *Mobile Hypermedia* queda conformado como se muestra en la Figura 5.15.

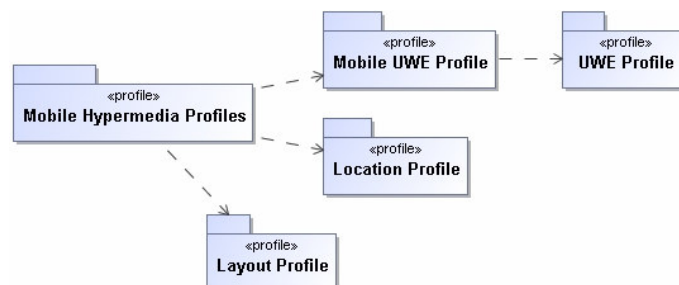


Figura 5.15: Perfil *Mobile Hypermedia* con la importación del perfil *Layout*.

El Modelo de Presentación detallado en esta sección es una posible implementación, pueden surgir otros tipos de presentación en futuras evoluciones de nuestro enfoque. Es decir, el perfil de *Layout* puede variar o volverse más complejo, tanto como sea

necesario. Mientras el nombre del perfil (*Layout*) se mantenga, esta evolución no afecta la importación realizada por el perfil *Mobile Hypermedia*.

5.8. Modelo del Usuario

El Modelo del Usuario debe contener la información necesaria para registrar el estado particular de un usuario en la aplicación. Este modelo puede ser tan complejo como lo requiera la aplicación que se está modelando, teniendo en cuenta el perfil del usuario, cuestiones de contexto, etc. En el enfoque se muestra un posible Modelo del Usuario que se presenta en la Figura 5.16. Este modelo está simplificado para los conceptos básicos del mismo, en la Sección 6.7 se amplía dicho modelo, incorporando las políticas de navegación relacionadas a la hipermedia para completar los mínimos elementos que debe tener un Modelo del Usuario. Este modelo fue creado a partir de la base de trabajos previos como [Challiol et al., 2007] y [Challiol et al., 2008].

En la parte superior de la Figura 5.16, se especifican las clases que se instancian cuando el usuario se registra a la aplicación y comienza a utilizarla. Se define el usuario (*User*) como una clase, la cual tiene asociadas varias aplicaciones (*UserApplicationInformation*). La clase *UserApplicationInformation* registra para cada una de las aplicaciones (*MobileHypermediaApplication*) el historial de nodos digitales y físicos por los que va pasando el usuario (*PhysicalNavigationHistory* y *DigitalNavigationHistory*). Estos historiales tienen los nodos por los que el usuario navegó digitalmente o en el mundo real. En particular cada historial tiene identificado cuál es el nodo digital y físico actual del usuario.

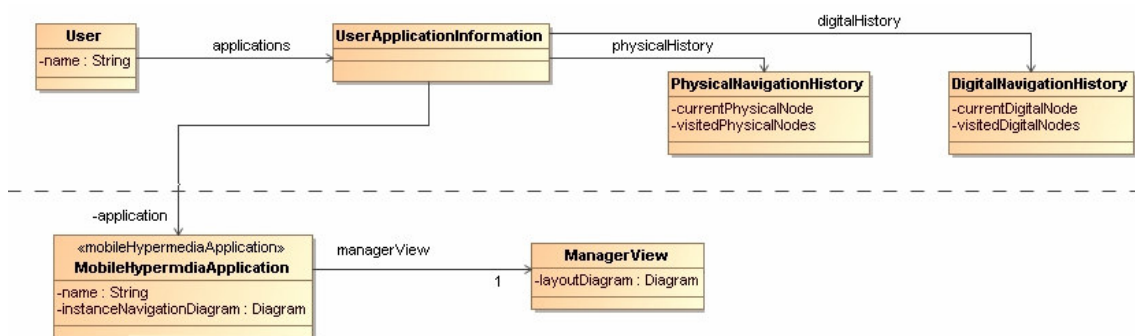


Figura 5.16: Modelo del Usuario.

En la parte inferior de la Figura 5.16 se pueden apreciar dos clases. *MobileHypermediaApplication* representa la aplicación móvil en particular la cual tiene un nombre asociado. Este nombre es usado para mostrarle al usuario qué aplicaciones tiene disponibles en su dispositivo. Esta clase conoce cuál es el Modelo de Instancias Navegacional que visualiza el usuario y del cual se toma la información para mostrarle. Es decir, el modelo de instancias de nodos navegacionales por los que se mueve (digitalmente o en el mundo real). La clase *MobileHypermediaApplication* además conoce cómo visualizará el usuario las instancias del Modelo Navegacional, esto se hace mediante la clase *ManagerView* que conoce un Modelo de Presentación acorde al Modelo Navegacional instanciado. Tanto las clases *MobileHypermediaApplication* como *ManagerView* contienen información que puede ser definida por el diseñador, ya que éste puede decidir cuál van a ser el Modelo de Instancias Navegacional y con qué presentación se muestra el mismo. Se puede apreciar en la Figura 5.16 que la clase *MobileHypermediaApplication* tiene indicado el estereotipo `<<mobileHypermediaApplication>>`. Este estereotipo permite establecer que esa clase representa una aplicación de HM. La definición del mismo se puede ver en la Figura 5.17, y sirve para especificar que una clase representa una aplicación o es el punto de entrada de una aplicación.

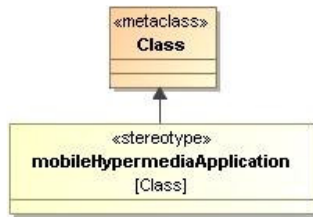


Figura 5.17: Estereotipo para representar a las aplicaciones de HM.

En la Figura 5.18 se puede observar una posible instanciación de las clases *MobileHypermediaApplication* y *ManagerView*. Se puede apreciar que el Modelo de Instancias Navegacional asociado a la instancia de la clase *MobileHypermediaApplication*, se denomina “*Mobile UWE Navigation Instance Diagram*” mientras que el Modelo de Presentación asociado a la instancia de la clase *ManagerView*, se denomina “*Mobile UWE Layout Diagram*”. Es decir, que se indica cuál es el Modelo de Instancias Navegacional del Modelo Navegacional que puede recorrer el usuario y cómo se visualiza cada nodo. También se puede apreciar que el nombre especificado para la aplicación es “*Tourist Application*”, este nombre es el que visualiza el usuario cuando se le listan en su dispositivo todas las aplicaciones en las cuales se puede registrar. Una vez registrado (el usuario) a una aplicación puede navegar tanto digitalmente, como por el mundo real por las instancias del modelo especificado.

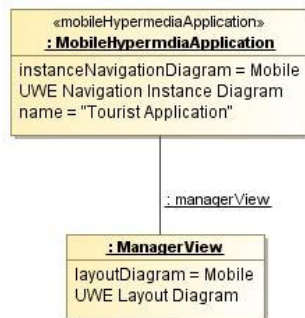


Figura 5.18: Instanciación del Modelo del Usuario.

Tanto el modelo de clases presentado en la Figura 5.16, como la definición del estereotipo <<mobileHypermediaApplication>> (Figura 5.17), están contenidos en un nuevo perfil denominado *User Model* como se puede apreciar en la Figura 5.19. De esta manera, se logra que los datos referentes al Modelo del Usuario queden agrupados en un perfil específico, pudiendo evolucionar de forma independiente a los perfiles ya presentados.

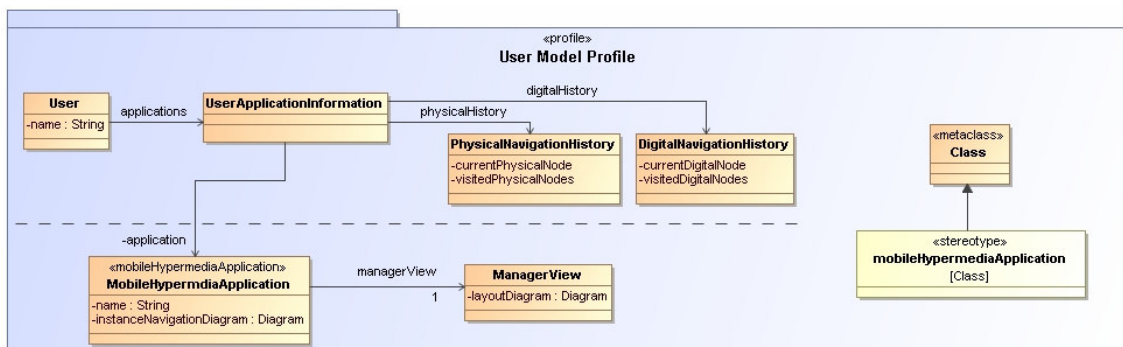


Figura 5.19: Perfil *User Model* con sus elementos.

El nuevo perfil *User Model* es importado por el perfil *Mobile Hypermedia* como se

muestra en la Figura 5.20. Así, se logra un único perfil (*Mobile Hypermedia*) que contiene los elementos necesarios para poder crear todos los modelos del enfoque.

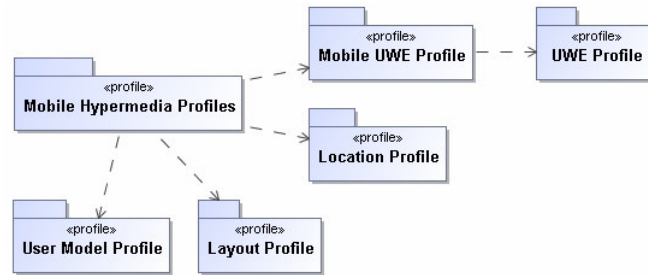


Figura 5.20: Perfil *Mobile Hypermedia* con la importación del perfil *User Model*.

Como se menciona anteriormente, el Modelo del Usuario detallado en esta sección es una posible implementación, puede surgir un nivel de detalle mayor según las características que tenga la aplicación. Es decir, el perfil del Modelo del Usuario puede volverse más complejo. Siempre que el nombre del perfil del usuario (*User Model*) se mantenga, cualquier modificación interna que se haga en el mismo no afecta la importación realizada por el perfil *Mobile Hypermedia*.

5.9. Resumen

En este capítulo se describieron en detalles todos los modelos del enfoque propuesto. Se presentaron aquellos modelos independientes de la plataforma (*PIM*) como son el Modelo de Contenido, el Modelo Unificado y el Modelo Navegacional. Se describieron cómo se realizan las dos transformaciones $PIM \rightarrow PIM$, una del Modelo de Contenido al Modelo Unificado y otra del Modelo Unificado al Modelo Navegacional. Además, se presentó una posible implementación del Modelo de Presentación y otra del Modelo del Usuario, ambos modelos son dependientes de la plataforma (*PSM*). A modo de resumen, se presenta en la Figura 5.21 todos los elementos introducidos en este capítulo en los perfiles *Layout* y *User Model*.

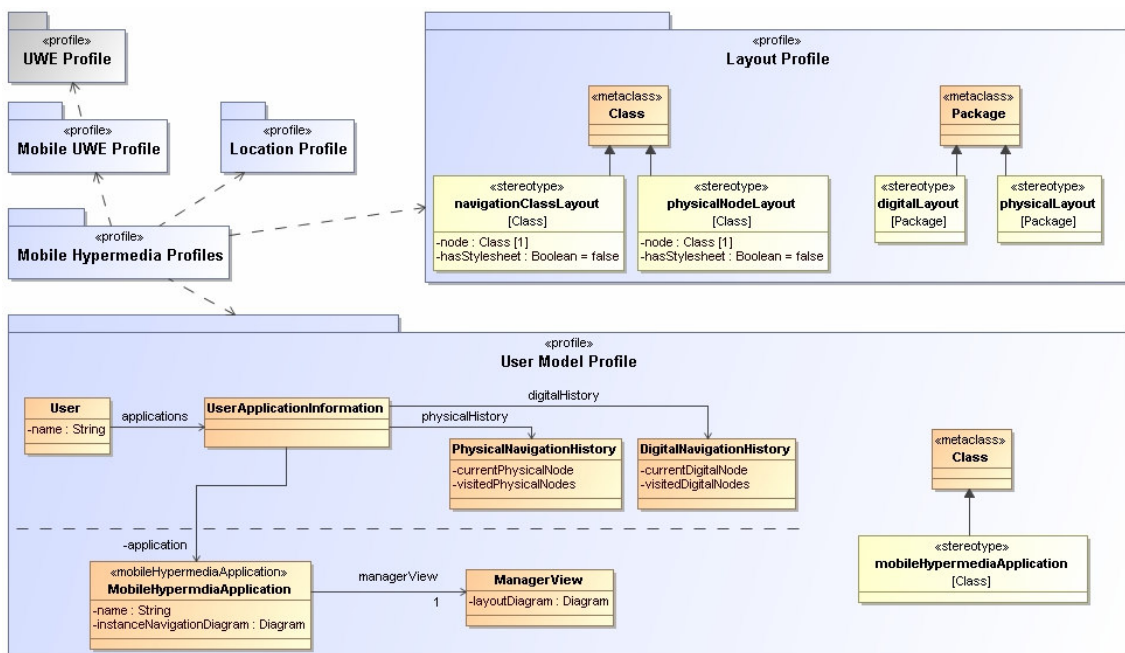


Figura 5.21: Perfiles destacando los elementos de los perfiles *Layout* y *User Model*.

Cabe destacar que el perfil *Mobile Hypermedia* sirve simplemente para importar todos

los demás perfiles (*Mobile UWE, Location, Layout y User Model*), y de esta manera agrupar todos los perfiles. El perfil *Mobile Hypermedia* no agrega ningún elemento nuevo. Cuando un diseñador modela una aplicación de HM usando el enfoque la misma tiene que usar el perfil *Mobile Hypermedia* para la definición de cada uno de los elementos de nuestros modelos.

En la Tabla 5.5 se muestra un resumen del nombre de cada uno de los estereotipos definidos en este capítulo, junto al elemento *UML* al cual se le define el estereotipo. En la tercera columna se detalla si se el estereotipo se corresponde al Modelo de Presentación o al Modelo del Usuario.

Tabla 5.5: Estereotipos definidos por el perfil *Layout y User Model* para representar los conceptos de nuestros Modelos de Presentación y del Usuario.

Estereotipo	Elemento de UML	Modelos en los que se usa el estereotipo
digitalLayout	Package	Modelo de Presentación
physicalLayout	Package	Modelo de Presentación
navigationClassLayout	Class	Modelo de Presentación
physcialNodeLayout	Class	Modelo de Presentación
mobileHypermediaApplication	Class	Modelo del Usuario

En la Figura 5.22³⁵ se muestra un esquema general de los niveles de modelado relacionados con el desarrollo dirigido por modelos.

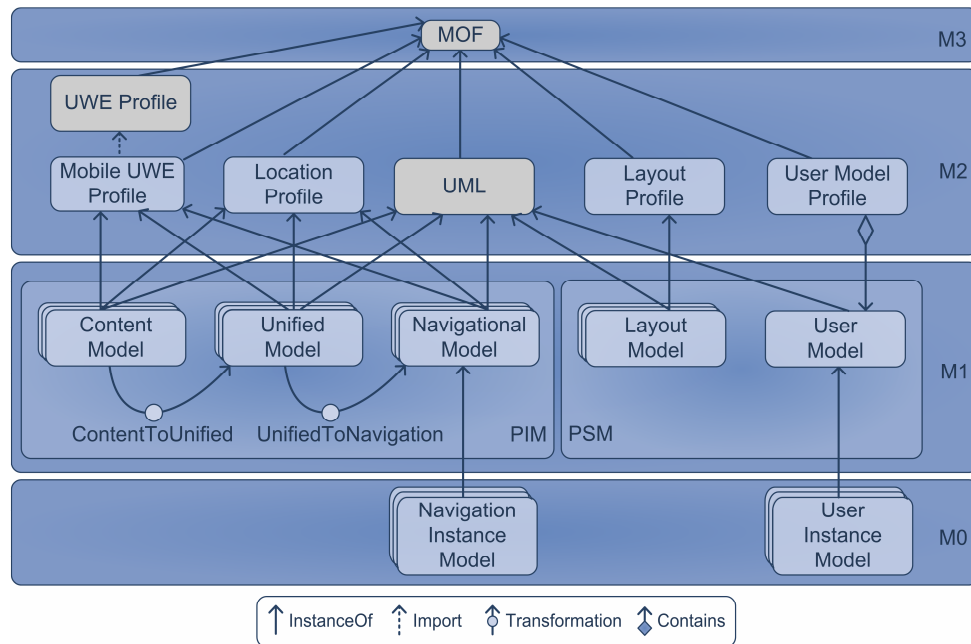


Figura 5.22: Nuestros modelos, transformaciones y perfiles según los niveles de modelado del desarrollo dirigido por modelos.

Se puede observar en la Figura 5.22, que en el nivel *M2* se ubican todos nuestros perfiles y *UML*. En el nivel *M1* están nuestros modelos y se indican cuáles son *PIM* y cuáles *PSM*, además en este nivel se especifican las transformaciones (*ContentToUnified* y *UnifiedToNavigation*). En el nivel *M0* se puede apreciar el Modelo

³⁵ No se incluyó en la figura el perfil *Mobile Hypermedia* para mostrar claramente que perfil utiliza cada modelo, porque en el caso de incluirlo se visualizaría que todos usan este perfil y no se notaría la diferencia.

de Instancias Navegacional, el cual representa información particular de una aplicación concreta.

Además, se puede visualizar en la Figura 5.22, que el perfil *User Model* contiene el Modelo del Usuario propuesto (el cual se ubica en un nivel *M1*). La instanciación del Modelo del Usuario se encuentra en el nivel *M0*, es en este modelo donde se define cuál es el Modelo de Instancias Navegacional y el Modelo de Presentación para una aplicación particular.

6. MAGICMOBILEUWE: HERRAMIENTA PARA HIPERMEDIA MÓVIL

En este capítulo se muestra la creación de la herramienta que permite el desarrollo dirigido por modelos descrito en los capítulos anteriores. Se describen los conceptos básicos de esta herramienta y cómo la misma puede ser usada para crear cada uno de los modelos de nuestro enfoque.

Se hace hincapié en mostrar cómo el desarrollador (diseñador) puede usar la herramienta para agilizar su tarea de crear aplicaciones de HM, como por ejemplo usar las transformaciones entre modelos o funcionalidad contextual (propia de cada modelo o de cada tipo de elemento).

6.1. Características básicas de *MagicMobileUWE*

En el Capítulo 4 se justificó el uso de *UWE* como metodología base para extenderla con los conceptos propios de las aplicaciones de HM, mostrando que elementos se deben definir para poder tener representados aquellos conceptos que son propios de este tipo de aplicaciones.

Como nuestro enfoque extiende *UWE*, fue conveniente crear nuestra herramienta en base a una herramienta existente de *UWE*, permitiéndonos tener los elementos de *UWE* ya definidos.

Actualmente *UWE* cuenta con varias herramientas de modelado, una de ellas es un plugin llamado *MagicUWE* que permite la creación de sus modelos y elementos. La herramienta de base de este plugin se llama *MagicDraw*³⁶ (herramienta de modelado *UML*). El plugin *MagicUWE* agrega a *MagicDraw* funcionalidad propia para el desarrollo dirigido por modelos para aplicaciones de Hipermedia.

Además, *UWE* cuenta con la herramienta *UWE4JSF*³⁷ que permite el desarrollo dirigido por modelos, para la creación de aplicaciones de Hipermedia basadas específicamente en la plataforma *JSF* (*JavaServer Faces*).

Otra herramienta disponible es *UWLet*³⁸ que es una extensión de la herramienta de modelado *UML* llamada *UMLet*³⁹, la cual tiene una interfaz muy simple y todavía no soporta la versión *UML 2.0*⁴⁰.

Una ventaja que tiene *MagicDraw*, respecto de *UWE4JSF* y *UWLet*, es que cuenta con una *API* abierta, la cual permite crear plugins de una manera sencilla, permitiendo extender esta herramienta con nueva funcionalidad acorde a las necesidades de cada dominio. La *API* abierta permite acceder a los metamodelos de *UML* como así también incorporar a *MagicDraw* nuevos menús y elementos en las barras de herramientas. Además, *MagicDraw* es una herramienta que está en constante crecimiento y actualmente soporta la versión *UML 2.0*.

Se eligió *MagicUWE*⁴¹ como herramienta por sobre *UWE4JSF* y *UWLet*, porque, por un lado no limita a trabajar sólo con la plataforma *JSF* (como si lo hace *UWE4JSF*), y por otro lado trabaja con la versión *UML 2.0* (ya que *UWLet* no soporta esta versión). Además, *MagicDraw* cuenta con una *API* abierta que facilita su extensión.

Nosotros creamos una extensión (plugin) para *MagicDraw* que permite ampliar la funcionalidad de esta herramienta para soportar desarrollo dirigido por modelos para

³⁶ Página de *MagicDraw*: <http://www.magicdraw.com/>

³⁷ Página de *UWE4JSF*: <http://uwe.pst.ifi.lmu.de/toolUWE4JSF.html>

³⁸ Página de *UWLet*: <http://uwe.pst.ifi.lmu.de/toolUWEet.html>

³⁹ Página de *UMLet*: <http://www.umlet.com/>

⁴⁰ Pagina con información respecto a *UML 2.0*: <http://www.uml.org/>

⁴¹ Página de *MagicUWE*: <http://uwe.pst.ifi.lmu.de/toolMagicUWE.html>

aplicaciones de HM. La extensión se denomina *MagicMobileUWE* (ver Anexo A) y puede funcionar simultáneamente con el plugin *MagicUWE*, permitiendo usar los elementos de *UWE* en nuestros modelos sin inconvenientes.

Nuestra extensión *MagicMobileUWE*, permite crear los modelos de nuestro enfoque junto con sus elementos como así también realizar las transformaciones especificadas en la Sección 5.

La herramienta *MagicMobileUWE* usa, para la creación de los modelos y sus elementos, el perfil⁴² *Mobile Hypermedia* definido en la Sección 4.2 y completado en las Secciones 5.7 y 5.8 (el cual importa los perfiles *Mobile UWE*, *Location*, *Layout* y *User Model*).

Para esta tesis se pudo aprovechar la experiencia del plugin (*MagicUWE*) de *UWE* contando con el código fuente de la versión 1.2.5 provisto por los autores de dicho plugin. El código fuente de *MagicUWE* sirvió como base para la creación de *MagicMobileUWE*, para el cual seguimos la misma línea de resolución del plugin de *UWE*. *MagicMobileUWE* actualmente esta funcionando para la versión 16.6 SP1 de *MagicDraw*. A esta versión de *MagicDraw* se le instaló, además, el plugin *MagicUWE* versión 1.3.0, el cual funciona con el perfil de *UWE* versión 1.8.

Como *MagicDraw* esta desarrollado íntegramente en *Java*, el desarrollo de *MagicMobileUWE* también se realizo usando *Java* y de forma independiente del plugin *MagicUWE*; logrando de esta manera que ambos evolucionen en forma independiente. *MagicDraw* permite instalar *MagicUWE* y *MagicMobileUWE*, permitiendo que cada uno provea sus modelos (con sus elementos) y sus transformaciones.

Al iniciar *MagicDraw* se cargan todos los plugins disponibles incluyendo *MagicMobileUWE* (en el Anexo A se pueden leer más detalles de cómo se definió el plugin y cómo es cargado por *MagicDraw*). En la Figura 6.1 se puede visualizar la pantalla inicial de *MagicDraw*, en la cual se destacó el menú *MagicMobileUWE* creado por la herramienta.

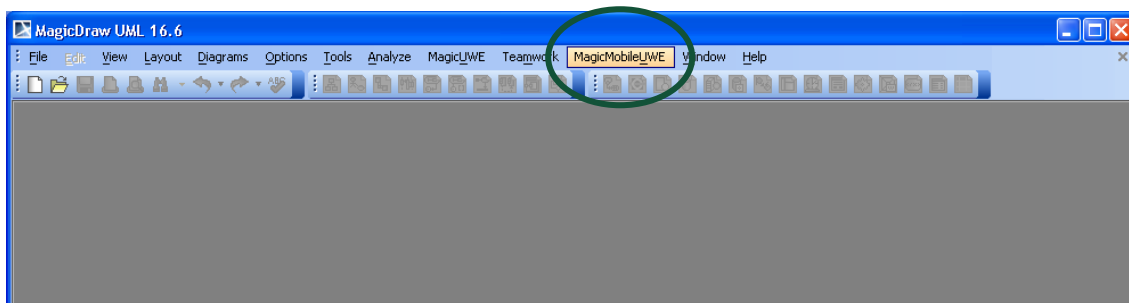


Figura 6.1: Menú *MagicMobileUWE* en la herramienta *MagicDraw*.

MagicDraw permite empezar a trabajar a partir de la creación de un proyecto. Al crear un proyecto la herramienta pregunta si se le quiere asociar al proyecto, el perfil *Mobile Hypermedia* de *MagicMobileUWE*. Al indicar que si, se levanta el perfil *Mobile Hypermedia* junto con los perfiles que importa: *Mobile UWE* (que a su vez se importa el perfil de *UWE*), *Location*, *Layout* y *User Model*. Si no se tienen asociados al proyecto estos perfiles no se pueden utilizar los elementos definidos en los mismos.

Supongamos que creamos un proyecto nuevo llamado “*Ejemplo del Turista*” (en este proyecto se representa una aplicación turística), se puede apreciar en la Figura 6.2 cómo quedan especificados los perfiles del proyecto. Dentro de cada proyecto quedaran creados los modelos asociados al mismo.

También se puede ver el menú desplegado *MagicMobileUWE*. Cabe aclarar, que hay

⁴² Todos los perfiles usados por nuestra herramienta se crearon usando *MagicDraw*.

opciones no disponibles como por ejemplo las transformaciones, ya que éstas se habilitan cuando hay creado y abierto al menos un modelo, caso contrario no tiene sentido aplicar una transformación.

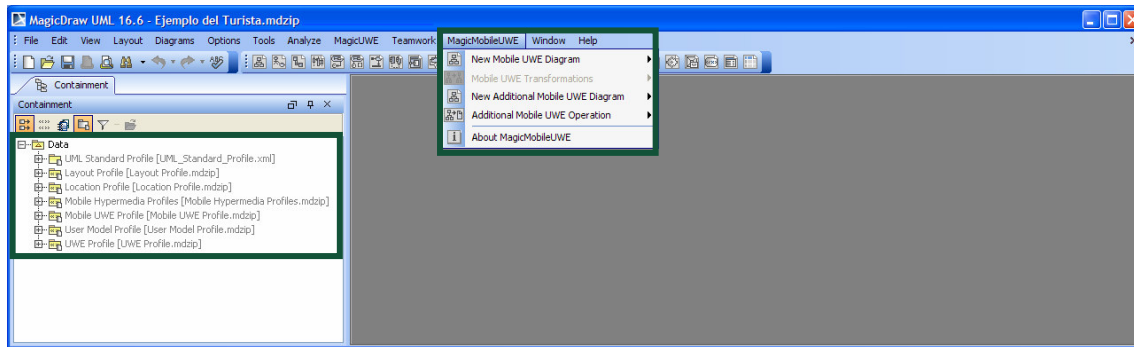


Figura 6.2: Creación de un Proyecto y menú desplegado de *MagicMobileUWE*.

En las siguientes secciones se presenta la funcionalidad definida por *MagicMobileUWE*, mostrando cómo se crean los distintos modelos del enfoque (presentados en la Sección 5). Se muestra además que opciones brinda esta herramienta para facilitar la creación de estos modelos.

6.2. *MagicMobileUWE*: Modelo de Contenido

El Modelo de Contenido debe ser especificado íntegramente por el diseñador, ya que en el mismo debe definir los concerns de la aplicación, junto con sus clases y sus relaciones específicas. Para crear este modelo, el diseñador cuenta, en el menú principal, con una opción llamada *Mobile UWE Content Diagram* como se muestra en la Figura 6.3. Al seleccionar esta opción se crea un Modelo de Contenido en el cual el diseñador puede definir los elementos que contiene el mismo.

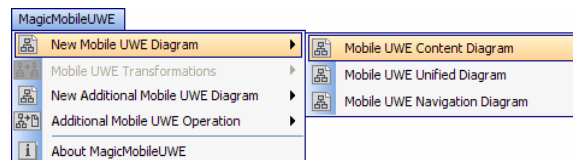


Figura 6.3: Opción del menú para crear un Modelo de Contenido.

Una vez seleccionada la opción para crear el Modelo de Contenido (*Mobile UWE Content Diagram*), el diseñador recibe una pantalla con el modelo creado (ver Figura 6.4). En la solapa llamada *Containment* se crea un paquete denominado *Content*⁴³ y dentro del mismo, un modelo denominado *Mobile UWE Content Diagram*.

El paquete *Content* contiene todos los Modelos de Contenido que cree el diseñador en este proyecto, permitiendo tener organizados los distintos modelos involucrados en el enfoque. Inicialmente, los Modelos de Contenido se crean con el nombre *Mobile UWE Content Diagram*, pero el diseñador cuenta con la posibilidad de modificarlo. Si se crea más de un Modelo de Contenido, se le incrementa al final del nombre un número para que no haya conflictos de nombres. Es decir, los modelos quedan especificados con el nombre *Mobile UWE Content Diagram i*, donde *i* es un número incremental el cual se incrementa a medida que se crean nuevos modelos. Esta forma de creación se aplica también al resto de los modelos del enfoque.

En el centro de la Figura 6.4 se destaca la barra de herramientas creada por la herramienta, que permite la creación a nivel de dibujo, de todos los elementos del

⁴³ *MagicUWE* usa la creación de paquetes para agrupar los modelos, *MagicMobileUWE* también sigue esta línea (agrupar los modelos en paquetes particulares).

enfoque tanto sea para los Modelos de Contenido, Unificado y Navegacional. Es decir, todos los estereotipos definidos en el perfil *Mobile UWE*, tienen su correspondiente dibujo para facilitar al diseñador la creación de estos elementos. Se puede ver que los elementos están agrupados según el modelo que los contiene: *Mobile UWE Content*, *Mobile UWE Unified* o *Mobile UWE Navigation*. También se destaca en la Figura 6.4 la información relacionada al modelo, como es el paquete donde está contenido (*Content*), el nombre de dicho modelo (*Mobile UWE Content Diagram*) y su estereotipo `<<mobileUWEContentDiagram>>`. Este estereotipo permite expresar que este modelo no es un diagrama de clases de UML común, sino que es un modelo específico del enfoque. Mediante este estereotipo se identifican nuestros Modelos de Contenidos.

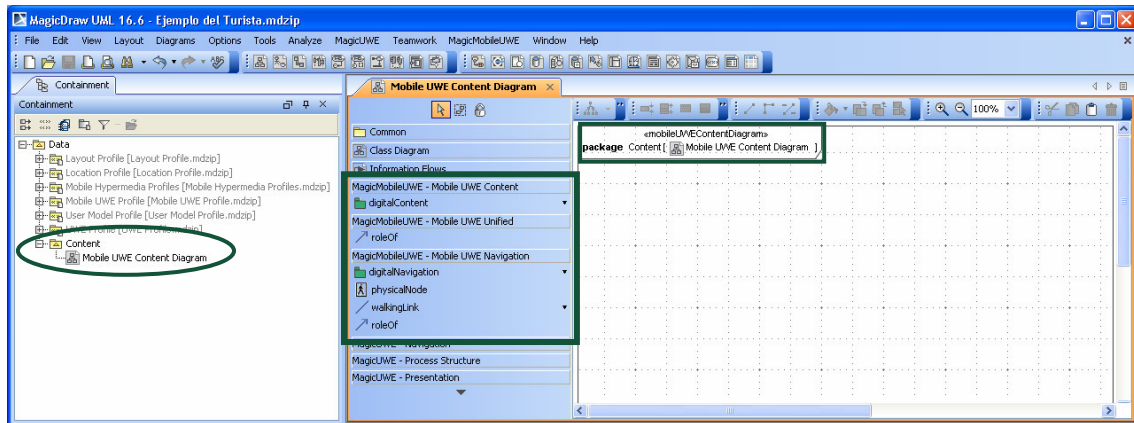


Figura 6.4: Visualización del Modelo de Contenido creado.

Veamos como el diseñador debe empezar a crear los elementos de su Modelo de Contenido usando la herramienta. Primero debe definir un concern para luego definirle los elementos internos que contiene.

Se puede apreciar en la barra de herramientas de la Figura 6.5 se tiene disponible la posibilidad de especificar tanto concerns digitales como el concern físico. Al elegir cualquiera de estos dos elementos y clicar luego en el modelo, se crea un paquete con el estereotipo correspondiente `<<digitalContent>>` o `<<physicalContent>>`. En la figura se puede apreciar que se crearon dos concerns, el histórico y el físico. Estos concerns aparecen ahora dentro del paquete *Content* como se visualiza en la solapa *Contentment*.

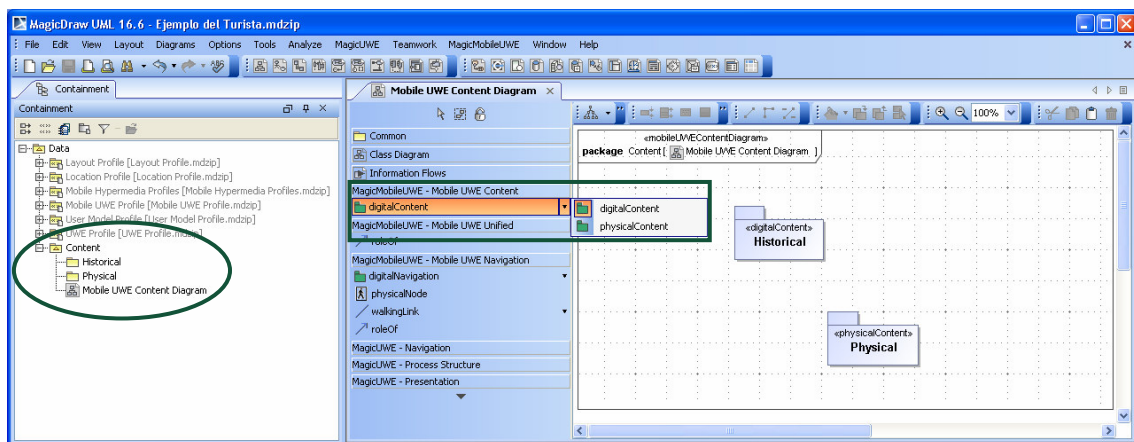


Figura 6.5: Creación de concerns en el Modelo de Contenido.

Las clases internas a cada concern se crean eligiendo el elemento *Class* de la barra de herramientas especificada para los diagramas de clases (*Class Diagram*). Esta barra (destacada en la Figura 6.6) es provista por *MagicDraw*. Cada clase se crea con

sus propiedades (indicándole a cada una al menos un nombre y el tipo de la misma). Por el contrario, las clases que están definidas en el concern físico tienen la particularidad de definir la propiedad de ubicación. La herramienta facilita esta creación brindando un menú contextual en las clases del concern físico llamado *MagicMobileUWE: Create Location Property* como se puede ver en la Figura 6.6.

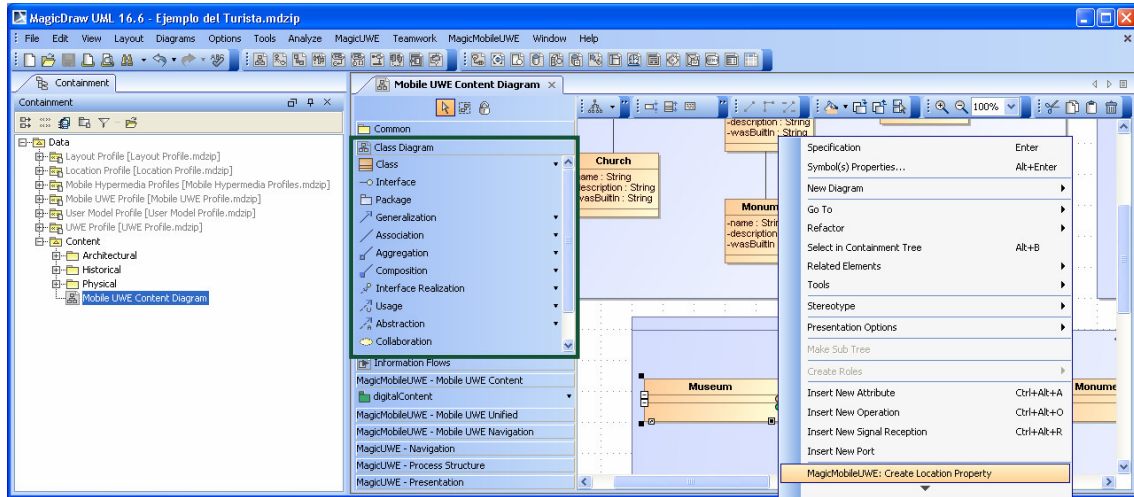


Figura 6.6: Menú contextual para crear la propiedad de ubicación.

Al seleccionar la opción *MagicMobileUWE: Create Location Property*⁴⁴ el diseñador recibe una pantalla con los tres estereotipos definidos que puede tomar la propiedad de ubicación (ver Figura 6.7). De esta manera, el diseñador elige cuál es el estereotipo con que se crea dicha propiedad.

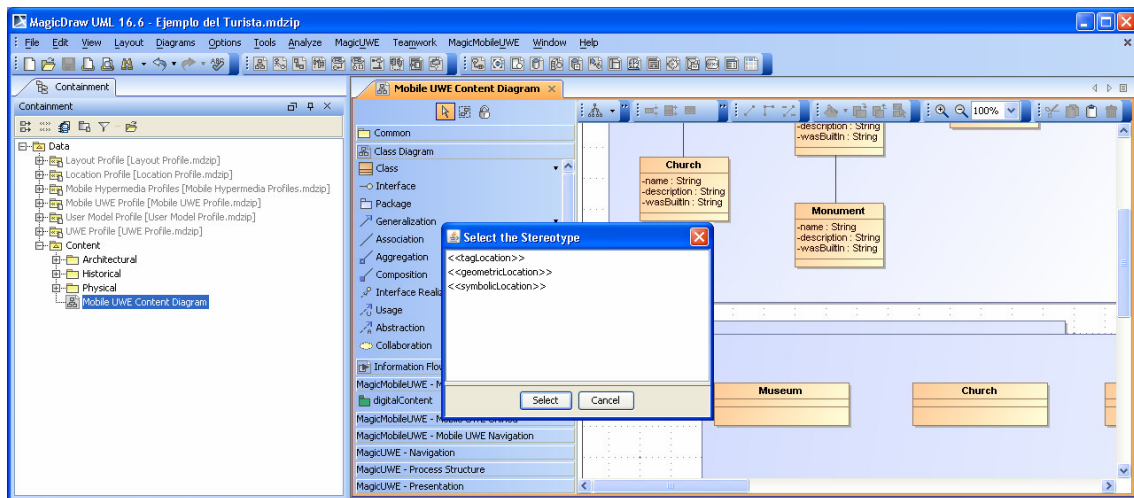


Figura 6.7: Elección del estereotipo para la propiedad de ubicación.

Supongamos que el diseñador elige estereotipar la propiedad de ubicación como <<tagLocation>>, el diseñador visualiza una pantalla como se muestra en la Figura 6.8. Esta propiedad siempre se define con tipo *String* sin importar el estereotipo. El estereotipo permite dar la semántica adecuada al valor que tome esta propiedad. El diseñador debe definir para cada clase del concern físico la propiedad de ubicación. Este menú contextual (para crear la propiedad de ubicación) también aparece si el diseñador selecciona más de una clase del concern físico. Al elegir un estereotipo, la propiedad se crea en todas las clases seleccionadas, con el estereotipo seleccionado. De esta manera, se agiliza la tarea del diseñador.

⁴⁴ Este menú contextual sólo está disponible para los Modelos de Contenido.

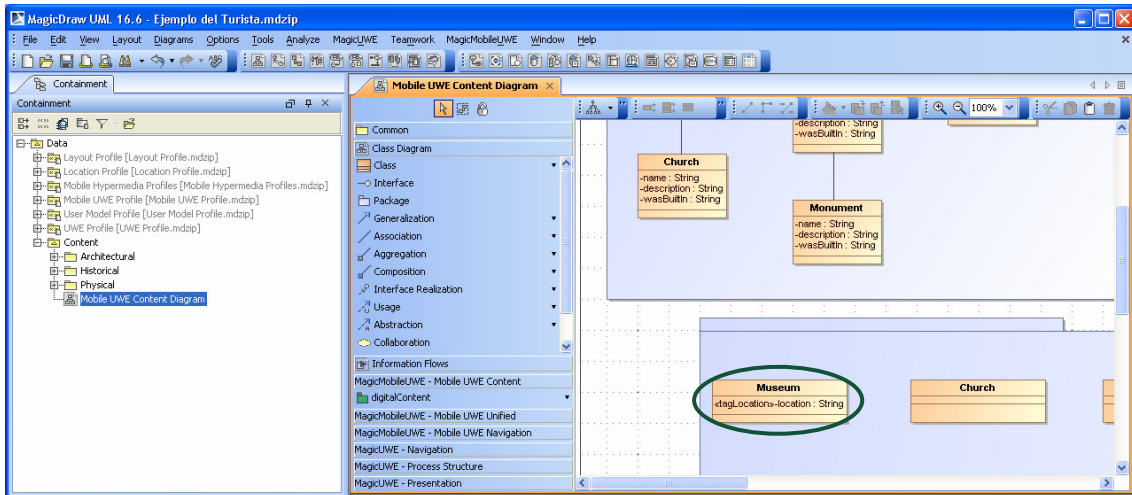


Figura 6.8: Propiedad de ubicación creada.

Cabe aclarar que la opción *MagicMobileUWE: Create Location Property* del menú contextual está disponible sólo cuando se seleccionan clases del concern físico, caso contrario la opción no aparece como disponible.

6.3. *MagicMobileUWE*: Modelo Unificado

Una vez especificado el Modelo de Contenido, el diseñador puede generar a partir de éste el Modelo Unificado, para lo esto debe elegir cuál es el concern *Core* de la aplicación. Para realizar la transformación de un modelo al otro, se provee dentro del menú *MagicMobileUWE*, el submenú *Mobile UWE Transformation* con la opción *Mobile UWE Content → Mobile UWE Unified*, como se puede apreciar en la Figura 6.9. Con esta transformación se pasa todo el contenido de los concerns del Modelo de Contenido al Modelo Unificado, donde uno de ellos es elegido como *Core* de la aplicación. Esta transformación agrega las relaciones de rol.

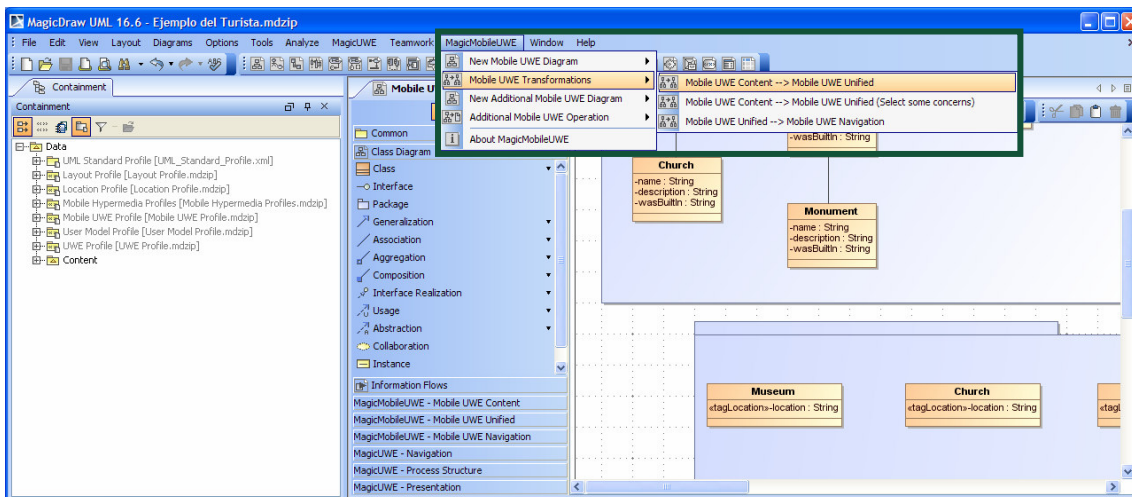


Figura 6.9: Menú con la transformación del Modelo de Contenido al Modelo Unificado.

Cuando el diseñador elige la transformación *Mobile UWE Content → Mobile UWE Unified*, le aparece en una pantalla un listado con todos los concerns digitales que tiene el Modelo de Contenido actual (es decir el Modelo de Contenido que se está visualizando) como se muestra en la Figura 6.10. El diseñador debe elegir uno de estos como *Core* o cancelar dicha transformación. En el caso que el modelo que se

está visualizando no sea un Modelo de Contenido, se le avisa al diseñador que esta transformación no se puede realizar.

También se controla que el Modelo de Contenido tenga concerns digitales para listar, en el caso de no tener ningún concern digital disponible, se le indica también que no se puede realizar dicha transformación.

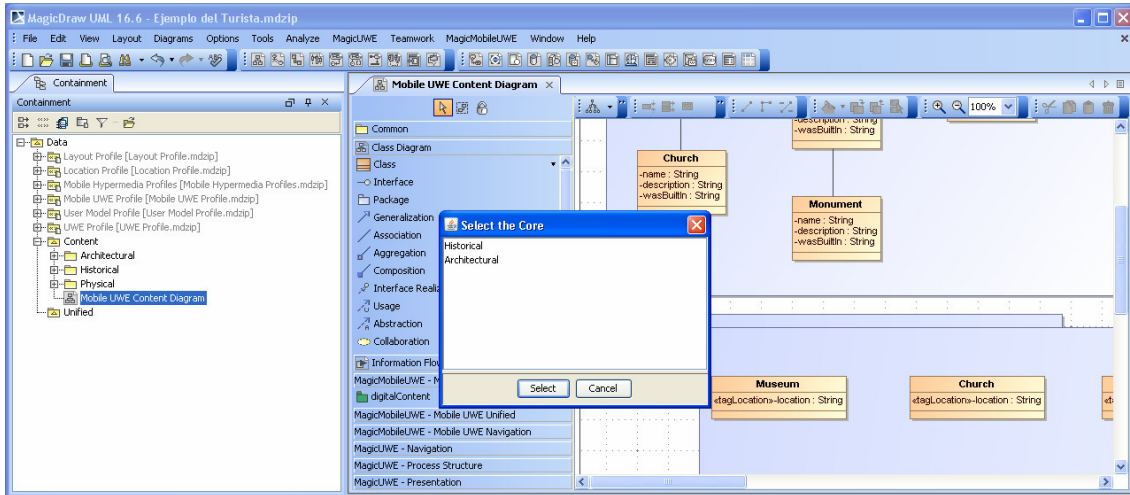


Figura 6.10: Elección del concern Core.

Supongamos que el diseñador selecciona, por ejemplo, el concern histórico, la transformación es realizada por la herramienta en forma automática⁴⁵ (ver Anexo B) según la descripción indicada en la Sección 5.3.

Como resultado de la transformación (*Mobile UWE Content* → *Mobile UWE Unified*) se crea un nuevo modelo denominado *Mobile UWE Unified* (ver Figura 6.11 dentro de la solapa *Containment*). En este caso, se eligió crear el nuevo modelo dentro del paquete *Tourist All Concerns*, que a su vez está contenido en el paquete *Unified*, esto es posible porque la transformación permite especificar donde se quiere guardar⁴⁶ el nuevo modelo creado.

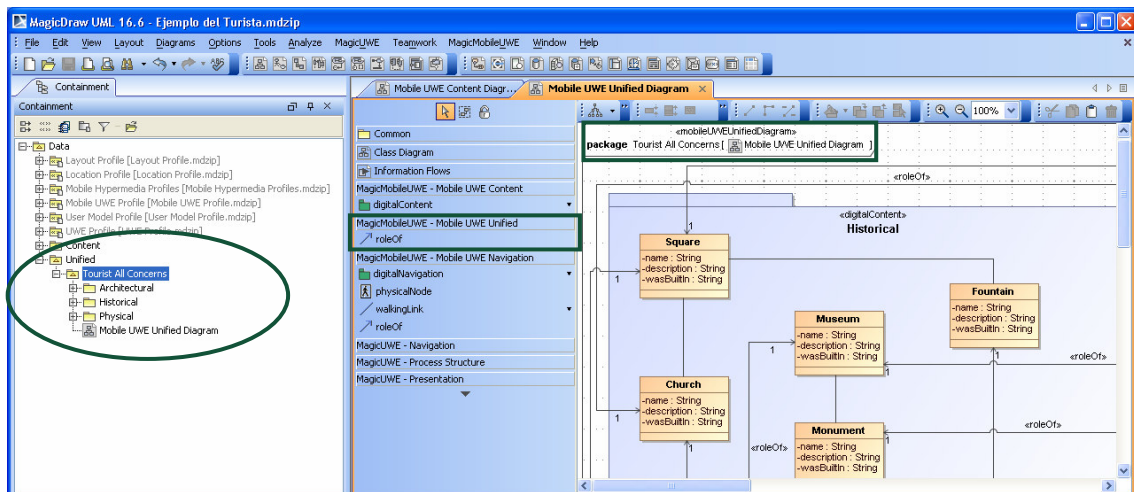


Figura 6.11: Modelo Unificado creado con la transformación.

⁴⁵ Para una mejor visualización del modelo creado a partir de la transformación, se acomodaron de forma manual algunas líneas de las asociaciones.

⁴⁶ En el caso de no seleccionar ningún paquete en particular, el modelo es creado dentro del paquete llamado *Unified*.

El nuevo Modelo Unificado se crea con el estereotipo <<mobileUWEUnifiedDiagram>> como se puede apreciar en la parte derecha de la Figura 6.11. Este estereotipo sirve para indicar que es un Modelo Unificado del enfoque. Además, el estereotipo permite almacenar el Modelo de Contenido que fue derivado el Modelo Unificado, permitiendo de esta manera recuperar su modelo origen.

Se puede visualizar en la Figura 6.11, que se agregaron las asociaciones estereotipadas como <<roleOf>> acordes al concern que se eligió como *Core* (en este caso el concern histórico).

En la barra de herramientas (dentro del grupo *Mobile UWE Unified*) el diseñador tiene como opción el elemento *roleOf*, que le permite dibujar la relación de rol en el caso que quiera relacionar dos clases de dos concerns diferentes (que tienen diferentes nombres); ya que la transformación sólo relaciona con la asociación estereotipada como <<roleOf>> las clases de distintos concerns que tienen el mismo nombre.

Como se mencionó en la Sección 5.3, el diseñador puede elegir transformar sólo algunos concerns al Modelo Unificado. Para esto, se provee la transformación llamada *Mobile UWE Content* → *Mobile UWE Unified (Select some concerns)* como se puede observar en la Figura 6.12.

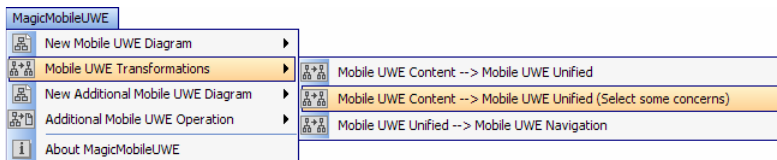


Figura 6.12: Menú con la transformación del Modelo de Contenido al Modelo Unificado considerando sólo algunos concerns de interés.

Cuando el diseñador selecciona la transformación mencionada, le aparece una pantalla con la lista de todos los concerns digitales del Modelo de Contenido (como se mostró en la Figura 6.10). El diseñador elige cuál es el concern *Core* de la aplicación. Supongamos que elige el concern histórico, una vez que lo selecciona, le aparece otra pantalla (Figura 6.13) con el resto de los concerns digitales para que indique cuáles quiere considerar para la transformación al Modelo Unificado. En ese momento, el diseñador puede elegir alguno de los concerns restantes o no elegir ninguno. La transformación se lleva a cabo considerando sólo el concern *Core* y los concerns seleccionados en la segunda pantalla.

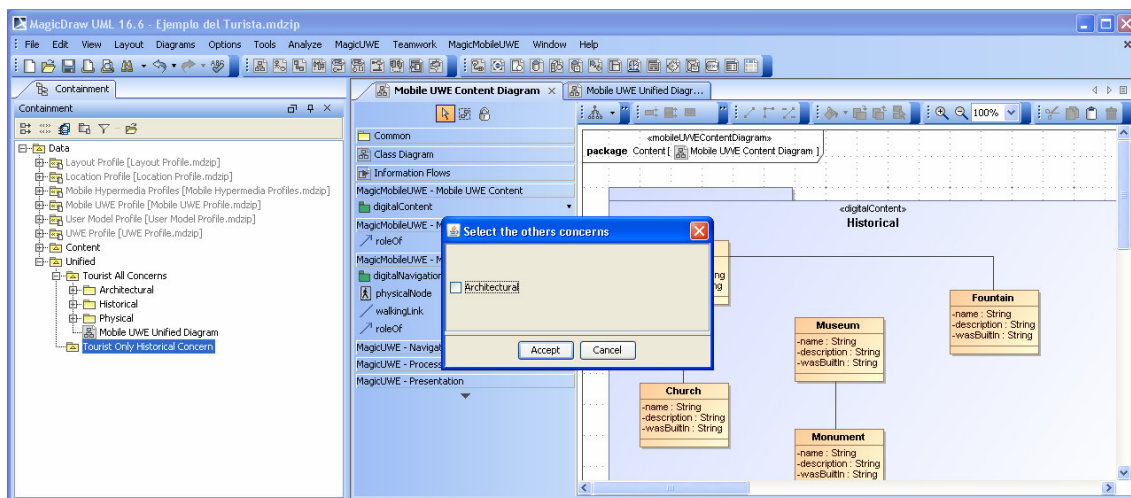


Figura 6.13: Selección de los concerns de interés.

Por como está definida la transformación del Modelo de Contenido al Modelo Unificado, elegir los concerns de interés, asegura que al menos el Modelo Unificado cuente con el concern *Core*. El concern físico siempre se pasa en la transformación y no se le permite al diseñador excluirlo. El código relacionado con esta transformación se puede observar en el Anexo B.

En la Figura 6.13 se puede ver que se crea, dentro del paquete *Unified* (dentro de la solapa *Containment*), un nuevo paquete llamado *Tourist Only Historical Concern* donde se almacena el nuevo Modelo Unificado creado a partir de esta transformación. De esta manera quedan bien organizados los distintos modelos en paquetes separados permitiendo una buena organización.

Supongamos que en la pantalla de la Figura 6.13, el diseñador no elige pasar el concern arquitectural. En este caso la transformación sólo considera el concern histórico y físico para crear el Modelo Unificado. Esto se puede apreciar en la Figura 6.14, que como producto de la transformación se genera el Modelo Unificado *Mobile UWE Unified Diagram* dentro del paquete *Tourist Only Historical Concern*.

Se puede observar que este Modelo Unificado también tiene el estereotipo `<<mobileUWEUnifiedDiagram>>`. Además, se puede visualizar que no se crea el paquete correspondiente al concern *Architectural*, solamente se crean los paquetes *Historical* y *Physical*.

Usando esta transformación, el diseñador puede filtrar los concerns que le interesa seguir considerando, para luego crearles una representación en el Modelo Navegacional. En este filtrado puede excluir aquellos concerns de la aplicación que no tienen sentido que sean visualizados por el usuario.

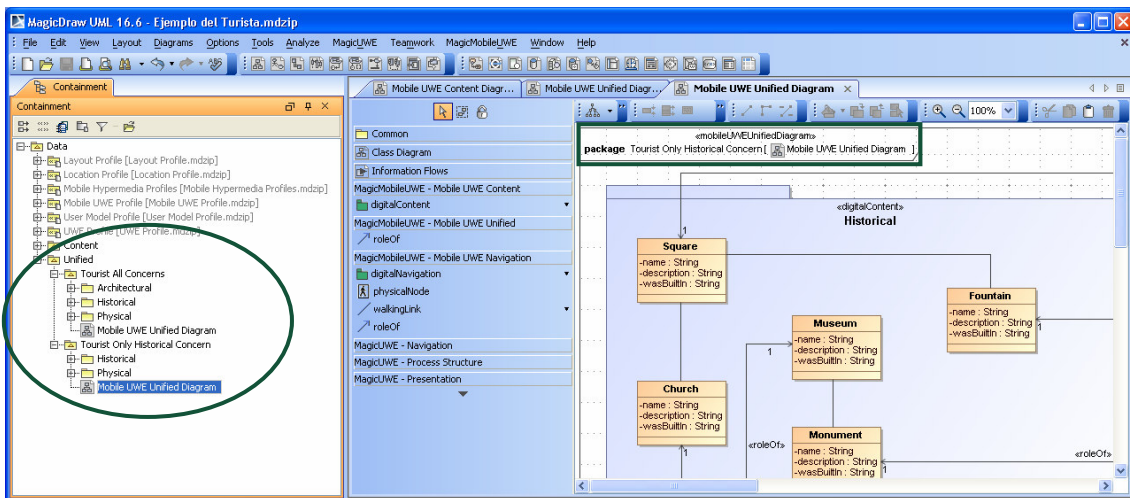


Figura 6.14: Modelo Unificado creado solamente con el concern histórico.

Una vez realizada la transformación, el diseñador tiene la posibilidad de agregar nuevas asociaciones de rol (usando el elemento de dibujo *roleOf*), entre las clases de distinto nombre, que considere necesario.

La herramienta *MagicMobileUWE* permite además crear Modelos Unificados sin que provengan de un Modelo de Contenido. Esta opción se provee en el menú principal (ver Figura 6.15). Este modelo tiene que ser creado en forma completa por el diseñador, indicando manualmente todos los concerns junto a sus clases y sus asociaciones; como así también la definición de las asociaciones que representan al rol. Para realizar la creación de todos estos elementos, el diseñador puede usar los elementos disponibles en la barra de herramientas.

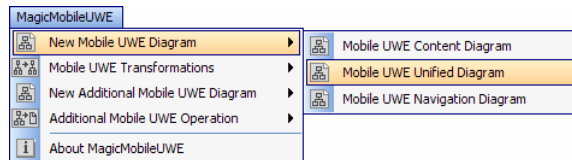


Figura 6.15: Menú para la creación del Modelo Unificado.

6.4. MagicMobileUWE: Modelo Navegacional

El Modelo Navegacional se puede crear a partir de un Modelo Unificado. Para poder realizar esta creación se puede utilizar la transformación llamada *Mobile UWE Unified* → *Mobile UWE Navigation*, que provee la herramienta como se observa en la Figura 6.16.

Para que esta transformación se pueda llevar a cabo se debe tener abierto un Modelo Unificado, en caso contrario se le indica al diseñador que la transformación no se puede realizar.

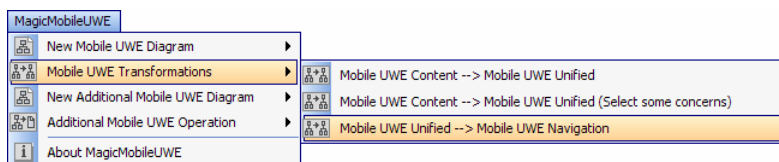


Figura 6.16: Menú con la transformación del Modelo Unificado al Modelo Navegacional.

Supongamos que el usuario tiene abierto el Modelo Unificado que se creó con todos los concerns (Figura 6.11). Al elegir la transformación *Mobile UWE Unified* → *Mobile UWE Navigation*, se le crea un nuevo Modelo Navegacional llamado *Mobile UWE Navigation Diagram* como se puede visualizar en la Figura 6.17⁴⁷.

Supongamos que el nuevo modelo creado se usa para representar el Modelo Navegacional asociado a una aplicación turística de la ciudad de *Roma*. El diseñador puede elegir donde almacenar este modelo, en este caso se almacenó dentro del paquete *Roma* (que está contenido en el paquete *Tourist All Concerns*, que a su vez está contenido en *Navigation*). En el caso de no seleccionar ningún paquete en particular, el Modelo Navegacional creado se almacena por default en el paquete *Navigation*.

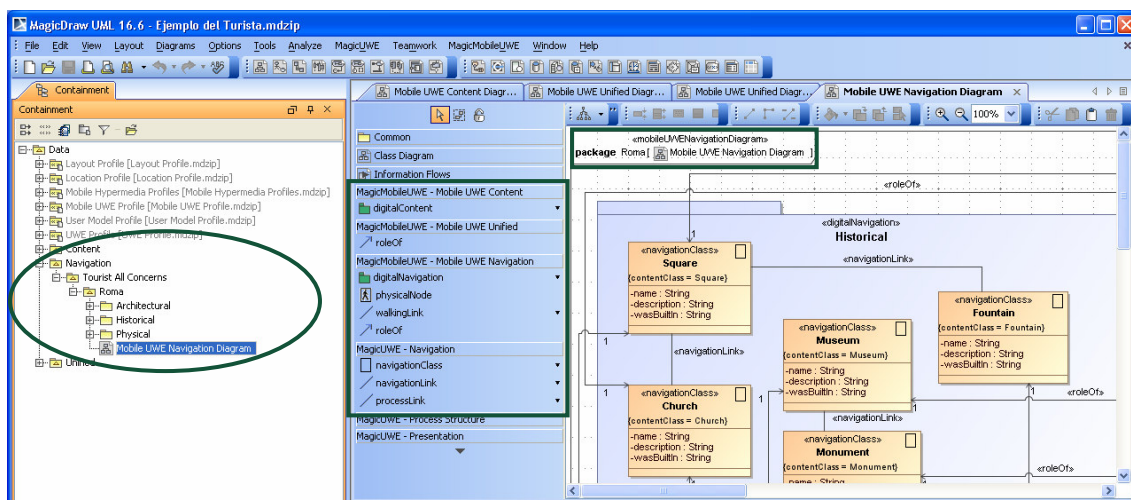


Figura 6.17: Modelo Navegacional creado con la transformación.

⁴⁷ Para una mejor visualización del modelo creado a partir de la transformación se acomodaron de forma manual algunas líneas de las asociaciones.

La transformación del Modelo Unificado al Modelo Navegacional, se realiza siguiendo la especificación detallada en la Sección 5.5 y el código asociado a la misma se puede ver en el Anexo B. El nuevo modelo es creado con el estereotipo <<mobileUWENavigationDiagram>>. Este estereotipo representa a nuestros Modelos Navegacionales y contiene el Modelo Unificado desde el cual fue creado el Modelo Navegacional.

Cabe recordar que a partir de un mismo Modelo Unificado se pueden derivar varios Modelos Navegacionales.

Como se mencionó en la Sección 5.5 una vez realizada la transformación el diseñador tiene la posibilidad de refinar el Modelo Unificado, tanto sea agregando o sacando tanto nodos como links.

Para la creación de los elementos asociados a este modelo, el diseñador puede usar, de la barra de herramientas (remarcada en la Figura 6.17), tanto los elementos definidos por *MagicUWE* para sus Modelos Navegacionales como así también nuestros elementos definidos en *MagicMobileUWE*. Los elementos de *MagicUWE* se usan únicamente para la definición de los elementos de los concerns digitales. La herramienta es compatible con la definición de los elementos de *MagicUWE*, permitiendo que los mismos puedan ser usados en nuestros Modelo Navegacionales.

El diseñador puede ajustar el Modelo Navegacional de acuerdo a las necesidades de su aplicación, ya que mediante el estereotipo asociado a dicho modelo se puede recuperar el Modelo Unificado original.

Además, en la etapa de refinamiento, como se mencionó en la Sección 5.5, se pueden definir las operaciones de links caminables derivados en los nodos físicos. Se provee un menú contextual para los nodos físicos que permite la creación de esta operación, como se puede visualizar en la Figura 6.18. Esta opción de menú sólo se despliega para los nodos físicos.

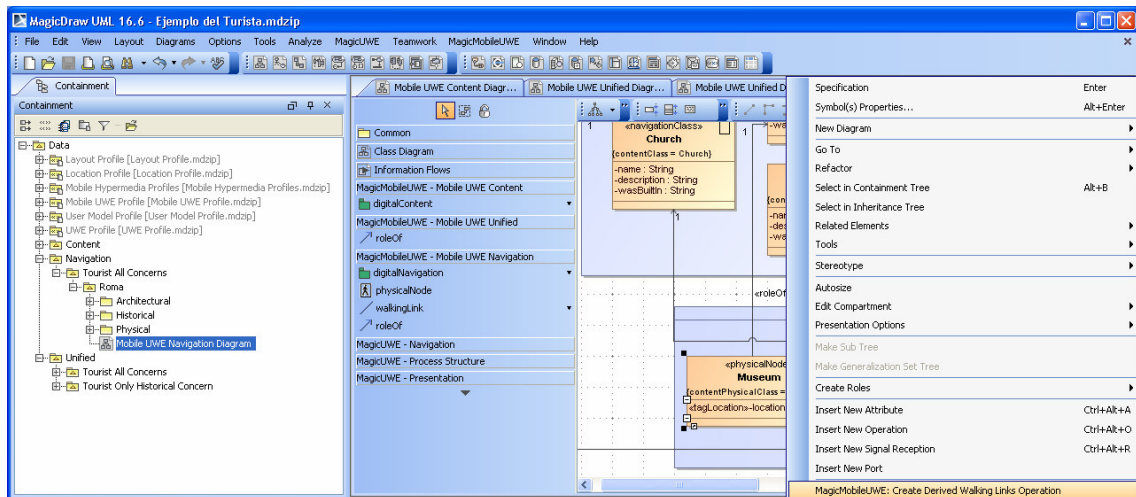


Figura 6.18: Menú que permite crear la operación de link caminable derivado.

Una vez que el diseñador elige la opción *MagicMobileUWE: Create Derived Walking Link Operation*, se listan los dos estereotipos que (por ahora) provee el enfoque, como se observa en la Figura 6.19. Para que se cree dicha operación se debe elegir uno de los dos estereotipos, caso contrario la operación no es creada. Por ahora se tienen definidos estas dos operaciones, pero en futuras evoluciones podrían surgir más elementos a esta lista.

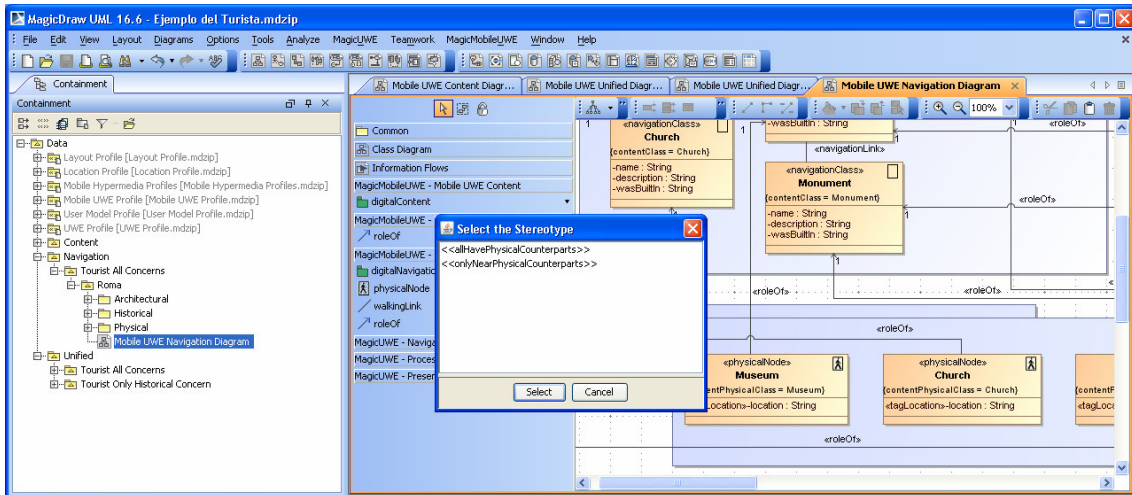


Figura 6.19: Opciones de estereotipos para la operación de links caminables derivados.

Una vez que el diseñador elige un estereotipo para la operación, se crea la operación llamada `derivedWalkingLinks` con el estereotipo seleccionado como se muestra en la Figura 6.20. Supongamos que el diseñador elige, por ejemplo, el estereotipo `<<allHavePhysicalCounterparts>>`, entonces la operación se define con este estereotipo determinando así el comportamiento de la misma.

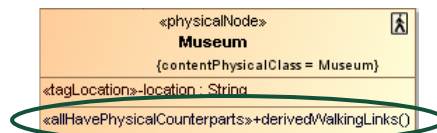


Figura 6.20: Operación de links caminables derivados creada.

La herramienta permite que el diseñador seleccione más de un nodo físico, y también le aparece el menú contextual para crear la operación de links caminables derivados. Al elegir un estereotipo, se crea en todos los nodos físicos seleccionados la operación de links caminables derivados con el estereotipo elegido. En este caso, se agiliza la tarea del diseñador ya que puede pasar que varios o todos los nodos físicos definan la misma operación para derivar los link caminables. Esta forma de definir la operación de links caminable derivados permite la flexibilidad de tener nodos que no definen dicha operación o que la definen con distinto estereotipo.

También permite crear el Modelo Navegacional sin que provenga de un Modelo Unificado. Esto se puede realizar mediante la opción del menú principal (ver Figura 6.21). Al ser creado el Modelo Navegacional de esta manera, el diseñador debe especificar en forma completa todos sus elementos: concerns, nodos, links y roles).

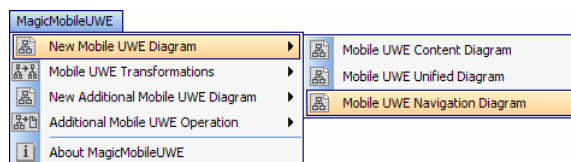


Figura 6.21: Menú para la creación del Modelo Navegacional.

Para poder crear los elementos de este modelo (Navegacional) el diseñador puede hacer uso de los elementos provistos por la barra de herramientas. Para este modelo particular, el diseñador puede usar tanto los elementos definidos por el plugin *MagicUWE* como los elementos definidos por *MagicMobileUWE*.

6.5. MagicMobileUWE: Modelo de Instancias Navegacional

Una característica distintiva de los tres modelos que se presentan en las siguientes tres secciones, es que ninguno de estos se puede crear de manera aislada. En el caso del Modelo de Instancias Navegacional, se crea a partir de un Modelo Navegacional de base, sin un modelo de base no tiene sentido el Modelo de Instancias Navegacional.

Para crear este modelo, se provee una opción en el menú como se muestra en la Figura 6.22.

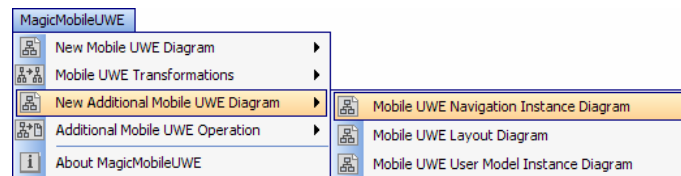


Figura 6.22: Menú para la creación del Modelo de Instancias Navegacional.

Al seleccionar la opción *Mobile UWE Navigation Instante Diagram*, el diseñador recibe la lista de posibles Modelos Navegacionales que se pueden instanciar. Los modelos listados son los que se encuentran dentro del proyecto y están estereotipados como <<mobileUWENavigationDiagram>>. La pantalla que recibe el diseñador con el listado de los Modelos Navegacionales se puede apreciar en la Figura 6.23. En el caso que no haya ninguno disponible, se le indica al diseñador que no puede crear un Modelo de Instancias Navegacional. El código relacionado con esta creación se puede observar en el Anexo C.

La creación del Modelo de Instancias Navegacional no requiere que el Modelo Navegacional que se elija de base esté abierto, situación que también se visualiza en la Figura 6.23.

Por default el Modelo de Instancias Navegacional creado se almacena en el paquete *Navigation Instante* (en la solapa *Containment*), pero el diseñador puede elegir donde almacenarlo, por ejemplo en este caso, el diseñador elige almacenarlo en el paquete *Roma* (contenido en el paquete *Tourist All Concerns*).

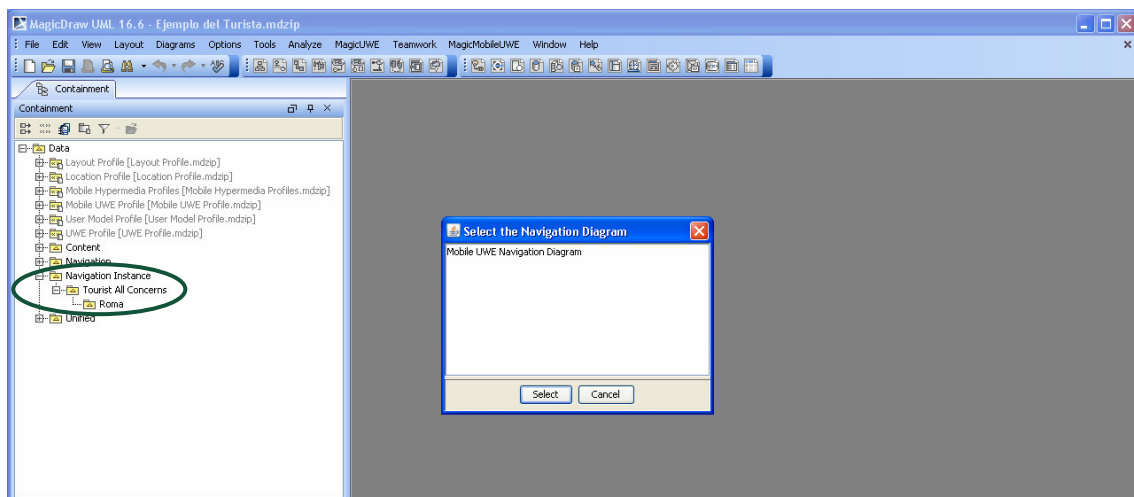


Figura 6.23: Elección de Modelo Navegacional de base.

Supongamos que el diseñador selecciona como base el único Modelo Navegacional (*Mobile UWE Navigation Diagram*) disponible, el Modelo de Instancias Navegacional que se crea se puede ver en la Figura 6.24. Se puede observar que se crearon vacíos los concerns que contiene el Modelo Navegacional elegido. El diseñador debe crear las instancias adecuadas según la aplicación que quiera especificar. Se puede

apreciar además (en la Figura 6.24) que se crea el Modelo de Instancias Navegacional con el estereotipo `<<mobileUWENavigationInstanceDiagram>>`, este estereotipo contiene el Modelo Navegacional de base que permite saber que objetos se pueden instanciar.

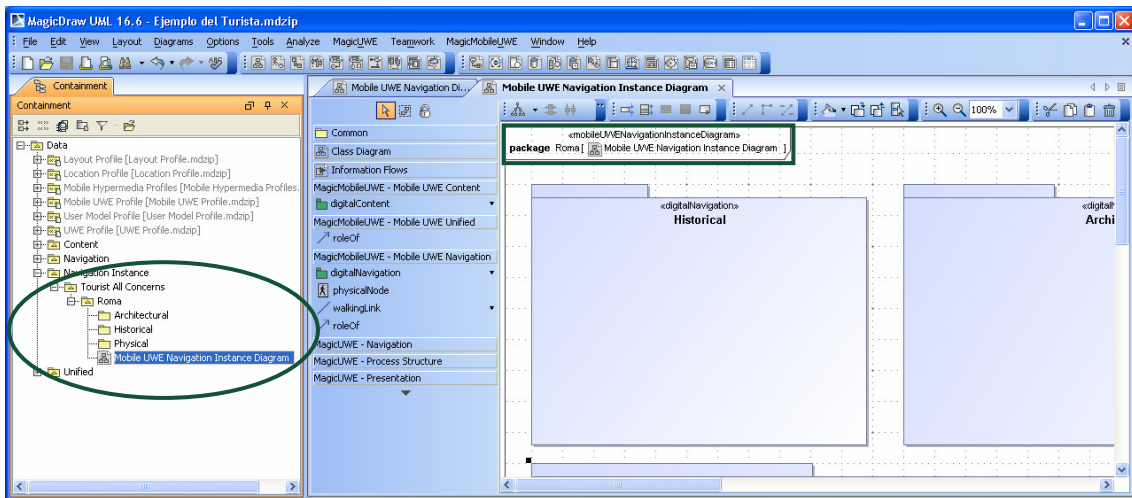


Figura 6.24: Modelo de Instancias Navegacional creado.

Para crear las instancias de los nodos digitales, la herramienta (*MagicMobileUWE*) provee para los concerns digitales un menú contextual (ver Figura 6.25). Este menú sólo está disponible cuando se selecciona un concern digital.

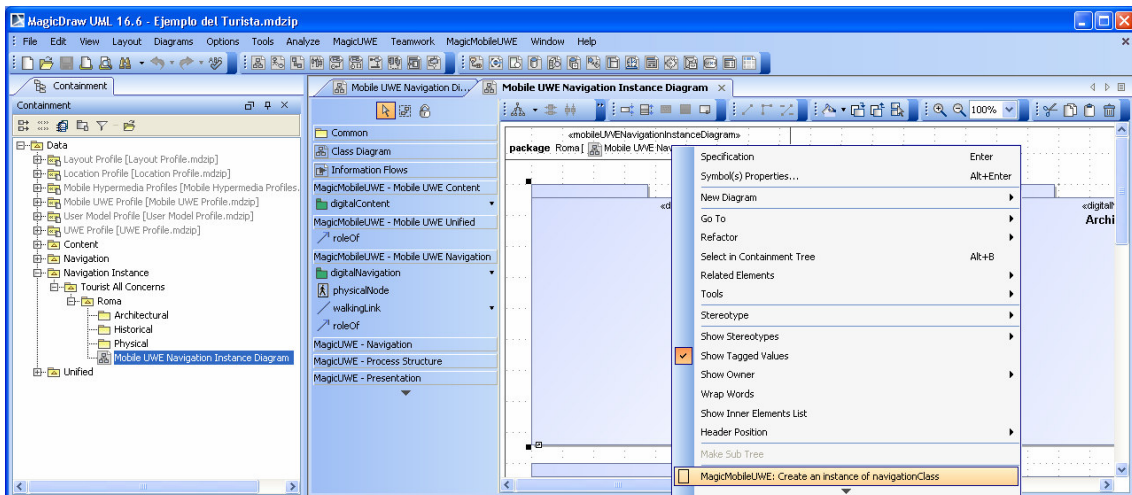


Figura 6.25: Menú contextual para la creación de instancias de nodos digitales.

Al elegir la opción *MagicMobileUWE: Create instante of navigationClass*, se crea un objeto como se puede apreciar en la Figura 6.26. Este objeto se crea estereotipado como `<<navigationClass>>`, pero no se le define el nodo base (o la clase) del mismo. Para definir cuál es nodo base de la instancia creada, se usa el menú contextual que tienen todas las instancias estereotipadas como `<<navigationClass>>` (ver Figura 6.26). Al elegir la opción *MagicMobileUWE: Set Classifier* se lista al diseñador todos los nodos que tiene el Modelo Navegacional para el concern donde está definida la instancia. Esta consideración permite ayudar a instanciar sólo nodos definidos en cada concern del Modelo Navegacional.

El diseñador puede instanciar cada nodo las veces que sea necesario para la aplicación que está representando. También puede pasar que en un Modelo de

Instancias Navegacional no se instancian todos los nodos del Modelo Navegacional. Esto le permite al diseñador tener flexibilidad a la hora de especificar los nodos (digitales o físicos) de una aplicación en particular.

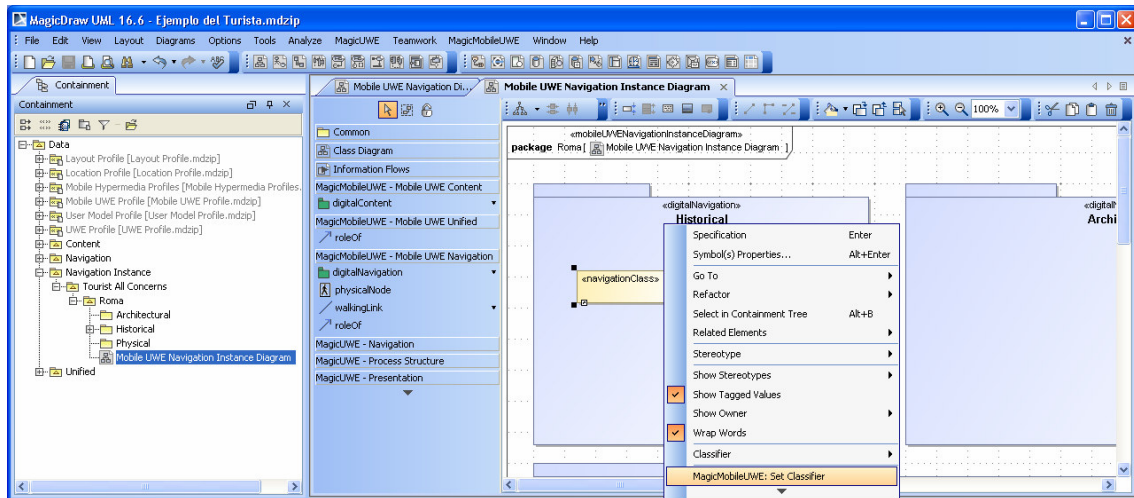


Figura 6.26: Setear el nodo base de la instancia.

Al elegir la opción *MagicMobileUWE: Set Classifier*, el diseñador ve una pantalla como se muestra en la Figura 6.27. Se puede apreciar la lista de nodos que tiene el Modelo Navegacional para el concern histórico. Para realizar este cálculo se toma el Modelo Navegacional de base y se busca el concern que tiene el mismo nombre que el concern que contiene la instancia. Una vez encontrado el concern en el Modelo Navegacional, se obtienen todos los nodos digitales contenidos en él y se listan sus nombres en la pantalla.

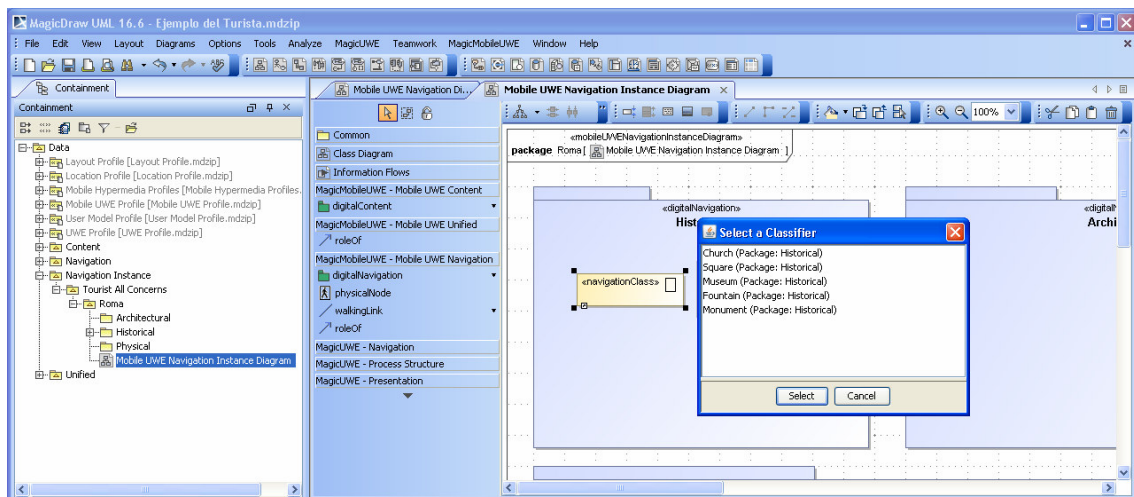


Figura 6.27: Listado de las clases del concern Histórico.

Si los nodos digitales no tienen un nombre asociado, no son listados como posibles nodos a instanciar, está es una cuestión que debe considerar el diseñador que utilice la herramienta. De esta manera, se trata de evitar el problema de tener nodos sin nombres, ya que en el caso de haber más de un nodo sin nombre no se pueden diferenciar entre si.

Supongamos que el diseñador elige como nodo base *Museum*. Se puede apreciar en la Figura 6.28 como queda en la instancia indicado el nodo base de la misma. Este

seteo se realiza agregando el nodo base como *Classifier*⁴⁸ [UML Superstructure] de la instancia.

También se puede apreciar (Figura 6.28) que los nodos se listan para la instancia creada en el concern arquitectural. Comparando las Figuras 6.27 y 6.28 se puede apreciar que las listas de nodos varían acorde al concern que contiene a la instancia.

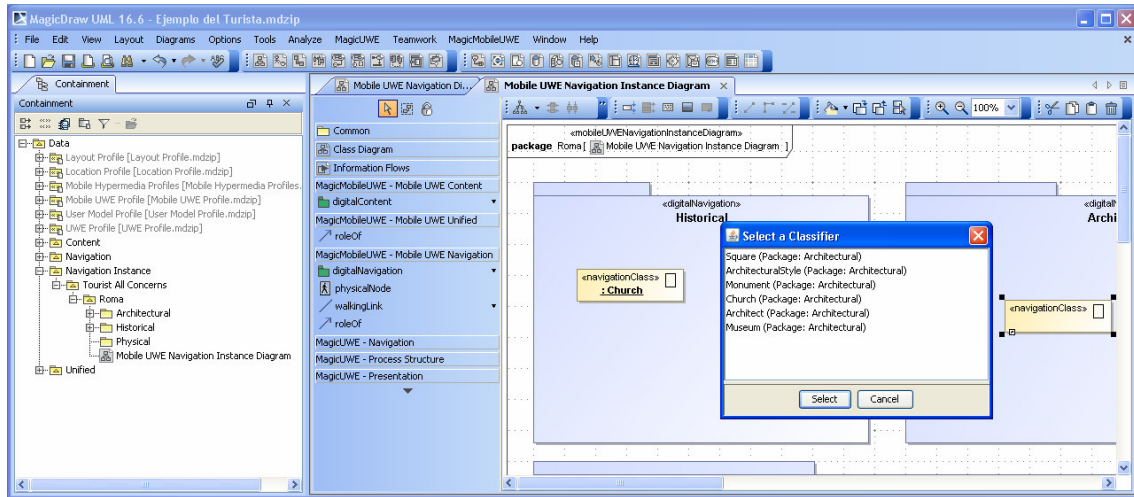


Figura 6.28: Listado de las clases del concern Arquitectural.

El diseñador debe indicar a qué nodo corresponde cada instancia para poder determinar qué propiedades y asociaciones tiene cada instancia, para poder así determinar cómo interpretar dicha instancia y su comportamiento.

En el caso del concern físico, se provee un menú contextual que permite crear instancias de nodos físicos como se puede apreciar en la Figura 6.29. Esta opción sólo está disponible cuando se selecciona el concern físico.

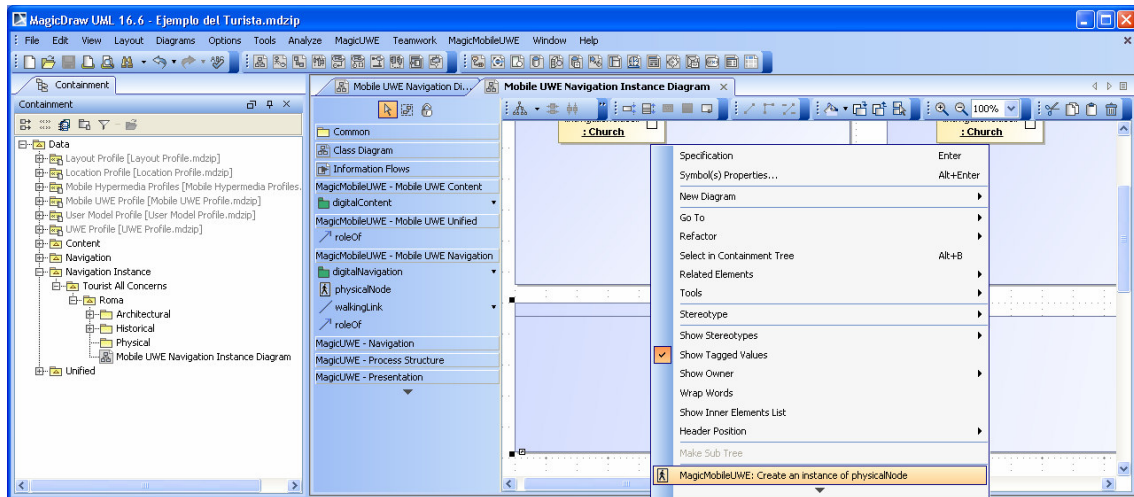


Figura 6.29: Menú contextual para la creación de instancias de nodos físicos.

Cuando el diseñador elige la opción *MagicMobileUWE: Create instante of physicalNode* se le crea una instancia estereotipada como `<<physicalNode>>` como se visualiza en la Figura 6.30. Esta instancia también cuenta con la opción de un menú

⁴⁸ Elemento de UML que permite especificar las clases de la instancia. Para UML una instancia puede tener más de un *Classifier*, por esta razón el diseñador debe considerar que sólo debe setear un sólo *Classifier* por instancia. Esta es la manera de representar a que nodo pertenece cada instancia sino una instancia puede quedar asociada a más de un nodo.

contextual que permite al igual que los nodos digitales setearle el nodo físico correspondiente (como se mostró en la Figura 6.26). Al seleccionar la opción *MagicMobileUWE: Set Classifier*, se muestra la lista de los nodos físicos del Modelo Navegacional (ver Figura 6.30).

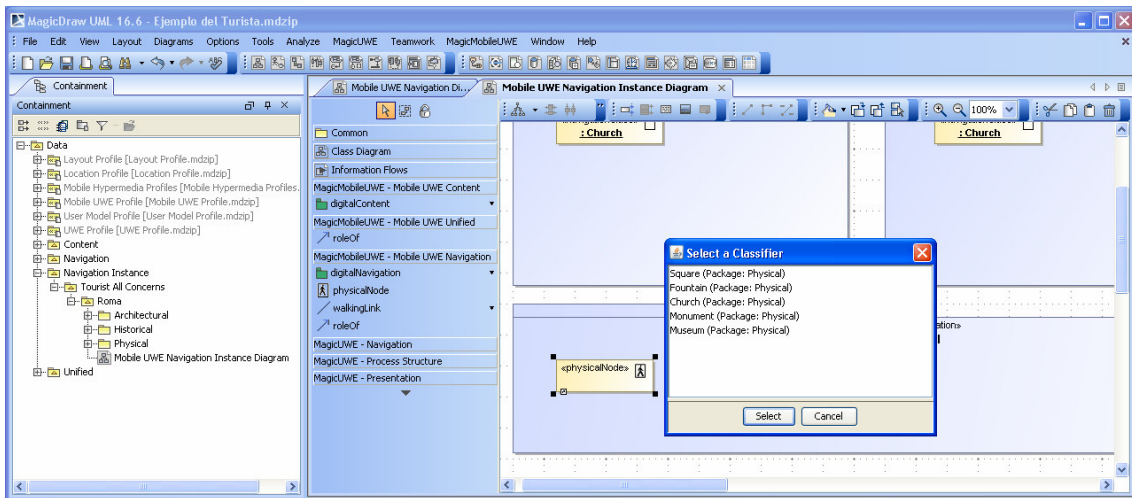


Figura 6.30: Listado de las clases del concern físico.

Para facilitar la tarea del diseñador, se provee un menú contextual para los Modelos de Instancias Navegacionales, que permite completar dicho modelo de acuerdo al Modelo Navegacional de base. Es decir, de acuerdo al nodo base que tenga asociado cada instancia, se crean sus propiedades con un valor por default para que luego el diseñador especifique su valor. También se crean las asociaciones de roles entre las instancias que corresponda, considerando que las mismas tengan igual nombre. La opción de completar se puede visualizar en la Figura 6.31, la misma se puede ejecutar sólo una vez en cada Modelo de Instancias Navegacional. El estereotipo de los Modelos de Instancias Navegacionales (`<<mobileUWENavigationInstanceDiagram>>`) tiene un valor asociado que indica si el modelo ya fue completado o no. Según el valor almacenado, el menú contextual está o no disponible para el modelo en cuestión.

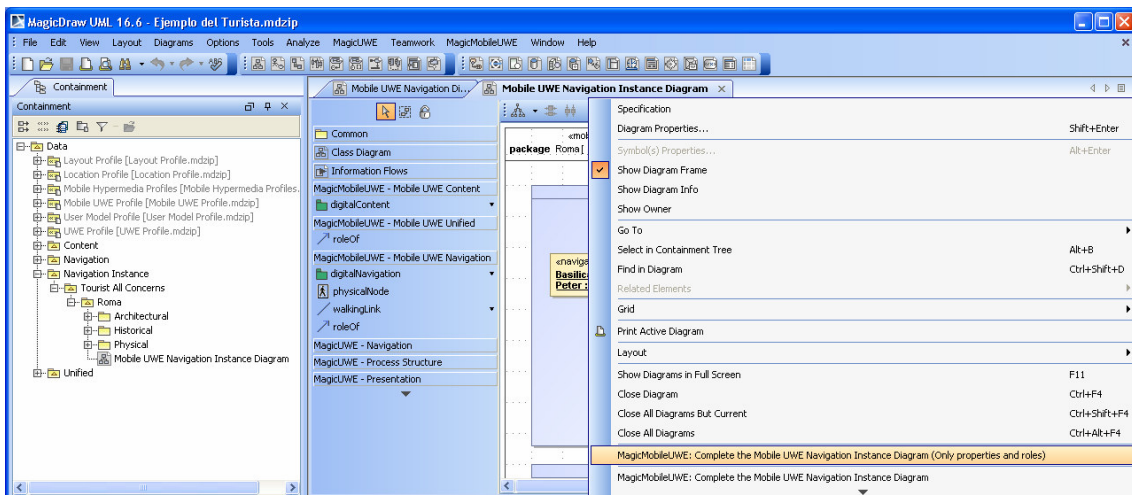


Figura 6.31: Menú contextual para completar el Modelo de Instancias Navegacional.

Supongamos que para mostrar el funcionamiento de la función completar el diseñador define tres instancias del nodo *Church*, una en cada concern con el nombre de *Basílica de San Pedro* y tres instancias (una en cada concern) del nodo *Monument* con el nombre *Panteón*. Estas instancias son creadas usando los menús contextuales

presentados en la Figuras 6.24 y 6.29 y se les asigna el nodo base usando el menú presentado en la Figura 6.25.

Al realizar la operación de completar se tiene como resultado las instancias de la Figura 6.32. Se puede observar que se crea la asociación estereotipada como `<<roleOf>>`, esta asociación no solamente contempla que sean clases de nodos con el mismo nombre, sino que además las instancias se llamen iguales. Por esta razón se relaciona por un lado las instancias de la *Basilica de San Pedro* y por otro las instancias del *Panteón*. También se ve que se agregaron las propiedades acordes al nodo base de cada instancia. Según el nodo base de la instancia, se toman las propiedades del mismo y cada una se agrega en la instancia con un valor inicial. Hasta el momento la herramienta sólo completa con propiedades que están definidas con tipo *String* y como valor por default les asigna el *String* vacío.

Se puede apreciar que además de expresar la relación de rol, las instancias conocen la instancia *Core* de la cual son roles. Es necesario tener representada esta información a nivel de modelo, para que luego sea recuperada cuando se necesita generar la aplicación.

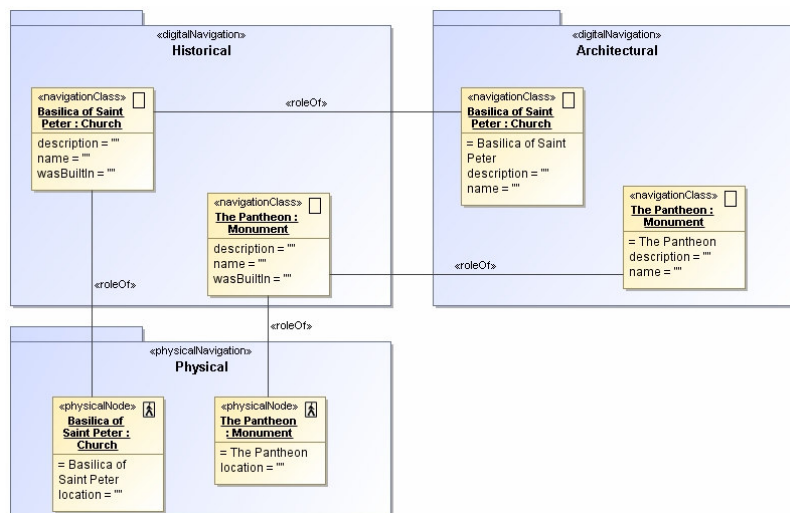


Figura 6.32: Modelos de Instancias Navegacional completo.

Una vez ejecutada la operación de completar, el diseñador debe completar el valor de cada propiedad, para esto utiliza las herramientas que provee *MagicDraw* para dar valor a las propiedades de una clase. Además, el diseñador debe agregar los links que crea convenientes entre cada una de las instancias, según las características de la aplicación que esta modelando.

Para un buen funcionamiento de la función completar, se le aconseja al diseñador indicar a qué nodo corresponde cada instancia, como así también el nombre de la misma. Caso contrario la función completa sólo aquellos datos que pueda deducir. La operación de completar agiliza la tarea del diseñador, realizando de manera automática el detalle de las propiedades y de las relaciones de rol.

6.6. *MagicMobileUWE*: Modelo de Presentación

Como pasa con el Modelo de Instancias Navegacional, el Modelo de Presentación sólo tiene sentido si se crea a partir de un Modelo Navegacional de base. Este modelo permite especificar cómo se muestra cada nodo de un Modelo Navegacional correspondiente. Para crear este modelo se provee una opción en el menú principal (*Mobile UWE Layout Diagram*) como se observa en la Figura 6.33.

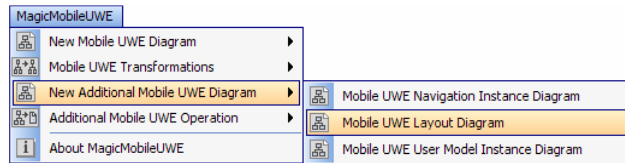


Figura 6.33: Menú con la creación del Modelo de Presentación.

Al seleccionar la opción *Mobile UWE Layout Diagram*, el diseñador recibe una pantalla como se visualiza en la Figura 6.34. Se le listan todos los posibles Modelos Navegacionales a los cuales se les puede definir una presentación. El código relacionado con esta creación se puede observar en el Anexo C.

Un Modelo Navegacional puede tener más de un Modelo de Presentación asociado; mientras que cada Modelo de Presentación específica, para cada nodo, cómo se muestran todas sus instancias si se elige visualizarlos con esa presentación.

Los Modelos de Presentación se crean por default en el paquete *Layout*, pero como en todas las creaciones realizadas usando la herramienta, el diseñador tiene la posibilidad de especificar un paquete en particular. En la Figura 6.34 se puede apreciar que el Modelo de Presentación se crea en el paquete *Roma* (contenido en el paquete *Tourist All Concerns*, que a su vez está contenido en el paquete *Layout*).

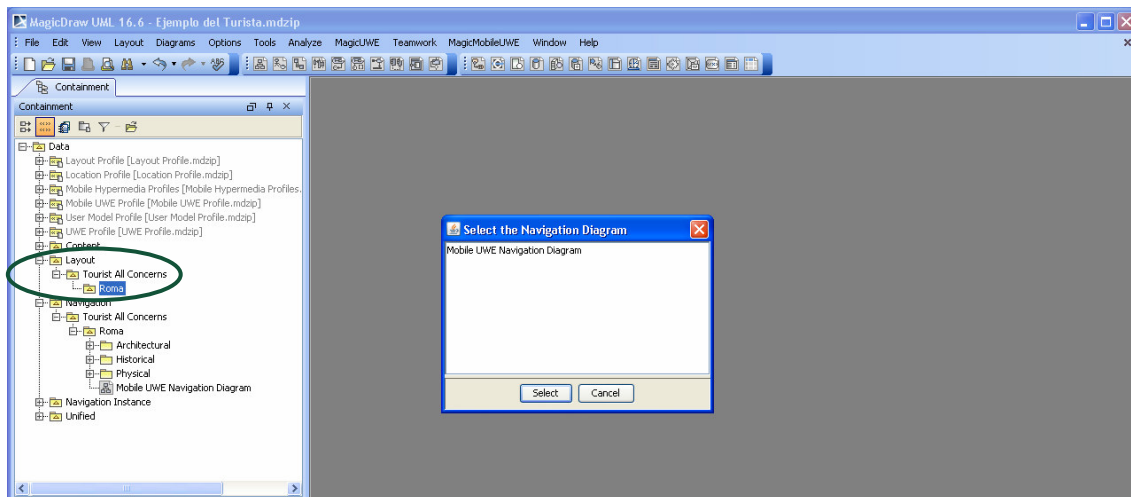


Figura 6.34: Elección del Modelo Navegacional de base.

Al seleccionar el Modelo Navegacional de base, se realiza la creación del Modelo de Presentación correspondiente según la especificación indicada en la Sección 5.7, y el código asociado a esta creación se puede observar en el Anexo C.

Al modelo creado se le asigna el estereotipo `<<mobileUWELayoutDiagram>>`, el cual almacena el Modelo Navegacional de base. El diseñador visualiza una pantalla similar a la Figura 6.35, donde se le muestran todas las clases de presentación (correspondientes a los nodos del Modelo Navegacional de base) sin un estilo definido. Se puede apreciar que las clases de presentación digitales, tienen el estereotipo `<<navigationClassLayout>>` que las caracteriza, lo mismo sucede con las clases de presentación física y ambos tipos de concerns, según la especificación detallada en la Sección 5.7.

Para este modelo el diseñador no cuenta con una barra de herramientas específica, ya que todos los elementos contenidos en este modelo, son especificados al momento de la creación del mismo a partir del Modelo Navegacional elegido como base.

El diseñador sólo debe setear el estilo a cada clase de presentación, pero no debe crear elementos nuevos, como por ejemplo nuevas clases de presentación o concerns,

ni tampoco nuevas asociaciones.

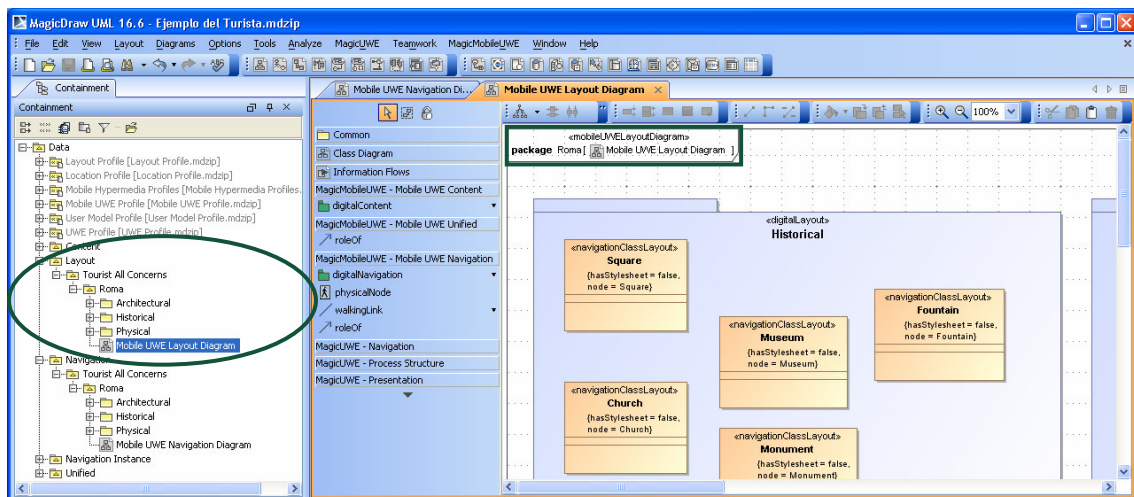


Figura 6.35: Modelo de Presentación creado.

Para las clases de presentación (independientemente del concerns), se ofrece un menú contextual (Figura 6.36) para setear el estilo a una clase. Este menú está disponible sólo para aquellas clases de presentación que aún no tienen un estilo definido, este control se realiza según el valor que tenga la etiqueta `hasStyleseet`.

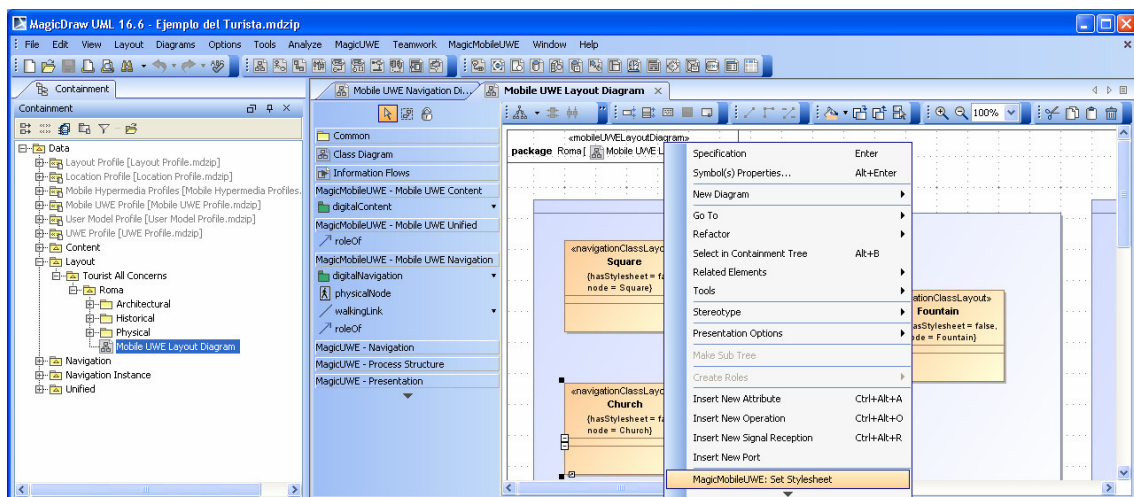


Figura 6.36: Setear el estilo a la presentación.

Como se mencionó en la Sección 5.7, para definirle el estilo a un nodo se elige asociarle un template *XSLT*⁴⁹ a la clase de presentación. Por esta razón, cuando se elige asignarle el estilo a una clase, mediante el uso del menú contextual *MagicMobileUWE: Set Stylesheet*, el diseñador recibe una ventana para que pueda seleccionar cuál es el archivo *XSLT* que se corresponde a la clase (ver Figura 6.37). La herramienta permite seleccionar más de una clase de presentación, mostrando también el menú contextual que permite setear el template *XSLT*. Cuando hay varias clases seleccionadas y se elige un archivo *XSLT*, el mismo es especificado como template en cada una de las clases seleccionadas. Esto le permite al diseñador agilizar la presentación de aquellos nodos que tienen la misma presentación. En el caso en que cada clase de presentación tenga un estilo particular, el diseñador debe especificar cada *XSLT* por separado.

⁴⁹ Junto con el template *XSLT* se asigna también el archivo *DTD* que permite la verificación del template y de los elementos que recibe dicho template.

Es esperable que el diseñador tenga al menos dos template *XSLT*, uno para los nodos digitales y otro para los nodos físicos. Ya que generalmente estos nodos contienen información diferente, en particular para los nodos físicos, se debe usar un archivo *XSLT* que considere la visualización de los links caminables predefinidos, como así también los links caminables derivados (siempre que el nodo tenga definida la operación que los calcula).

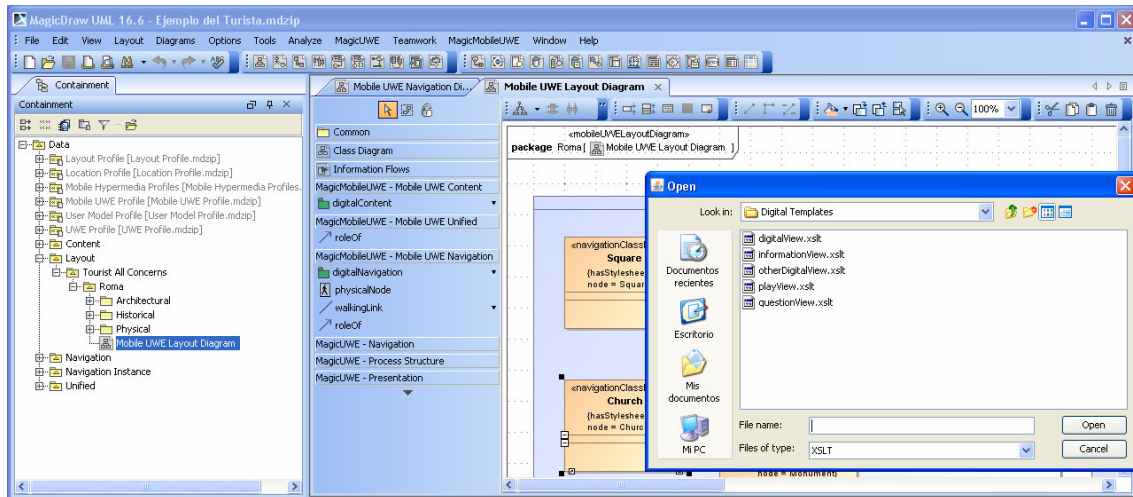


Figura 6.37: Ventana que permite elegir el estilo.

Al seleccionar el archivo *XSLT*, el mismo se parsea para detectar los parámetros que tiene y crear las propiedades correspondientes en la clase de presentación (Figura 6.38).

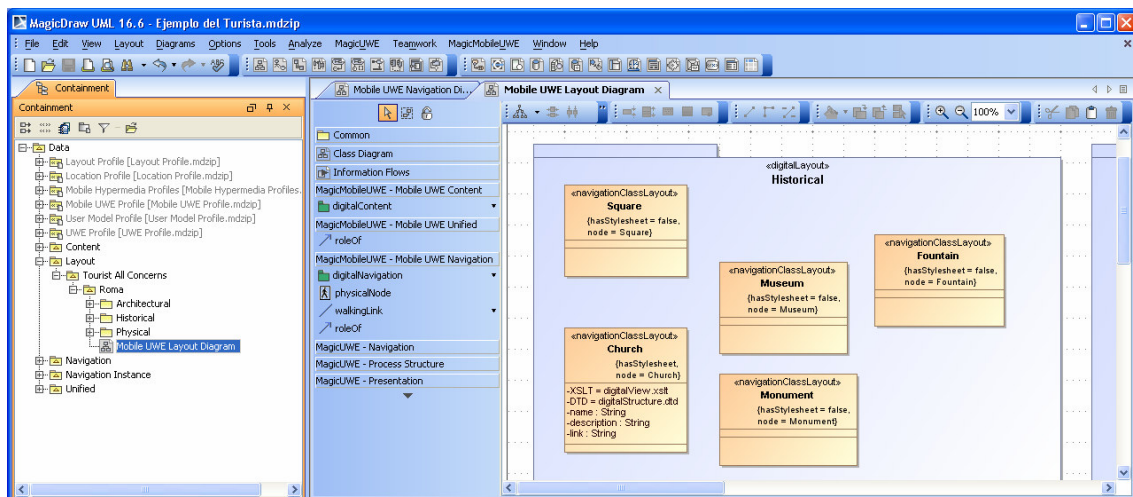


Figura 6.38: Nodo con estilo seteado.

El diseñador debe especificar qué propiedad del nodo se corresponde con cada parámetro del template, como se mostró en la Sección 5.7. Esta asignación de valores, se realiza usando las opciones que provee la herramienta *MagicDraw* para dar valor a las propiedades de una clase.

6.7. *MagicMobileUWE*: Modelo del Usuario

Para crear el Modelo del Usuario se provee la opción que se muestra en la Figura 6.39. Como se mencionó en la Sección 5.8, el diseñador sólo puede definir para este

modelo el nombre de la aplicación junto con el Modelo de Instancias Navegacional y su presentación (acorde al Modelo Navegacional de base). El código relacionado con esta creación se puede observar en el Anexo C.

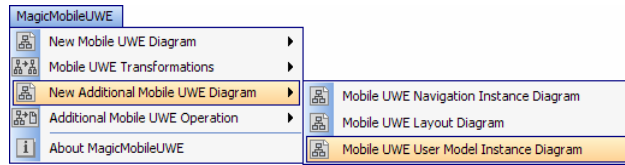


Figura 6.39: Menú para la creación del Modelo del Usuario.

Una vez elegida la opción de creación del Modelo del Usuario (Figura 6.39), el diseñador puede indicar donde se almacena este modelo (Para este ejemplo en el paquete *Roma*, contenido en el paquete *Tourist All Concerns*). Si no se indica una ubicación en particular, el paquete por default es *User Model*. El diseñador recibe una lista con todos los Modelos de Instancias Navegacionales del proyecto, es decir que estén estereotipados como `<<mobileUWENavigationInstanceDiagram>>` como se muestra en la Figura 6.40.

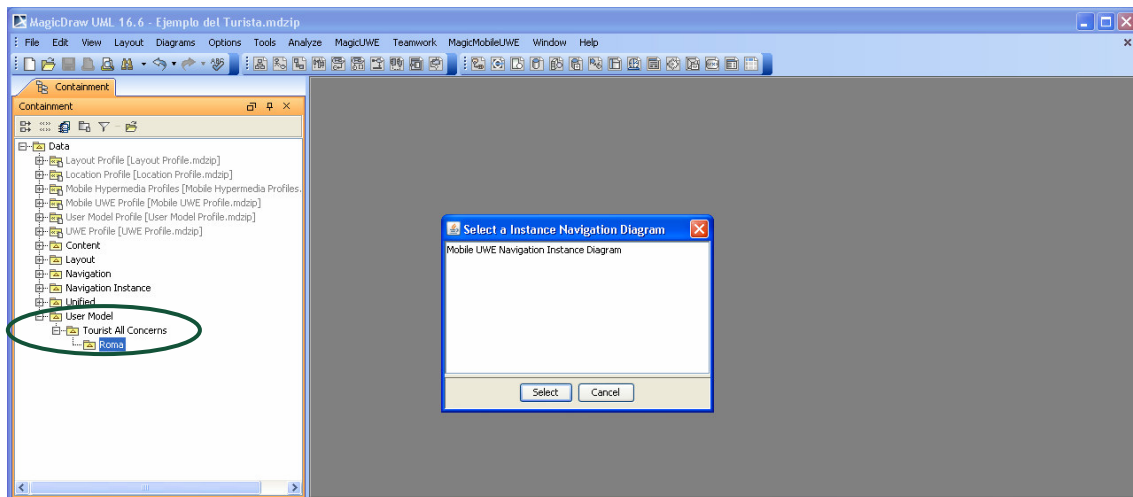


Figura 6.40: Elegir el Modelo de Instancias Navegacional para el Modelo del Usuario.

Para poder continuar con la creación del Modelo del Usuario, el diseñador debe elegir un Modelo de Instancias Navegacional. Al elegir un modelo (de Instancias Navegacional) determinado se usa el estereotipo del mismo `<<mobileUWENavigationInstanceDiagram>>` para determinar el Modelo Navegacional de base. Una vez que se encuentra este modelo, se busca dentro del proyecto todos los Modelos de Presentación que lo tengan asociado. Para buscar los posibles Modelos de Presentación que tienen igual Modelo Navegacional, se buscan todos los modelos con el estereotipo `<<mobileUWELayoutDiagram>>`. Para cada Modelo de Presentación, se controla si su Modelo Navegacional de base se corresponde con el Modelo Navegacional de base del Modelo de Instancias Navegacional. Una vez encontrados los correspondientes Modelos de Presentación para el Modelo Navegacional, estos son listados (como se ve en la Figura 6.41) al diseñador para que determine cómo se visualiza el Modelo de Instancias Navegacional previamente elegido.

Para que se cree el Modelo del Usuario, el diseñador debe elegir si o si un Modelo de Presentación. En el caso de no elegir ninguno, no se crea el Modelo del Usuario.

Puede pasar que el diseñador seleccione un Modelo de Instancias Navegacional y no haya para su Modelo Navegacional de base un Modelo de Presentación definido. En

este caso, se le avisa al diseñador que no es posible hacer la creación del Modelo del Usuario, ya que no hay al menos un Modelo de Presentación acorde al Modelo de Instancias Navegacional elegido.

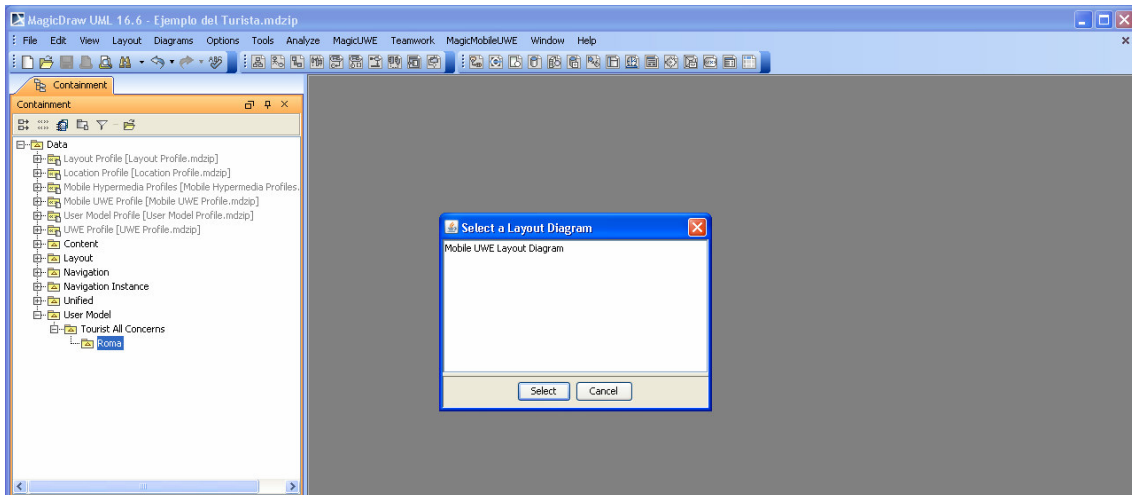


Figura 6.41: Elegir el Modelo de Presentación para el Modelo del Usuario.

Una vez seleccionado el Modelo de Presentación, se crea el Modelo del Usuario como se visualiza en la Figura 6.42. Se puede apreciar que el modelo creado tiene definido el estereotipo `<<mobileUWEUserModelInstanceDiagram>>`.

En la Figura 6.42 se puede apreciar que los modelos elegidos fueron seteados en las propiedades `instanceNavigationDiagram` y `layoutDiagram` de las instancias de las clases `MobileHypermediaApplication` y `ManagerView` respectivamente. También se puede observar que se amplió el Modelo del Usuario respecto del presentado en la Sección 5.8. Esto se debe a que hay otras características a considerar como, por ejemplo, las estrategias de navegación por concerns, las cuales incluyen la navegación digital y la navegación física. Por ahora el Modelo del Usuario se crea automáticamente con las estrategias especificadas en la Figura 6.42.

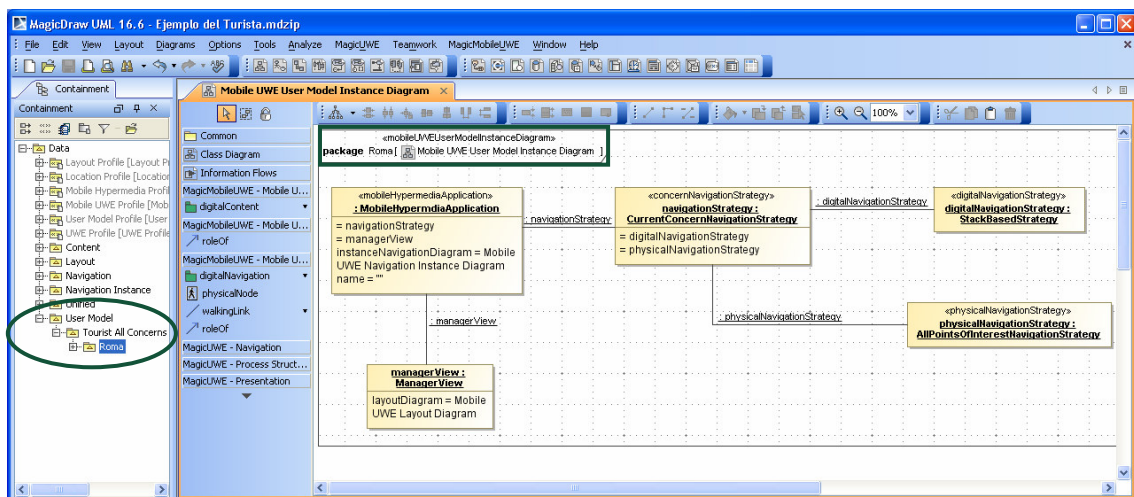


Figura 6.42: Modelo del Usuario creado.

El Modelo del Usuario, con las clases correspondientes que definimos en la Sección 5.8, se amplió como se muestra en la Figura 6.43. Las clases agregadas permiten el funcionamiento básico de las aplicaciones de HM. En este tipo de aplicaciones, las estrategias de navegación son indispensables para permitirle al usuario volver a nodos (digitales o físicos) visitados.

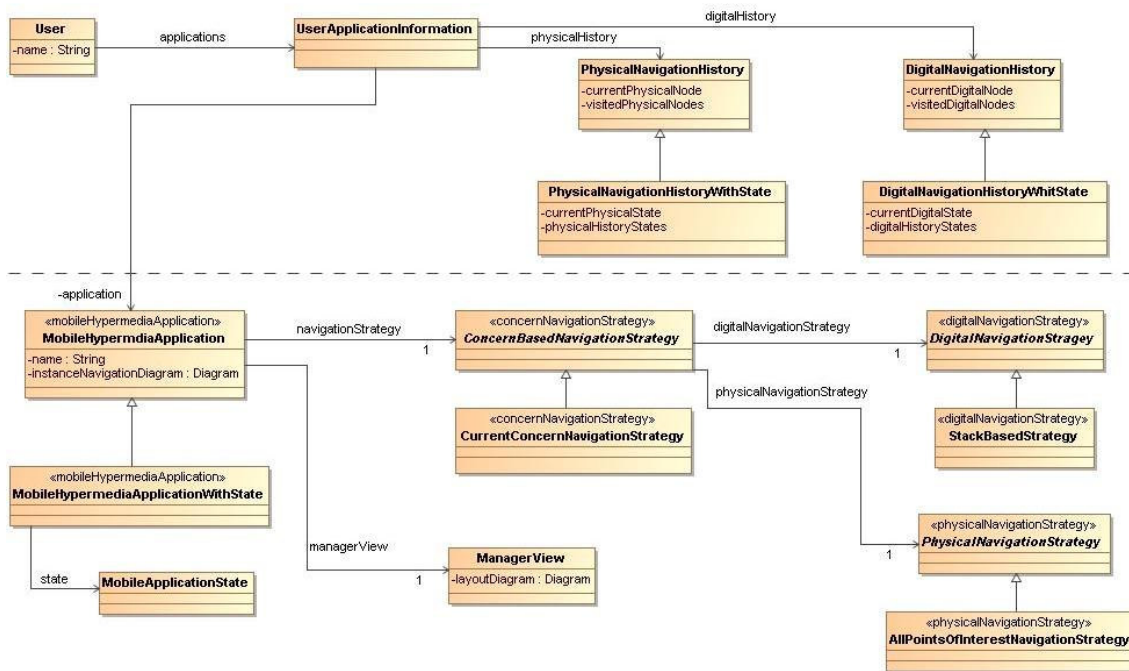


Figura 6.43: Modelo del Usuario creado en el perfil *User Model*.

Como se mostró en la Sección 5.8, la parte superior del modelo está relacionada a la navegación del usuario. En esta parte se subclasifican los historiales, agregando un nuevo historial digital y otro historial físico, los cuales, además de guardar nodos que visita el usuario, guardan los estados por los que va pasando el usuario (*Walking*, *InFrontOf*, etc.). Estos estados luego le sirven a la aplicación para brindar por ejemplo asistencia al usuario.

La parte inferior de la Figura 6.43, esta relacionada a cada aplicación particular de HM. En esta parte se agregan las estrategias de navegación cuando hay concerns, las cuales conocen cuáles son las estrategias digitales y físicas. Estas estrategias son las encargadas de guardar los nodos digitales y físicos en los historiales acorde a la naturaleza de cada estrategia.

Además, se define una subclase de *MobileHypermediaApplication* que permite crear aplicaciones que tienen asociado un estado particular. Este estado permite que la aplicación se comporte de manera diferente, o brinde distinta información a los usuarios según su estado actual.

Con el modelo presentado en la Figura 6.43, se pueden especificar aplicaciones de HM es con distintas características. Por ejemplo, aquellas que tienen asociados estados particulares, como aquellas simples que son aplicaciones con nodos para visitar. Además, se puede tener usuarios que sólo requieran guardar los nodos visitados, como también guardar los estados por los que fue pasando el mismo.

Adicionalmente, el diseñador puede elegir una estrategia de navegación de entre varias, que va a determinar cómo se guarda el historial del recorrido del usuario y las políticas de “*Next*” y “*Back*” para este último.

Las estrategias de navegación se definen en forma general, para cada aplicación el diseñador define cuál es la estrategia de navegación, independientemente de los usuarios y de los nodos que estos visiten.

De esta manera, quedan presentados todos los modelos del enfoque propuesto junto con la herramienta que se provee para crear cada uno de estos modelos. Esta herramienta le permite al diseñador agilizar su tarea a la hora de crear aplicaciones de HM.

6.8. Detalles adicionales de *MagicMobileUWE*

En esta sección se presentan otras opciones que provee *MagicMobileUWE* para facilitar al diseñador la creación de una aplicación de HM. La creación de los modelos, además de poder realizarse como se mostró en las secciones anteriores mediante el menú principal, se puede realizar mediante el menú contextual relacionado con la solapa *Containment* de la herramienta, como se puede visualizar en la Figura 6.44. En la opción *New Diagram*, hay un ítem indicado como *Mobile UWE* donde están detallados todos nuestros modelos. Al seleccionar uno de ellos se crea el modelo seleccionado. Esta es otra forma que tiene el diseñador de crear modelos.

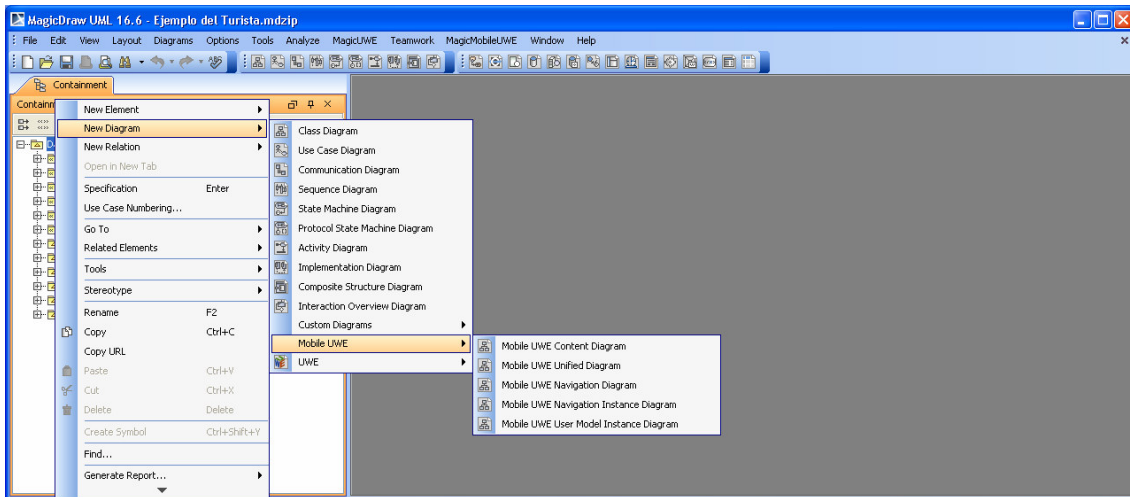


Figura 6.44: Creación de todos los diagramas del enfoque.

Algo similar sucede con las transformaciones, las mismas pueden ser realizadas por el diseñador no solamente usando el menú principal, sino también mediante el menú contextual del modelo, cuando el mismo es seleccionado desde la solapa *Containment* como se puede ver en la Figura 6.45. Para el Modelo de Contenido se pueden ver las dos transformaciones asociadas a este tipo de modelo.

En el caso que el diseñador eligiera realizar alguna de estas transformaciones, la misma se realiza sobre el modelo seleccionado sin necesidad de que el mismo esté abierto (como si lo requería el menú principal).

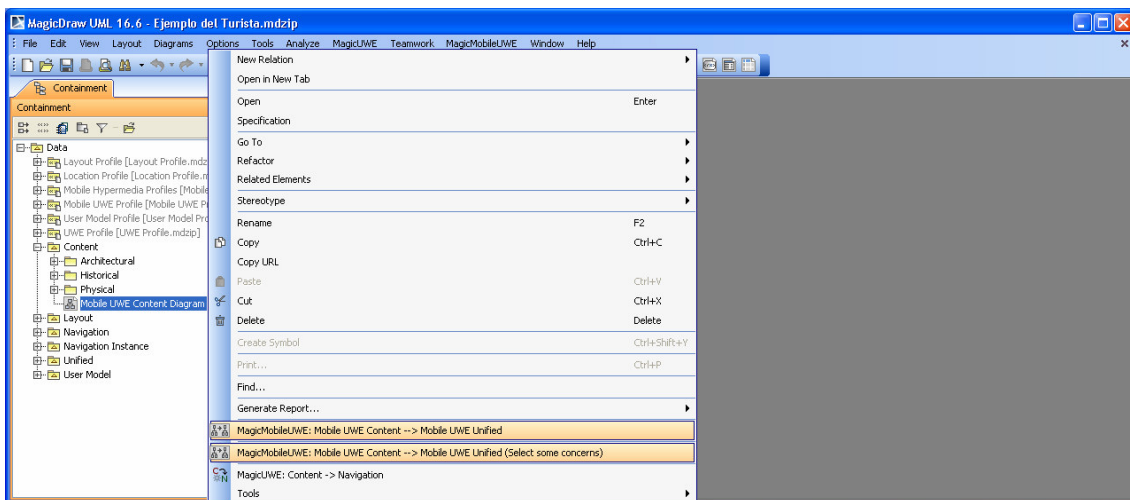


Figura 6.45: Transformaciones relacionadas al Modelo de Contenido.

Con el Modelo Unificado sucede algo similar, el menú contextual que aparece cuando un Modelo Unificado es seleccionado desde la solapa *Containment* se puede visualizar

en la Figura 6.46.

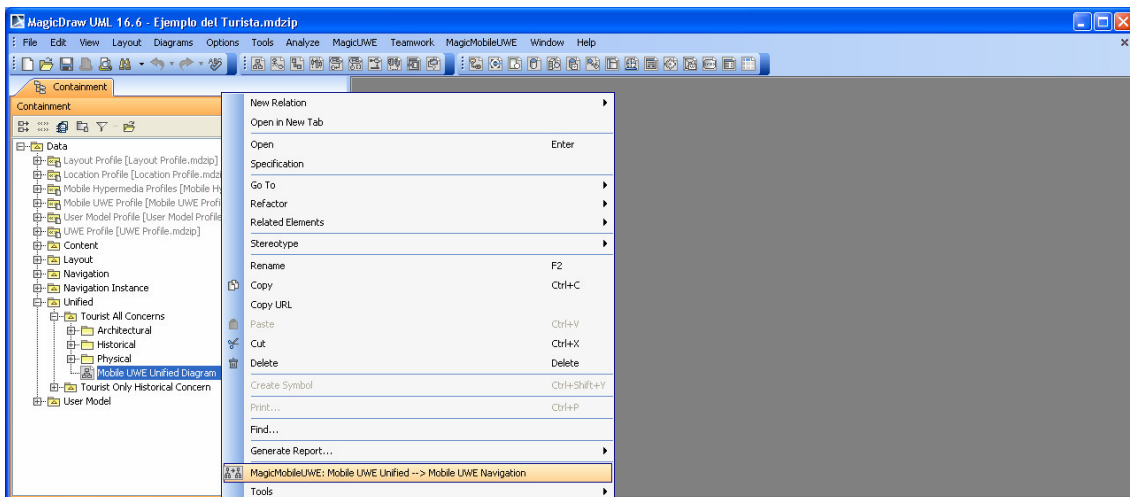


Figura 6.46: Transformación relacionada al Modelo Unificado.

Comparando las Figuras 6.45 y 6.46 se puede ver que las opciones de transformación varían según el modelo seleccionado, es decir, varían si es el Modelo Contenido o el Modelo Unificado. Para el resto de los modelos, no hay ninguna transformación asociada, por lo tanto no se muestra ninguna opción nueva.

Otra característica que brinda la herramienta, está relacionada con la propiedad de ubicación. Se mostró en la Sección 6.1 que se puede crear para una clase (del concern físico) una propiedad de ubicación. También se provee la posibilidad de definir una propiedad en las clases del concern físico y al seleccionar (la propiedad) aparece un menú contextual con la opción *MagicMobileUWE: Set Property as Location*⁵⁰ que permite convertir esa propiedad a una propiedad de ubicación (Figura 6.47).

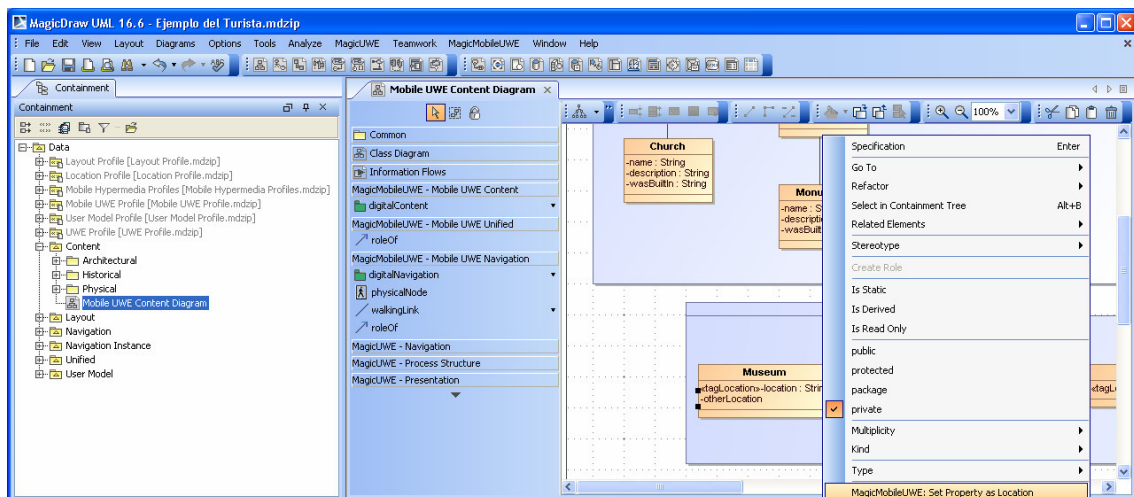


Figura 6.47: Setear una propiedad como de ubicación.

Cuando el diseñador elige esta opción, se le muestra una pantalla con los tres estereotipos que puede tener esta propiedad, y, al seleccionar uno de estos, se le agrega a la propiedad seleccionada el estereotipo elegido por el diseñador. De esta manera una propiedad, por ejemplo, llamada `OtherLocation` se transforma en una propiedad de ubicación como se muestra en la Figura 6.48. Además se puede observar que se pueden crear tantas propiedades de ubicación como sean necesarias

⁵⁰ Este menú sólo está disponible para los Modelos de Contenido.

para la aplicación. El diseñador debe tener en cuenta cómo interpretar la definición de más de una propiedad de ubicación, asociada a una clase determinada.

En el caso de elegir nuevamente la opción para una propiedad de ubicación a la clase, la misma se crea con el nombre `location` y se le agrega al final un número incremental. Para el ejemplo de la Figura 6.48 se puede ver la propiedad llamada `location1`.

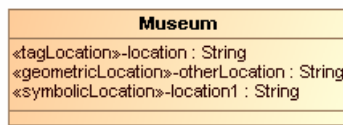


Figura 6.48: Creación de distintas propiedades de ubicación.

Como se mencionó en la Sección 6.4, el diseñador puede crear las operaciones de links caminables derivados usando la herramienta. En el caso de elegir más de una vez esta opción se crea la operación con el nombre `derivedWalkingLinks` y se le agrega al final un número incremental como se muestra en la Figura 6.49. En este ejemplo queda especificada la operación `derivedWalkingLinks` y `derivedWalkingLinks1`. Se permite la flexibilidad para crearlas a nivel de modelo, pero en la implementación el diseñador debe establecer bajo qué criterio se usa cada una.

Se puede apreciar que se crean dos operaciones, una con cada uno de los estereotipos disponibles. En el caso de la operación `derivedWalkingLinks1` se define con el estereotipo `<<onlyNearPhysicalCounterparts>>` y se le redefine el valor etiquetado `distance` por 300 (ya que por default tenía 500).

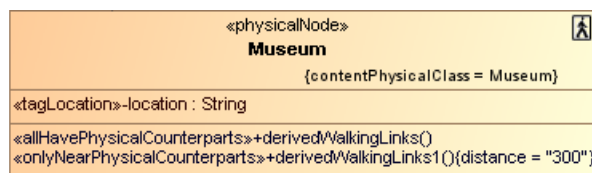


Figura 6.49: Creación de operaciones de links caminables derivados.

Como se mostró en el Modelo de Instancias Navegacional, se provee una función que completa las propiedades y los roles del diagrama acordes al Modelo Navegacional de base. Además, se provee otra función que completa el diagrama teniendo en cuenta también las relaciones de links (digitales y físicos) del Modelo Navegacional de base. Esta nueva opción se puede ver destacada en la Figura 6.50.

Supongamos que usamos el mismo ejemplo que se utilizó en la Sección 6.5 (donde se mostró la función que completaba sólo las propiedades y los roles). Para este caso se tenían tres instancias del nodo *Church*, una en cada concern (histórico, arquitectural y físico) con el nombre de *Basílica de San Pedro* y tres instancias del nodo *Monument* con el nombre *Panteón* (también en los tres concerns). Supongamos que agregamos una instancia nueva, que representa una instancia de *ArchitecturalStyle* llamada *Corintio*. En el Modelo Navegacional de base, los nodos *Monument* y *Church* en el concern arquitectural tienen definido un link al nodo *ArchitecturalStyle*. Al realizar la operación de completar indicada en la Figura 6.50 (al ejemplo de instancias planteado) se tiene como resultado la Figura 6.51. Se puede observar que se crearon las asociaciones estereotipadas como `<<roleOf>>` y se agregaron las propiedades acordes al nodo base de cada instancia (como así la función presentada en la Sección 6.5). Pero además se agregaron dos link digitales entre las instancias de *Monument* y *Church* (del concern arquitectural) y la instancia *Corintio*.

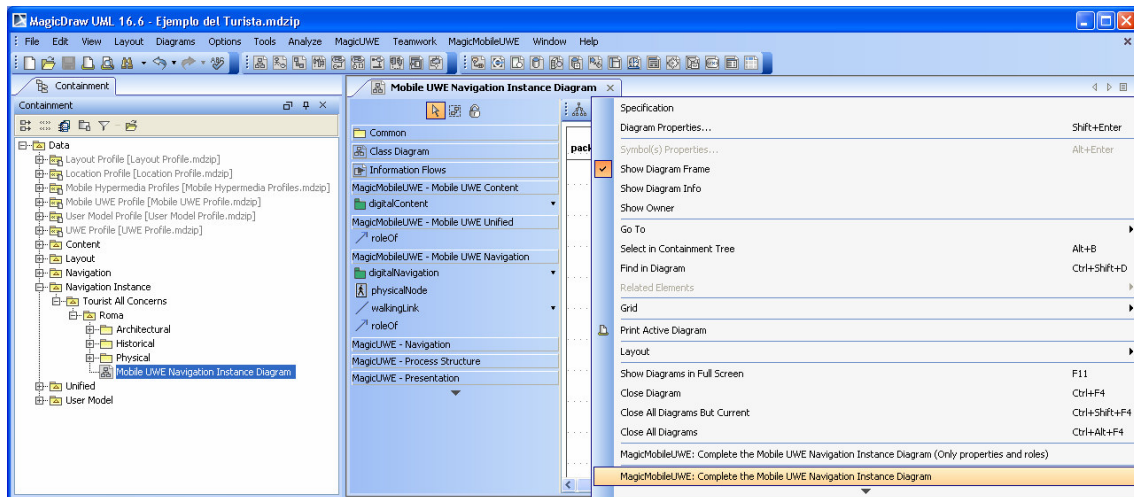


Figura 6.50: Otra opción para completar el Modelo de Instancias Navegacional.

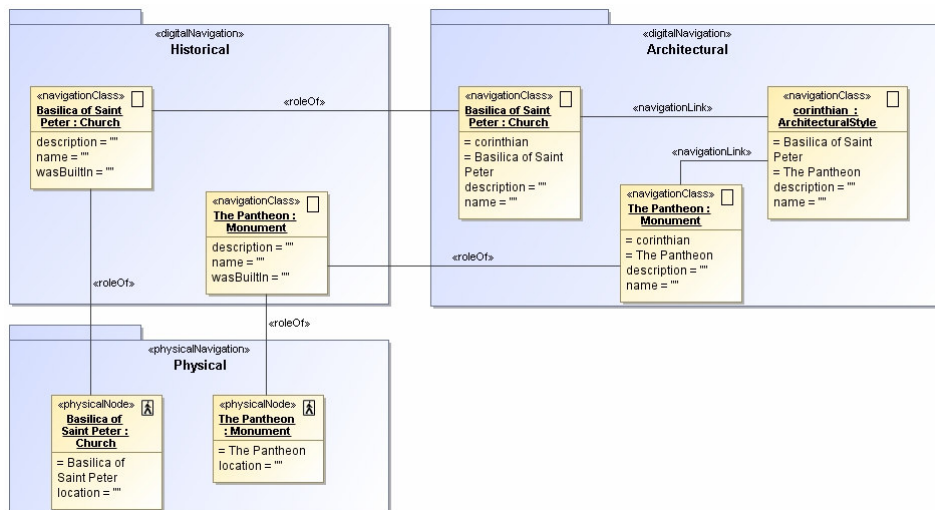


Figura 6.51: Resultado de completar el Modelo de Instancias Navegacional.

Esta función de completar el Modelo de Instancias Navegacional puede servir en algunos casos al diseñador. Pero puede no ser adecuada para algunas aplicaciones por la generación de links. Supongamos que existe un link entre dos nodos llamémoslos N_a y N_b , la función de completar toma todas las instancias del nodo N_a y las relaciona mediante links, con todas las instancias del nodo N_b . Puede pasar que a nivel de Modelo Navegacional esté definido un link entre dos nodos, pero a nivel de instancias, no es cierto que todas las instancias de ambos nodos se relacionen. Sólo las instancias que tengan sentido se relacionan mediante el link.

Esta situación debe ser considerada por el diseñador para ver cuáles de las dos funciones de completar disponibles se utiliza. Si el diseñador tiene un Modelo de Instancias Navegacional en el que tiene sentido que los links entre nodos se creen entre todas las instancias de los mismos, entonces va a usar la función descrita en esta sección, caso contrario va a usar la función descrita en la Sección 6.4.

6.9. Estereotipos creados para el funcionamiento de *MagicMobileUWE*

Como se mencionó en las Secciones 6.1 a la 6.7, para el funcionamiento de *MagicMobileUWE* se crearon estereotipos específicos para cada modelo. Logrando de esta manera agregarle a cada modelo la semántica correspondiente. Veamos como se

definen cada uno de estos estereotipos, indicando además, cuáles tienen en su definición valores etiquetados que son usados para guardar información, y de esta manera proveer funcionalidad específica al diseñador.

Para el Modelo de Contenido se creó el estereotipo `<<mobileUWEContentDiagram>>` como se muestra en la Figura 6.52. Para este estereotipo no fue necesario definir ningún valor etiquetado.



Figura 6.52: Estereotipo para el Modelo de Contenido.

Para el Modelo Unificado (Figura 6.53) se definió el estereotipo llamado `<<mobileUWEUnifiedDiagram>>`, el cual define el valor etiquetado `contentDiagram`. Este valor etiquetado permite guardar el Modelo de Contenido desde el cual fue derivado el Modelo Unificado, cuando se usó una transformación. En el caso que el Modelo Unificado no se haya creado desde una transformación, este valor etiquetado no tiene ningún Modelo de Contenido asignado.



Figura 6.53: Estereotipo para el Modelo Unificado.

Algo similar ocurre con el Modelo Navegacional, se creó el estereotipo `<<mobileUWENavigationDiagram>>` que define como valor etiquetado `unifiedDiagram` (como se visualiza en la Figura 6.54). Este valor etiquetado, es inicializado con un Modelo Unificado, sólo en el caso que el Modelo Navegacional haya sido generado mediante una transformación.

Se usa este estereotipo cuando se debe buscar los Modelos Navegacionales, para listarlos como posibles modelos a ser seleccionados como base de un Modelo de Instancias Navegacional, o para definirles una presentación.



Figura 6.54: Estereotipo para el Modelo Navegacional.

Como se muestra en la Figura 6.55 en el caso del Modelo de Instancia Navegacional se creó el estereotipo `<<mobileUWENavigationInstanceDiagram>>`.



Figura 6.55: Estereotipo para el Modelo de Instancias.

En la Figura 6.55 se puede visualizar, el estereotipo tiene dos valores etiquetados. Uno de ellos es `navigationDiagram`, el cual se completa con el Modelo Navegacional base que se quiere instanciar. Este valor etiquetado siempre debe tener un Modelo Navegacional seteado.

El otro valor etiquetado asociado al estereotipo es `wasComplete`, el cual se define inicialmente como falso, representando que dicho modelo todavía no fue completado.

Este valor etiquetado toma valor verdadero cuando el diseñador elige la opción de completar el Modelo de Instancias Navegacional. Se verifica el valor de esta etiqueta (*wasComplete*) para proveer, o no, el menú contextual con las opciones de completar el Modelo de Instancias Navegacional.

Para los Modelos de Presentación, se definió el estereotipo denominado `<<mobileUWELayoutDiagram>>` (ver Figura 6.56) que tiene un valor etiquetado `navigationDiagram`. Este valor etiquetado siempre debe tener el Modelo Navegacional base al cual se le está definiendo una presentación. Este estereotipo es usado para buscar todos los Modelos de Presentación a ser listados como posibles presentaciones que puede tener un Modelo de Instancias Navegacional, cuando se crea un Modelo del Usuario.



Figura 6.56: Estereotipo para el Modelo de Presentación.

Los Modelos del Usuario tienen el estereotipo `<<mobileUWEUserModelDiagram>>`. Este estereotipo no requiere definir ningún valor etiquetado en particular como se visualiza en la Figura 6.57.

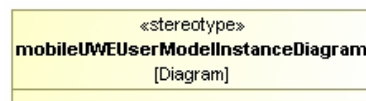


Figura 6.57: Estereotipo para el Modelo del Usuario.

Cada uno de los estereotipos especificados para cada uno de nuestros modelos, fue agregado en alguno de los perfiles presentados tanto en la Sección 4.2 como en las Secciones 5.7 y 5.8.

La Figura 6.58 muestra a qué perfil fue agregado cada estereotipo. Se puede apreciar que en el perfil *Mobile UWE* se agregaron los estereotipos relacionados a los Modelos de Contenido, Unificado, Navegacional y de Instancias Navegacional. El estereotipo del Modelo de Presentación se agregó al perfil *Layout* mientras que el estereotipo del Modelo del Usuario, se agregó al perfil *User Model*.

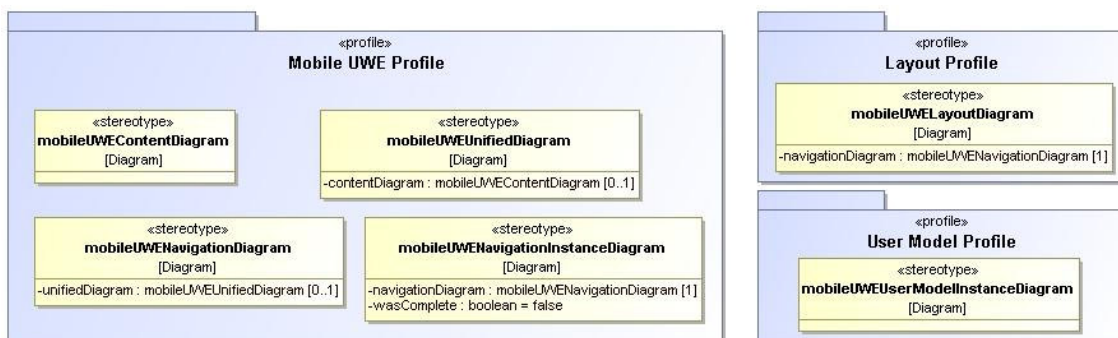


Figura 6.58: Estereotipos de los modelos en sus correspondientes perfiles.

En la Tabla 6.1 se muestran los estereotipos definidos para cada uno de los modelos, los mismos son definidos para el elemento *UML* llamado *Diagram* [UML Superstructure]. En la segunda columna, se detalla a qué modelo caracteriza el estereotipo.

Tabla 6.1: Estereotipos definidos para representar cada uno de nuestros modelos.

Estereotipo	Modelos que representa el estereotipo
mobileUWEContentDiagram	Modelo de Contenido
mobileUWEUnifedDiagram	Modelo Unificado
mobileUWENavigationDiagram	Modelo Navegacional
mobileUWENavigationInstanceDiagram	Modelo de Instancia Navegacional
mobileUWELayoutDiagram	Modelo de Presentación
mobileUWEUserModelInstanceDiagram	Modelo del Usuario

En las Secciones 6.2 a la 6.7 se definió cada modelo dentro de paquetes con nombres específicos. Este concepto se tomo de *UWE*, que define para su plugin *MagicUWE* paquetes para almacenar y agrupar los diferentes modelos.

Cada paquete en realidad es un elemento *Model* [UML Superstructure] de *UML*. En *UML*, este elemento (*Model*) representa un paquete que contiene específicamente modelos. Para que un elemento *Model* tenga la semántica de contener específicamente nuestros diferentes modelos, se definen estereotipos particulares para cada paquete *Model*.

MagicUWE tiene dos estereotipos⁵¹ para representar paquetes que almacenan sus modelos de contenido y navegacionales. Es decir, para el paquete de modelos llamado *Content* definen el estereotipo `<<contentModel>>` y para el paquete de modelos llamado *Navigation* definen el estereotipo `<<navigationModel>>`.

Para que la herramienta sea compatible con el plugin *MagicUWE* se usan los dos estereotipos de *Model* mencionados anteriormente para almacenar nuestros Modelos de Contenido y Navegacionales respectivamente. De esta manera, se logra que la herramienta utilice toda la funcionalidad definida por *MagicUWE* para sus Modelos Navegacionales.

Para nuestros otros cuatro modelos, se crearon cuatro estereotipos nuevos de *Model*. Para el paquete *Unified* que contiene los Modelos Unificados se definió el estereotipo `<<unifiedModel>>`. El paquete *Navigation Instance* que almacena los modelos de instancia se le creó el estereotipo `<<navigationInstanceModel>>`. Al paquete *Layout* (que almacena los Modelos de Presentación) se le creó el estereotipo `<<layoutModel>>`. Y por último, al paquete *User Model* que contiene los modelos del usuario se le creó el estereotipo `<<userModelInstance>>`.

En la Figura 6.59 se puede apreciar cómo es la visualización de cada uno de los paquetes de modelos con sus correspondientes estereotipos. En el caso de tener paquetes de modelos internos, estos también deben tener el estereotipo adecuado.

El buen funcionamiento de las operaciones, depende de tener bien estereotipados los paquetes de modelos, ya que mucha funcionalidad hace controles relacionados con los estereotipos de los modelos.

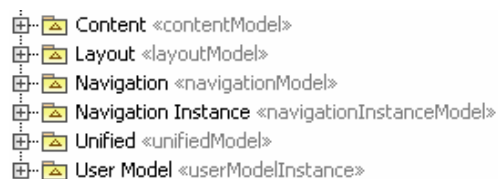


Figura 6.59: Estereotipos definidos para los paquetes de modelos.

⁵¹ *UWE* tiene definidos otros estereotipos de *Model* que no son mencionados ya que no son usados por la herramienta.

La solapa *Containment* permite elegir si mostrar o no los estereotipos de los elementos. Por una cuestión de no sobrecargar las figuras de esta sección con demasiada información, se optó por no mostrar el estereotipo asociado a cada paquete de modelos.

Los estereotipos creados de *Model* se agregaron en nuestros diferentes perfiles. En el perfil *Mobile UWE* se agregaron los estereotipos `<<unifiedModel>>` y `<<navigationInstanceModel>>`, en el perfil *Layout* se agregó el estereotipo `<<layoutModel>>` mientras que en el perfil *User Model* se agregó el estereotipo `<<userModelInstance>>`. En la Figura 6.60 se muestra estos cuatro estereotipos creados y los perfiles donde fueron agregados. Además se puede visualizar en la figura los dos estereotipos que se usan del perfil de *UWE*.

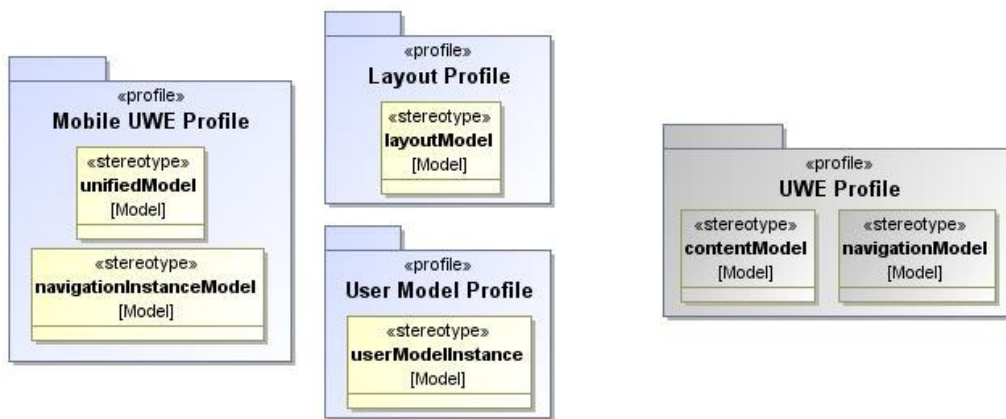


Figura 6.60: Estereotipos de los paquetes de modelos.

En la Tabla 6.2 se muestran los estereotipos definidos para cada uno de las paquetes que contienen nuestros modelos, los mismos son definidos para el elemento *UML* llamado *Model* [UML Superstructure]. En la segunda columna se detalla a que modelos contiene el paquete que usa el estereotipo y en la tercera columna quien provee el estereotipo, *UWE* o nuestro enfoque.

Tabla 6.2: Estereotipos usados para representar cada uno de los paquetes que contienen a nuestros modelos.

Estereotipo	Modelos que contiene el paquete	Provisto
contentModel	Modelo de Contenido	UWE
unifiedModel	Modelo Unificado	Nuestro enfoque
navigationModel	Modelo Navegacional	UWE
navigationInstanceModel	Modelo de Instancia Navegacional	Nuestro enfoque
layoutModel	Modelo de Presentación	Nuestro enfoque
userModelInstance	Modelo del Usuario	Nuestro enfoque

6.10. Resumen

En este capítulo se presentó la funcionalidad de la herramienta *MagicMobileUWE*, la cual permite el desarrollo dirigido por modelos para aplicaciones de HM.

La herramienta permite además de la creación de cada uno de los modelos de nuestro enfoque, realizar las siguientes transformaciones:

- *Mobile UWE Content* → *Mobile UWE Unified*

- *Mobile UWE Content* → *Mobile UWE Unified* (Selected some concerns)
- *Mobile UWE Unified* → *Mobile UWE Navigation*

Además, cuenta con diferentes menús contextuales que agilizan la tarea del diseñador. En la Tabla 6.3 se mencionan todos los menús de la herramienta, indicando el nombre del menú, el elemento para el cual esta disponible el menú (usando el estereotipo del mismo) y en qué modelo esta disponible.

Tabla 6.3: Menús definidos por *MagicMobileUWE*

Menú	Estereotipos para el cual esta disponible el menú junto con el correspondiente modelo
<i>MagicMobileUWE: Create Location Property</i>	<<physicalContent>> Modelo de Contenido
<i>MagicMobileUWE: Set Location as Property</i>	<<physicalContent>> Modelo de Contenido
<i>MagicMobileUWE: Create Derived Walking Link Operation</i>	<<physicalNode>> Modelo Navegacional
<i>MagicMobileUWE: Create an Instance of navigationClass</i>	<<digitalNavigation>> Modelo de Instancias Navegacional
<i>MagicMobileUWE: Create an Instance of physicalNode</i>	<<physicalNavigation>> Modelo de Instancias Navegacional
<i>MagicMobileUWE: Set Classifier</i>	<<navigationClass>> y <<physicalNode>> Modelo de Instancias Navegacional
<i>MagicMobileUWE: Complete the Mobile UWE Instance Navigation Diagram (only properties and roleOf)</i>	<<magicUWENavigationInstanceDiagram>> Modelo de Instancias Navegacional
<i>MagicMobileUWE: Complete the Mobile UWE Instance Navigation Diagram</i>	<<magicUWENavigationInstanceDiagram>> Modelo de Instancias Navegacional
<i>MagicMobileUWE: Set Stylesheet</i>	<<navigationClassLayout>> y <<physicalNodeLayout>> Modelo de Presentación

Nuestro desarrollo dirigido por modelos cumple con uno de los puntos básicos asociados a este tipo de desarrollo, como es automatizar la tarea del diseñador y proveer funcionalidad para agilizar la tarea de crear aplicaciones de HM.

En la Figura 6.60 se puede apreciar un esquema general, que muestra un proyecto de *MagicDraw*, el cual contiene los perfiles (*UWE* y *Mobile Hypermedia*) que utiliza cada proyecto que representa una aplicación de HM y además, se indica los modelos que puede contener el proyecto. Se puede observar que, en particular para el Modelo de Presentación, se destaca la implementación presentada (modelo basado en *XSLT*). Lo mismo ocurre con el Modelo del Usuario, se especifica el modelo (del Usuario) básico y el que considera estados.

Se puede ver en la Figura 6.61 que las relaciones entre nuestros modelos pueden estar dadas porque:

- De un modelo a otro se llega aplicando una transformación, como es el caso de las dos transformaciones entre el Modelo de Contenido y el Modelo Unificado, o la transformación entre el Modelo Unificado y el Modelo Navegacional.
- Un modelo es la instanciación de otro, como en el caso del Modelo de Instancias Navegacional.

- Un modelo se define en base a otro, es decir necesita conocer a otro modelo para que tenga sentido su creación. Como por ejemplo, del Modelo de Presentación (que necesita el Modelo Navegacional base) o el Modelo del Usuario (que necesita el Modelo de Instancias Navegacional y el Modelo de Presentación).

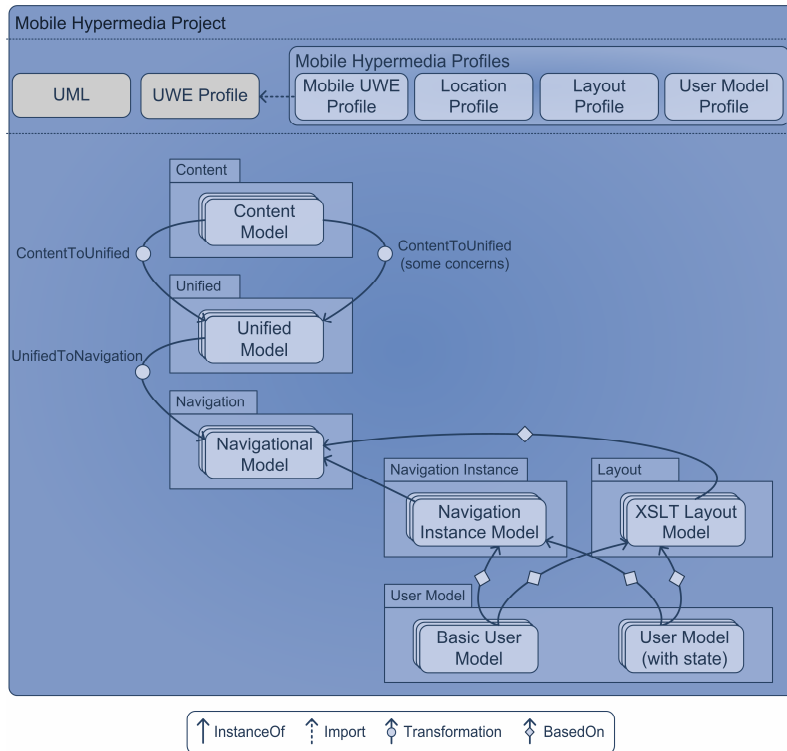


Figura 6.61: Proyecto que representa los modelos creados en este capítulo.

7. CASOS DE ESTUDIO

En este capítulo se presentan tres ejemplos de aplicaciones móviles con diferentes características, mostrando como, mediante el uso de la herramienta *MagicMobileUWE*, se puede aplicar para crear una aplicación de HM.

Uno de los ejemplos es la típica aplicación de asistencia turística, y para ello se va a continuar con el ejemplo presentado en los Capítulos 5 y 6. Los otros dos ejemplos están basados en el concepto de *Mobile Urban Drama* [Hansen et al., 2008].

Se describen las características particulares de cada uno de estos casos, comparándolos entre sí. Además, se especifica como sus características son modeladas usando el enfoque.

7.1. Asistencia Turística

En el Capítulo 5 se mostró un ejemplo de modelado de una aplicación turística. Si bien este ejemplo fue presentado para la ciudad de Roma, los Modelo de Contenido y Unificado pueden ser tomados como base para desarrollar una aplicación del mismo tipo para cualquier otra ciudad. En particular, en este capítulo el caso de estudio presentado representa una aplicación para la ciudad de *La Plata*.

A partir del Modelo Unificado presentado en la Figura 5.2, se realiza entonces una transformación para obtener un nuevo Modelo Navegacional que represente a la ciudad de *La Plata*. De esta manera, se muestra cómo usando un mismo Modelo Unificado, se pueden obtener distintos Modelos Navegacionales. En la Figura 7.1 se puede apreciar la organización que tenían en el Capítulo 6 los paquetes *Unified* y *Navigation*, y se destaca el Modelo Unificado que se usa para obtener el Modelo Navegacional para la ciudad de *La Plata*.

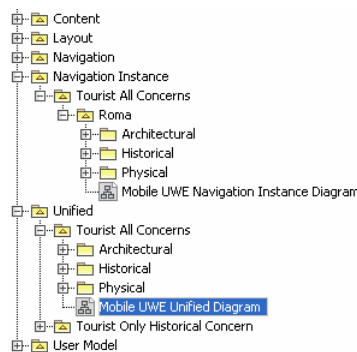


Figura 7.1: Organización de los modelos creados en la Sección 6.

En la Figura 7.2⁵² se puede apreciar el nuevo Modelo Navegacional creado (*La Plata Mobile UWE Navigation Diagram*) y además el que ya existía para la ciudad de *Roma*.

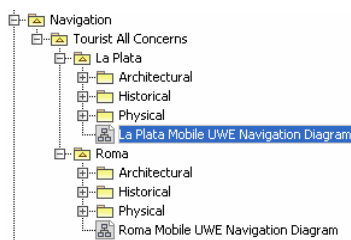


Figura 7.2: Nuevo Modelo Navegacional creado.

⁵² Para que no haya confusiones de nombres se renombró el Modelo Navegacional que existía de *Roma* como *Roma Mobile UWE Navigation Diagram*.

Como se mencionó anteriormente (en el Capítulo 6), la transformación del Modelo Unificado al Modelo Navegacional, pasa toda la información del primer modelo al segundo acorde a las reglas que se describieron en la Sección 5.5. Esto quiere decir que, el modelo resultante de la transformación, puede necesitar ser refinado por el diseñador para que se ajuste a las características de la nueva aplicación. El Modelo Navegacional generado se puede visualizar en la Figura 7.3.

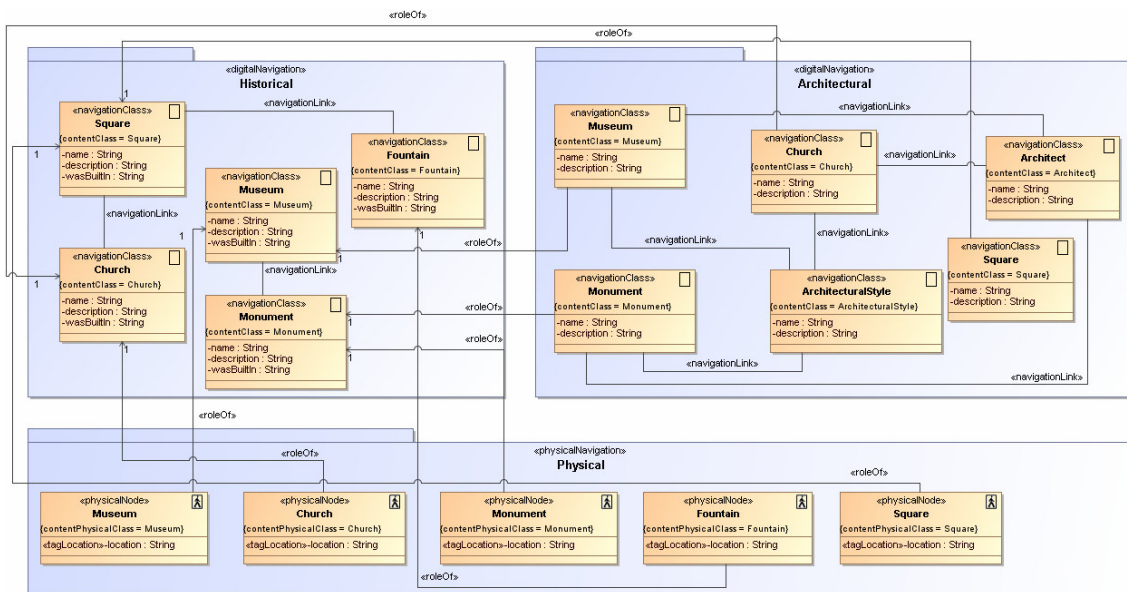


Figura 7.3: Modelo Navegacional generado por la transformación.

Supongamos que el diseñador decide refinar el modelo de la Figura 7.3 sacando algunos nodos y links que no son de interés para esta aplicación particular. También puede agregar un link digital en el concern histórico entre los nodos *Church* y *Square*, y un link caminable entre los nodos *Church* y *Museum*.

Además, puede generar para los nodos físicos la operación de `derivedWalkingLinks` estereotipada como `<<allHavePhysicalCounterparts>>` (el diseñador crea esta operación mediante el uso del menú contextual *MagicMobileUWE: Create Derived Walking Link Operation*). El Modelo Navegacional refinado, con los cambios anteriormente descritos, se puede apreciar en la Figura 7.4.

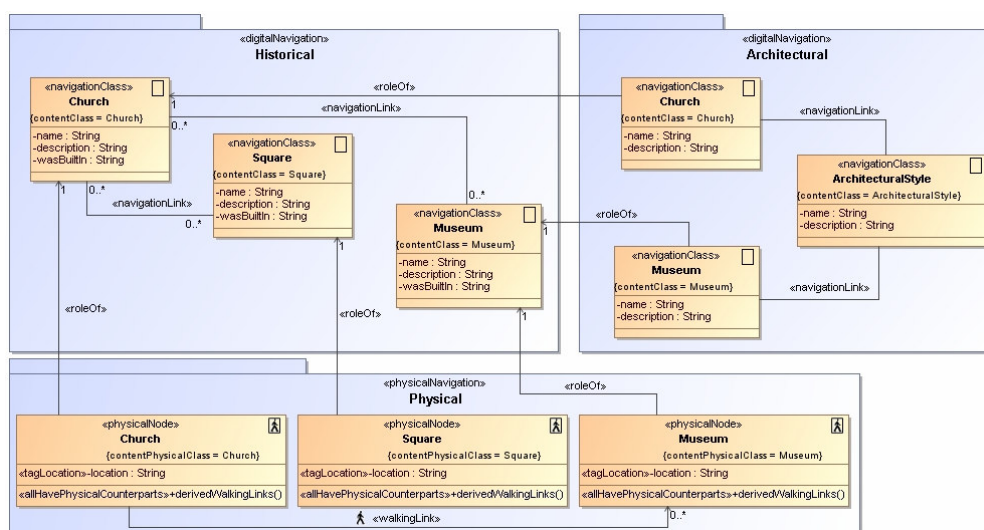


Figura 7.4: Modelo Navegacional refinado.

Todas las decisiones relacionadas al refinamiento del Modelo Navegacional, deben ser

establecidos por el diseñador y no se pueden resolver en el momento de la transformación. Sólo el diseñador conoce el dominio propio de la aplicación, y por lo tanto, los nodos que deben ser mostrados a los usuarios y los links que deben estar disponibles.

Supongamos que la aplicación de *La Plata* está pensada para ser usada por dos tipos de perfiles de usuarios diferentes, como por ejemplo, un turista y un historiador. Si bien las clases de nodos que se quiere tener para ambos perfiles son las mismas, el diseñador quiere que cada perfil reciba información diferente. Es decir, los usuarios con diferente perfil, reciben diferente información del mismo objeto físico. Para poder representar la situación descrita (perfiles de usuarios diferentes), el diseñador debe definir dos Modelos de Instancias Navegacional diferentes. Cada una de las instancias debe definir qué información se muestra para cada Punto de Interés. Como resultado se van a tener, a partir del mismo Modelo Navegacional, dos Modelos de Instancias Navegacional. Estos modelos se crean usando la opción que permite crear un nuevo Modelo de Instancias Navegacional. Al elegir esta opción, se muestra una lista con los Modelos Navegacionales que se pueden instanciar, como se puede observar en la Figura 7.5. Se puede apreciar que se listan los Modelos Navegacionales de *Roma* y de *La Plata*.

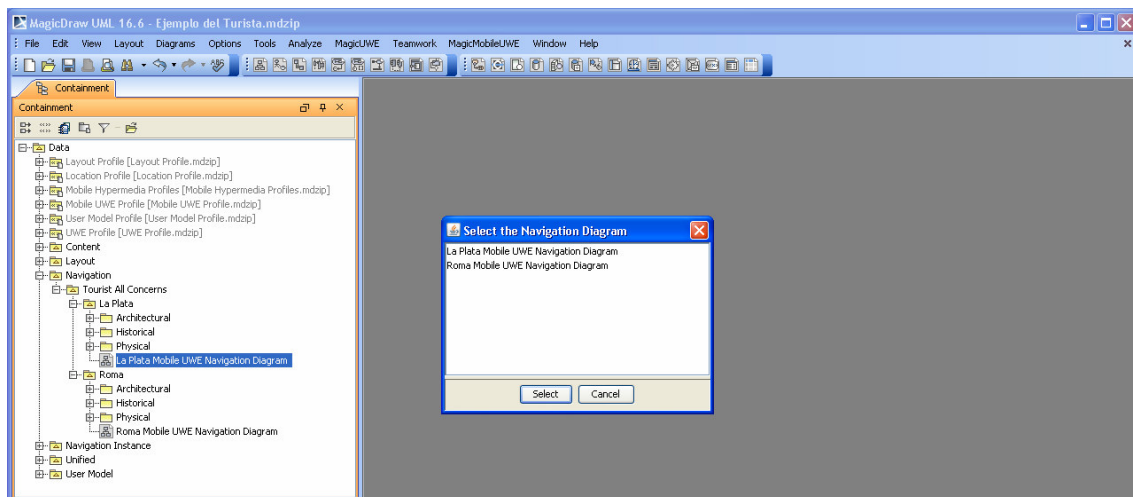


Figura 7.5: Elegir el modelo de base para crear el Modelo de Instancias Navegacional.

La Figura 7.6 muestra como cada uno de los modelos creados, se agrupó en diferentes paquetes, teniendo así una mejor organización. Ambos modelos tienen definidos los mismos concerns, ya que ambos se crearon a partir del mismo Modelo Navegacional. Por esta razón, es importante que el diseñador haga una buena organización de paquetes para cada modelo junto a sus elementos, ya que muchas veces se pueden repetir los nombres de los concerns entre los diferentes modelos. Es decir, si se tienen, todos los modelos en un mismo nivel no se pueden determinar a que modelo pertenece cada concern.

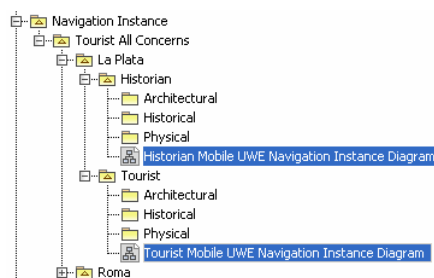


Figura 7.6: Los dos Modelos de Instancias Navegacional creados.

En la Figura 7.7 se pueden apreciar los dos Modelos de Instancias Navegacional creados, donde se muestran las instancias de los nodos pero sin información o links, a modo de empezar a ver las diferencias entre ambos modelos. La Figura 7.7.a muestra el Modelo de Instancias Navegacional creado para un turista mientras que la Figura 7.7.b muestra el Modelo de Instancias Navegacional para un historiador. Cada uno de los modelos define sus propias instancias para cada uno de los concerns.

Se puede apreciar que el historiador no cuenta con ninguna instancia en el concern arquitectural, pero tiene más instancias en el concern histórico, ya que es el foco de su perfil. Para el turista se puede ver que cuenta con información histórica y arquitectural. También se puede observar que hay diferencias en los nodos físicos instanciados, el historiador tiene un lugar más para recorrer.

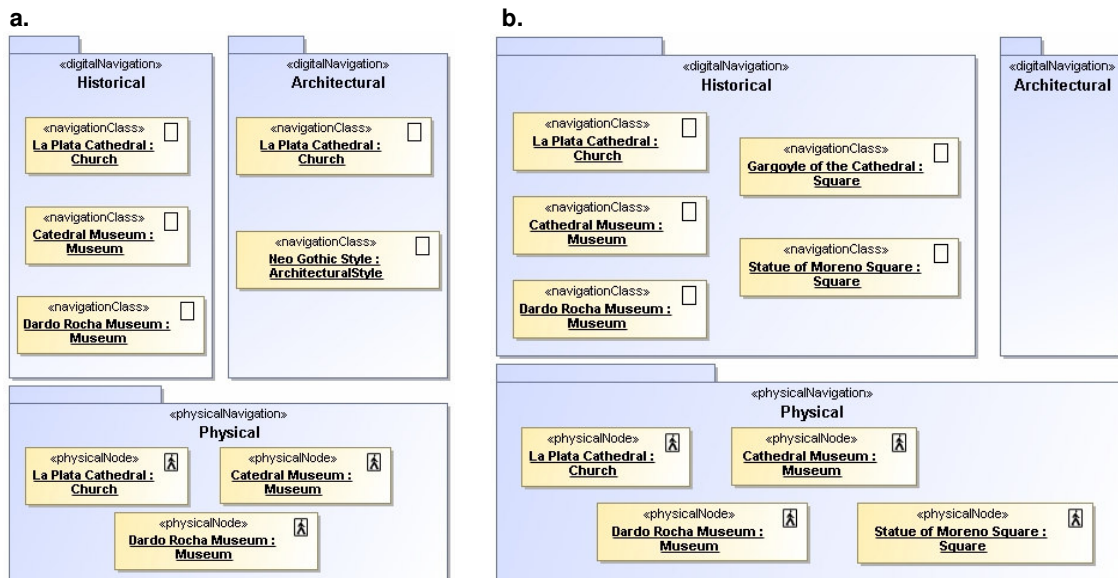


Figura 7.7: Los dos Modelos de Instancias Navegacional con algunas instancias creadas.

Puede pasar que estos modelos definan deferentes instancias para los mismos nodos, pero cada modelo tiene definido qué información se muestra y sus propios links.

En la Figura 7.8 se puede apreciar cómo se define, para el modelo del turista, la *Catedral de La Plata* (en los distintos concerns) con sus links. La figura muestra también cómo el usuario visualiza en su dispositivo la vista de información, como así también la vista del mapa con el link caminable predefinido al *Museo de la Catedral* y el link caminable derivado al *Museo Dardo Rocha*.

El turista tiene la posibilidad de cambiar de concern; esto es posible porque el Modelo de Instancias Navegacional tiene definido para la *Catedral de La Plata* también información en el concern arquitectural.

El Modelo Navegacional de base (Figura 7.4) tiene definida la operación `derivedWalkingLinks` estereotipada como `<<allHavePhysicalCounterparts>>`. Esto quiere decir, que todos aquellos links digitales (del nodo digital actual) que tienen contraparte física, son considerados como posibles links caminables (ya que puede pasar que estén incluidos como links caminables predefinidos y en ese caso no se agregan como links caminables derivados).

En el caso de la *Catedral de La Plata*, se tienen dos links digitales: uno al *Museo de la Catedral* y otro al *Museo Dardo Rocha*. Ambos targets tienen su contraparte física, por lo tanto pueden ser considerados para derivar links caminables. Pero el nodo físico de la *Catedral de La Plata* ya cuenta con un link caminable predefinido al *Museo de la Catedral*, por lo tanto, no se agrega un link caminable derivado a este museo. Es decir, como resultado de la operación se crea sólo un link caminable derivado al *Museo Dardo Rocha* como se visualiza en la última pantalla como *Suggestions*.

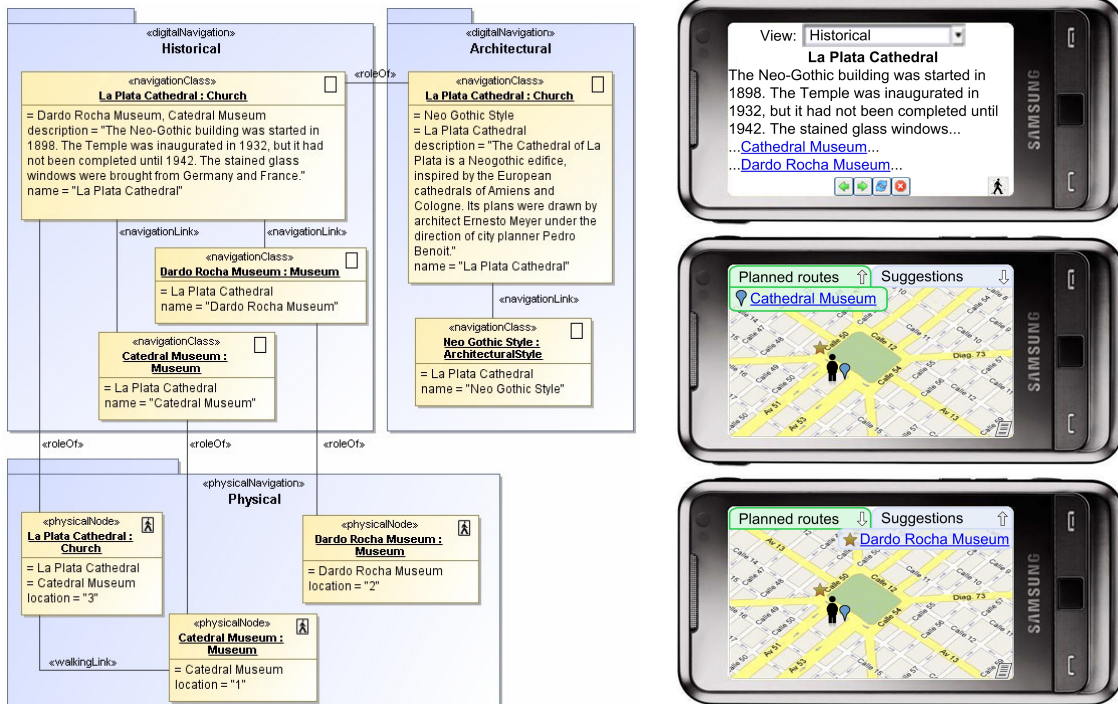


Figura 7.8: Un turista está viendo información histórica de la *Catedral de La Plata*.

Veamos en la Figura 7.9 cómo se instancia en el modelo del historiador la *Catedral de La Plata*. Se puede apreciar que la información digital y links que recibe el historiador varían respecto del turista.

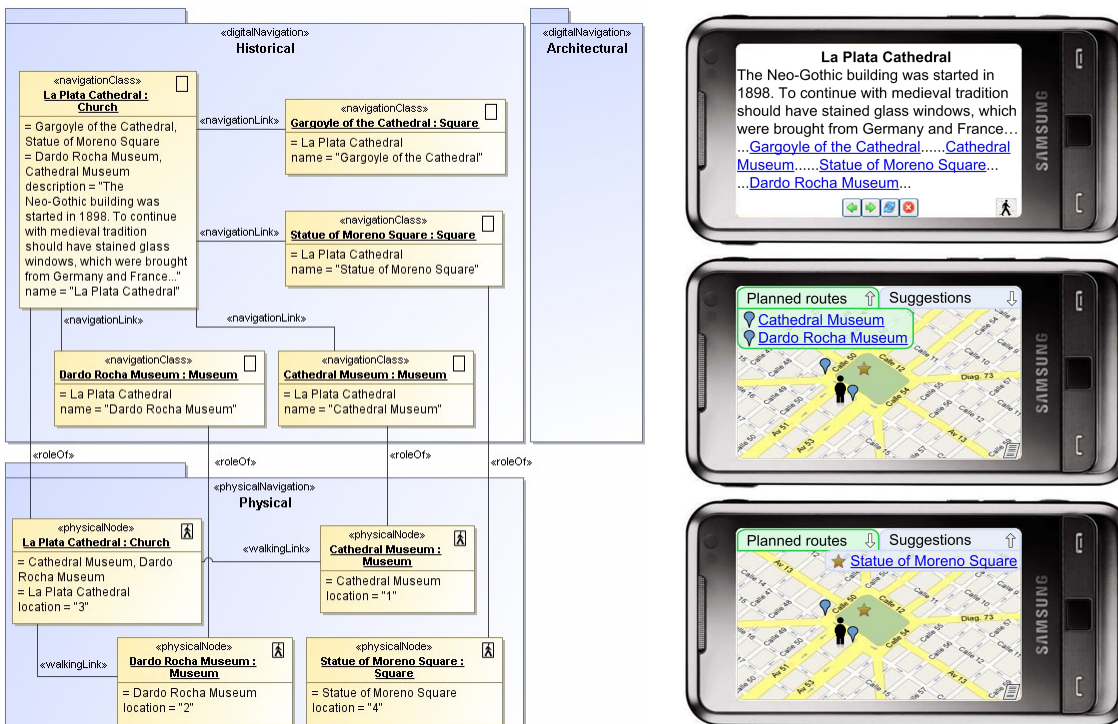


Figura 7.9: Un historiador está viendo información histórica de la *Catedral de La Plata*.

Cuando un historiador es sentido por la *Catedral de La Plata* puede visualizar su información digital, junto a cuatro links digitales (*Gárgolas de la Catedral*, *Museo de la Catedral*, *Estatua de la Plaza Moreno* y *Museo Dardo Rocha*). El historiador sólo va a tener información digital de la *Catedral*, por lo tanto no va a tener la posibilidad de

cambiar de concern como en el caso anterior. Esto es una consecuencia de cómo se definió su Modelo de Instancias Navegacional, que hace foco solamente en el concern histórico. Al cambiar a la vista del mapa, ve dos links caminables predefinidos, uno al *Museo de la Catedral* y otro al *Museo Dardo Rocha*.

La operación para calcular es la misma que se uso para el modelo del turista. Esta operación está definida en cada nodo del Modelo Navegacional y ambos Modelos de Instancias Navegacional tienen el mismo modelo de base.

La instancia de la *Catedral* tiene cuatro links digitales, de los cuales tres tienen contraparte física: el *Museo de la Catedral*, la *Estatua de la Plaza Moreno* y el *Museo Dardo Rocha*. Pero dos de estos ya están incluidos como links caminables predefinidos (el *Museo de la Catedral* y el *Museo Dardo Rocha*), por lo tanto la operación de links caminables derivados da como resultado el link caminable derivado a la *Estatua de la Plaza Moreno* como se visualiza en la última pantalla como *Suggestions*.

Comparando ambos modelos (Figuras 7.8 y 7.9) se puede ver que los nodos físicos tienen la misma forma de identificar la posición de los Puntos de Interés, en ambos casos se usa el mismo valor etiquetado. Es decir, se puede ver que los mismos objetos físicos se etiquetaron con los mismos números en ambos Modelos de Instancias Navegacional. En la aplicación real se puede resolver usando un código de barra y dependiendo del perfil del usuario que lo lee, se muestra la información pertinente. También se puede tener valores de ubicación distintos para cada modelo.

A partir de las Figuras 7.8 y 7.9 se pudo apreciar cómo se logra crear dos Modelos de Instancias Navegacional, a partir del mismo Modelo Navegacional y cómo dos usuarios con perfiles diferentes (turista e historiador) ven un mismo Punto de Interés con información acorde a su perfil.

En el caso de la operación de links caminables derivados, si bien la operación se define a nivel de nodo, es a nivel de instancias que se hace el cálculo, es decir, su resultado dependerá de los datos de cada instancia.

Se pudo ver en el ejemplo anterior, cómo instanciar el nodo *Cathedral* (del Modelo Navegacional) en dos Modelos de Instancias Navegacional diferentes, donde cada instancia tenía diferentes links haciendo variar el resultado de la operación `derivedWalkingLinks`.

Respecto del Modelo de Presentación para el Modelo Navegacional, en la Figura 7.10⁵³ se puede ver el nuevo modelo creado dentro del paquete *La Plata*. Para cada nodo se define una presentación particular usando la herramienta, de la misma manera que se había indicado en la Sección 6.5 para la aplicación de la ciudad de *Roma*.

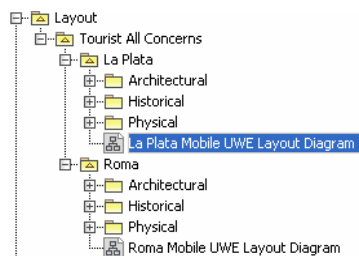


Figura 7.10: Modelo de Presentación creado.

En este ejemplo, como definimos un único Modelo de Presentación, tanto el turista como el historiador ven los nodos con la misma presentación, pero cada uno con la información especificada en su Modelo de Instancias Navegacional.

⁵³ Para que no haya confusiones de nombres se renombro el Modelo de Presentación que existía de *Roma* como *Roma Mobile UWE Layout Diagram*.

Para indicar que cada perfil tiene su propio Modelo de Instancias Navegacional, hay que definir dos Modelos del Usuario. Para esto se usa la opción crear un Modelo del Usuario de la herramienta y se muestra una pantalla como se visualiza en la Figura 7.11. En la misma se listan todos los posibles Modelos de Instancias Navegacional que tiene el proyecto, y el diseñador elige uno de ellos para crear el Modelo del Usuario.

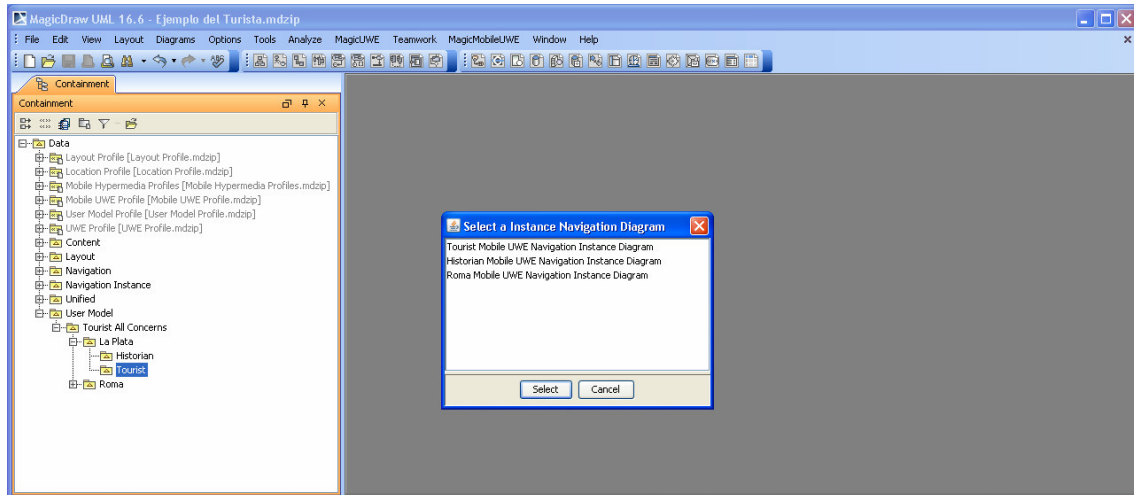


Figura 7.11: Elegir un Modelo de Instancias Navegacional para la creación del Modelo del Usuario.

Una vez elegido el Modelo de Instancias Navegacional del turista o del historiador, aparece una pantalla como se muestra en la Figura 7.12. Se puede ver que se listan todos los Modelos de Presentación acordes al Modelo Navegacional que tiene de base el Modelo de Instancias Navegacional elegido. En este caso sólo hay creado un Modelo de Presentación.

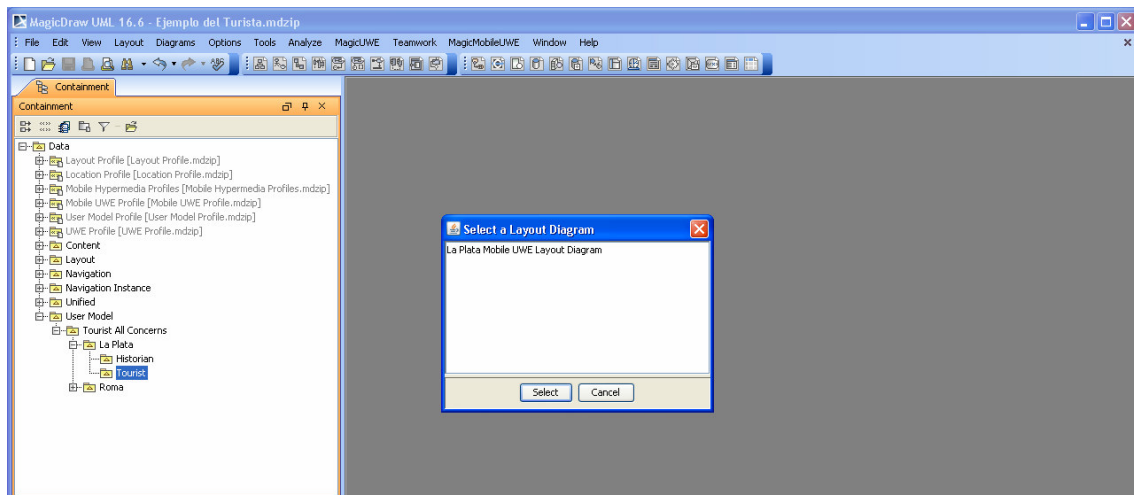


Figura 7.12: Elegir un Modelo de Presentación para la creación del Modelo del Usuario.

En las Figuras 7.13 y 7.14 se pueden apreciar los dos Modelos del Usuario creados. El Modelo del Usuario de la Figura 7.13 es el que se corresponde con un perfil de turista mientras que el de la Figura 7.14 se corresponde con el perfil del historiador.

Con las Figuras 7.13 y 7.14 se puede apreciar como se crean distintos Modelos del Usuario que se basan en el mismo Modelo Navegacional con la misma presentación (*ManagerView* tiene el mismo Modelo de Presentación) pero cada uno tiene un Modelo de Instancias Navegacional acorde a su perfil.



Figura 7.13: Modelo del Usuario para el turista.

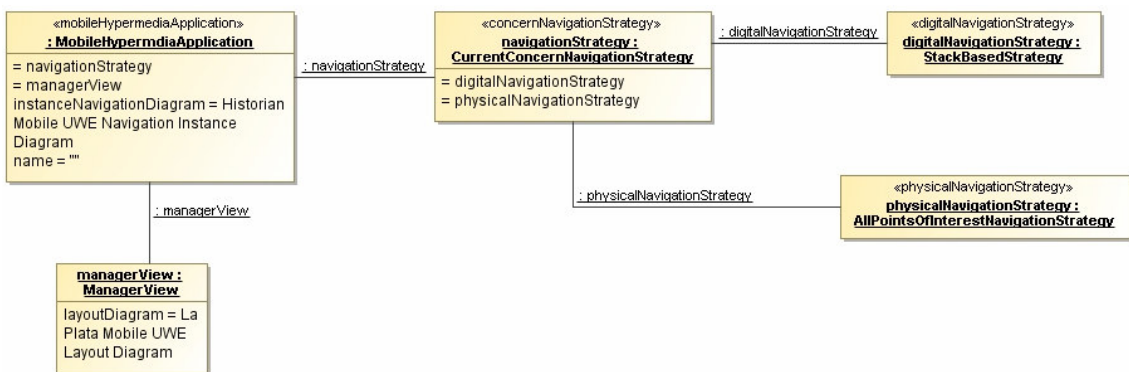


Figura 7.14: Modelo del Usuario para el historiador.

Se puede apreciar en las Figuras 7.13 y 7.14 que ambos modelos se crean con las estrategias navegacionales que define la herramienta por default. Cabe recordar que esta asignación de estrategias se hace automáticamente al crearse el Modelo del Usuario.

En esta sección se mostró como crear otro Modelo Navegacional a partir del mismo Modelo Unificado, logrando tener dos Modelos Navegacionales uno para la aplicación turística de *Roma* y otra para la aplicación de *La Plata*. Además, se mostró como a partir de un mismo Modelo Navegacional se pueden crear dos Modelos de Instancias Navegacional, uno para un turista y otro para un historiador. Logrando luego tener dos Modelos del Usuario diferentes ambos con una misma presentación asociada.

7.2. Ejemplos de *Mobile Urban Drama*

El término *Mobile Urban Drama* surge en [Hansen et al., 2008] y se utiliza para identificar aquellas aplicaciones móviles en donde el usuario toma el rol de actor de un determinado cuento o narración (storytelling). Este tipo de aplicaciones, se desarrolla en lugares abiertos donde el usuario se mueve para obtener la información referente a la aplicación. Los usuarios van recibiendo información en distintos lugares del mundo real y acorde a esa información realizan alguna tarea, o simplemente se informan sobre algún tema. Luego, la aplicación le indica al usuario cuál es el siguiente lugar a donde se debe dirigir para continuar con la historia. Se puede pensar este tipo de aplicaciones como un grafo de lugares donde en cada lugar el usuario recibe información.

Este tipo de aplicaciones tiene todas las características para catalogarse como una aplicación de HM. Por un lado se tiene la información que recibe el usuario en cada

lugar, la cual puede tener links (digitales) a otra información y además se tiene el concepto de *concern físico* que es donde se tiene representado los lugares del mundo real donde el usuario recibe la información. Estos lugares además están relacionados entre si (links caminables) para que el usuario pueda seguir el cuento o narración.

Como las aplicaciones de *Mobile Urban Drama* cuentan con las características de las aplicaciones de HM, podemos usar el enfoque para representarlas. En esta sección se muestra cómo se modelan estas aplicaciones. En particular, nos focalizaremos en mostrar cómo modelar dos aplicaciones con dominios con características distintas. Uno de ellos es un juego educativo y el otro una obra de teatro.

Veamos la descripción de un juego educativo simple (en la Sección 7.3 se describe cómo se puede volver más compleja esta aplicación). Cada alumno cuenta con un dispositivo móvil y a mediada que el alumno va recorriendo el mundo real, recibe preguntas sobre algún tema en particular. Estas preguntas están inasociadas a lugares determinados, acorde a cada juego, y el alumno la obtiene mediante la lectura de un código de barra.

Cuando el alumno está en un determinado lugar recibe (como resultado de leer un código de barra) una pregunta con una serie de posibles opciones de respuesta. Cada pregunta puede tener más de una opción correcta. Cuando el alumno responde la pregunta, el sistema evalúa esta respuesta, y le muestra los posibles lugares a donde se puede dirigir para continua el juego.

Las preguntas pueden tener disponible links a temas relacionados, para que el alumno pueda profundizar algún tema y de esta manera poder responder con más precisión. También, pueden ser referidas a la observación de alguna característica del ambiente donde esta ubicada la pregunta. Es decir, que el alumno tenga que realizar pruebas de observación del ambiente que lo rodea para luego responder la pregunta.

La finalidad del juego es brindarle al alumno la posibilidad de “aprender” fuera del aula y con el incentivo de poder terminar el juego. Las preguntas pueden estar planteadas en forma aislada o tener un hilo conductor (por ejemplo: una historia) que permita al alumno involucrarse más con el juego.

La obra de teatro consiste en un juego donde un usuario, con su dispositivo juega el papel de uno de los personajes de la historia. Esta historia se desarrolla en un lugar determinado que delimita las posibles escenas que tiene la obra (por ahora se describe un juego simple y en la Sección 7.3 se muestra cómo se puede volver más complejo). El usuario va recibiendo (como resultado, por ejemplo, de leer un código de barra) información en los diferentes lugares que le ayudan a saber cómo debe actuar ante determinadas situación, en cada momento de la historia. También pueden recibir preguntas que debe ir contestando.

Este tipo de aplicación tiene la particularidad de tener actores reales que interactúan con el usuario. Es decir, el usuario recibe información en su dispositivo que lo ayudan a seguir la dinámica de la obra y saber cómo debe comportarse en cada escena de la misma. El usuario pasa a ser un personaje más dentro de la obra involucrándose en forma activa en la misma.

La característica distintiva respecto de otras obras de teatros es que no tiene un guión definido sino que los actores tienen un esquema de situaciones que debe pasar en la obra y deben improvisar acorde a cómo se comporte el usuario recorriendo, además, el mundo real. Si bien el usuario recibe información de actividades que debe realizar, hay otros factores que son dinámicos, acordes a cómo reacciona cada usuario ante determinadas situaciones.

En la Figura 7.15 se puede apreciar un Modelo de Contenido para representar los conceptos generales básicos de las aplicaciones de *Mobile Urban Drama*. A partir de este modelo luego se creen las aplicaciones del juego educativo y la obra de teatro.

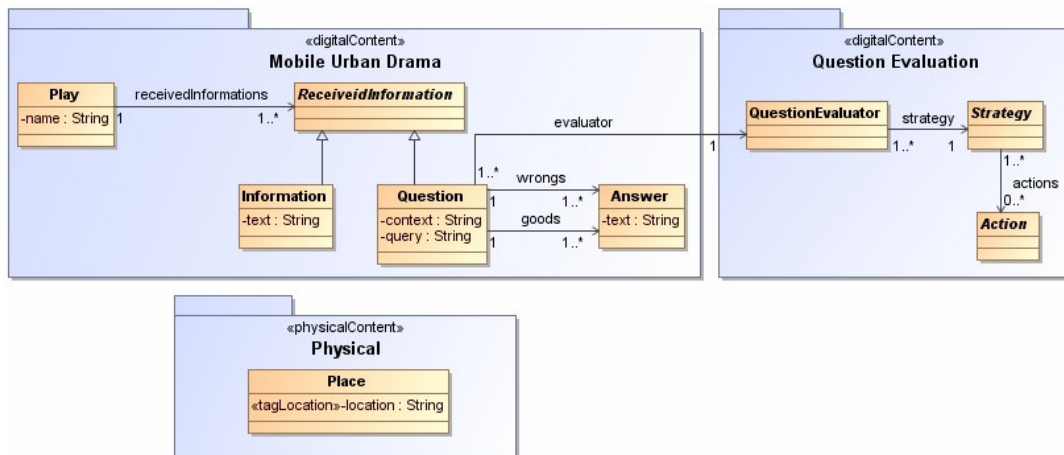


Figura 7.15: Modelo de Contenido para *Mobile Urban Drama*.

Por un lado se tiene el concern llamado *Mobile Urban Drama* donde se representan las clases relacionadas con la información a mostrar por la aplicación. *Play* es la clase que contiene toda la información relacionada a la aplicación planteada. Se define la información que recibe el usuario con la clase *ReceivedInformation* como una clase abstracta, la cual es subclasificada con la información concreta, que puede recibir un usuario.

Para nuestros dos ejemplos es suficiente con crear dos subclases de la clase *ReceivedInformation*, la clase *Information*⁵⁴ que representa la información textual que recibe el usuario y la clase *Question*⁵⁵ que representa las preguntas que puede recibir el usuario. De esta manera se especifica que un juego u obra puede brindar al usuario información, y preguntas según la aplicación que se quiera plantear.

En el caso de necesitar brindar al usuario otro tipo de información basta crear una subclase de *ReceivedInformation* con el comportamiento adecuado.

Otro concern digital (*Question Evaluation*) planteado es el que se encarga de evaluar las preguntas en cuestión. Para este concern se definieron tres clases a modo de ejemplificar los aspectos básicos relacionados con la evaluación de cada pregunta. Cada pregunta tiene un evaluador (*QuestionEvaluator*), el cual tiene definida su estrategia de resolución.

Pueden existir más de una estrategia⁵⁶ de resolución, por ejemplo, una pregunta con múltiples respuestas puede ser considerada correcta si se eligen todas las posibles respuestas o puede considerarse correcta si se eligió al menos una de las respuestas correctas.

Las estrategias tienen asociadas acciones⁵⁷ que se realizan cuando se evalúa la pregunta, por ejemplo, sumar un determinado puntaje por pregunta correcta.

En el concern físico sólo es necesario representar los lugares en donde el usuario

⁵⁴ Para simplificar el modelo, la clase *Information* no tiene relación con ninguna otra clase.

⁵⁵ En este caso las preguntas no tienen relación a ninguna información relacionada para simplificar el modelo presentado. El modelo se puede volver más complejo si cada pregunta tiene información asociada. Esta información, puede servir, por ejemplo, para ayudar a resolver la pregunta o profundizar sobre algún tema en particular.

⁵⁶ En el modelo se definió la clase abstracta *Strategy* y no se especificaron posibles subclases a modo de simplificar el modelo. Para las estas estrategias se aplica el patrón de diseño *Strategy* [Gama et al., 1995].

⁵⁷ Sólo se definió la clase abstracta *Action* a modo de simplificar el modelo, esta clase se puede subclasificar para definir las acciones que se podrían ejecutar cuando el alumno responde una pregunta, por ejemplo, si contesto correctamente sumarle un punto. Estas acciones aplican el patrón de diseño *Command* [Gama et al., 1995].

recibe información. Para este ejemplo la propiedad de ubicación esta estereotipada como un valor etiquetado. Usando esta propiedad como un valor etiquetado se pueden probar ambas aplicaciones usando, por ejemplo, códigos de barras ubicados en los lugares donde transcurre el juego o la obra. Cada código de barra se corresponde con una de las etiquetas representadas en el modelo, cuando el usuario lee un código de barra (mediante algún programa instalado en su celular) recibe información del Punto de Interés que se corresponde con esa etiqueta.

Si bien el concern *Mobile Urban Drama* es de interés a nivel navegacional, el concern *Question Evaluation* es sólo de interés a nivel funcional pero no para ser navegado por el usuario. Es decir, no todos los concerns (del Modelo de Contenido) son de interés a nivel de navegación del usuario.

Para lograr tener un Modelo Unificado con los concerns de interés, hay que elegir la opción de transformar el Modelo de Contenido en un Modelo Unificado, considerando sólo algunos concerns de interés. El concern que no es de interés es *Question Evaluation* (el cual tiene la funcionalidad para evaluar las preguntas), este concern sólo es de interés a nivel de Modelo de Contenido, por esta razón el diseñador no lo pasa al Modelo Unificado.

En la Figura 7.16 se muestra la pantalla que aparece cuando se elige la opción de transformar el Modelo de Contenido en un Modelo Unificado sólo algunos concerns. Se muestra la lista de concerns digitales para que el diseñador elija uno de estos como *Core* de la aplicación. En este caso (el diseñador) elige el concern *Mobile Urban Drama*.

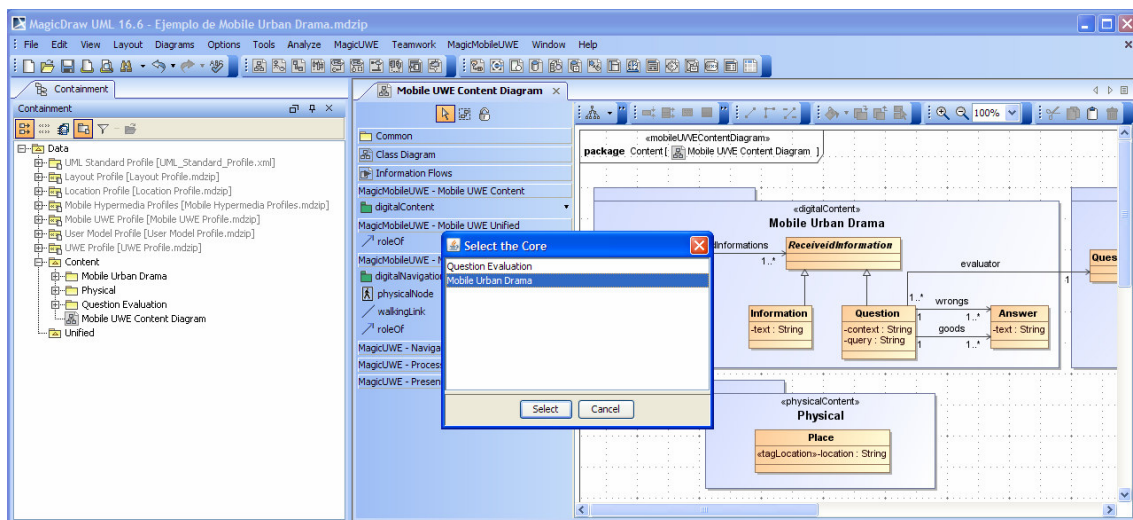


Figura 7.16: Elegir el Core de la aplicación de *Mobile Urban Drama*.

Una vez elegido el concern *Core*, sale en la pantalla la posibilidad de elegir cuáles son los otros concerns de interés, como se muestra en la Figura 7.17. Se debe elegir si los concerns restantes, que en este caso es *Question Evaluation*, pasa o no al Modelo Unificado. En este caso no se selecciona el concern *Question Evaluation*, ya que es sólo un concern de interés a nivel de Modelo de Contenido.

Hay que tener en cuenta que el modelo para representar aplicaciones de *Mobile Urban Drama* tiene una particularidad que el modelo del turista no tenía, y es que hay una asociación entre clases de diferentes concern. La transformación interpreta que si los concerns que contienen ambas clases son parte de los concerns de interés, esta relación es pasada al Modelo Unificado. Pero si alguno de los concerns (que contienen las clases) no están incluidos como concern de interés, esta asociación no es pasada al Modelo Unificado.

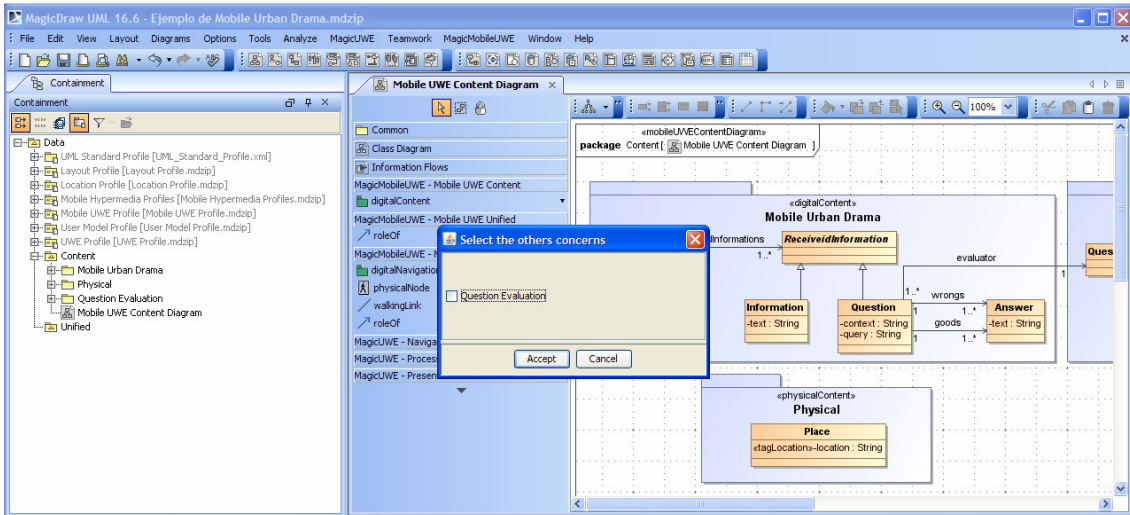


Figura 7.17: Elegir los otros concerns de la aplicación de *Mobile Urban Drama*.

El Modelo Unificado creado se puede visualizar en la Figura 7.18, donde se muestra los concerns *Mobile Urban Drama* y *Physical*. Se debe tener en cuenta que no se pasó la asociación que existe entre las clases *Question* y *QuestionEvaluator*, ya que la clase destino (*QuestionEvaluator*) está en un concern que no fue considerado de interés para el Modelo Unificado.

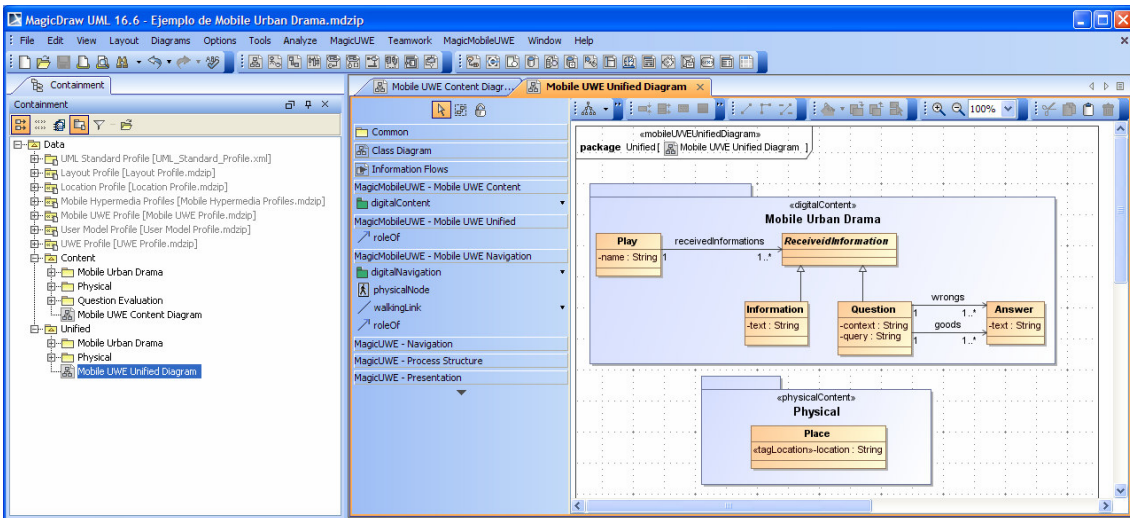


Figura 7.18: Modelo Unificado creado producto de la transformación.

Se puede observar en la Figura 7.18 que no se generó ninguna relación de rol, ya que no hay ninguna clase con el mismo nombre en ambos concerns. La condición que tiene la herramienta para crear la asociación de rol es tener que las clases de diferentes concerns tengan igual nombre.

El diseñador puede crear el rol entre clases, usando la barra de herramienta disponible por la herramienta. Supongamos que el diseñador decide que la clase *Place* se corresponde con la clase *ReceivedInformation*, y agrega la asociación que representa al rol como se muestra en la Figura 7.19. Con esta asociación, se está indicando que cualquier subclase de *ReceivedInformation* tiene una relación de rol con *Place*. Este conocimiento es propio del diseñador y sólo él puede hacer este refinamiento. Es decir, establecer las relaciones de rol cuando dos clases no tienen igual nombre es una tarea del diseñador.

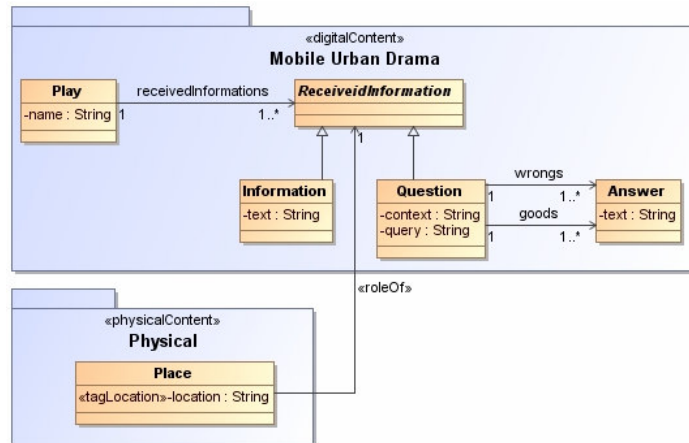


Figura 7.19: Modelo Unificado refinado.

A partir del Modelo Unificado refinado, se hace la transformación al Modelo Navegacional usando la herramienta. Como las aplicaciones del juego educativo y la obra de teatro van a nodos diferentes, se realizó dos veces esta transformación sobre el Modelo Unificado (Figura 7.19) para obtener dos Modelos Navegacionales que se van a corresponder con las dos aplicaciones que se quieren modelar.

Los dos Modelos Navegacionales generados se pueden visualizar en la Figura 7.20, cada uno almacenado en un paquete diferente.

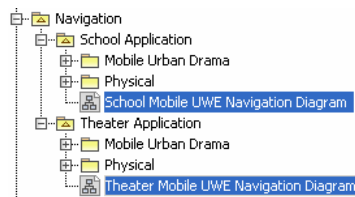


Figura 7.20: Los dos Modelos Navegacionales para el juego educativo y la obra de teatro.

Cuando se realiza la transformación del Modelo Unificado al Modelo Navegacional se genera un modelo como el que se puede observar en la Figura 7.21. Tanto el Modelo Navegacional del juego educativo como el de la obra de teatro, inicialmente tienen los elementos que se muestran en esta figura. Luego veremos cómo refinarlos para que cada uno se ajuste a las características de la aplicación.

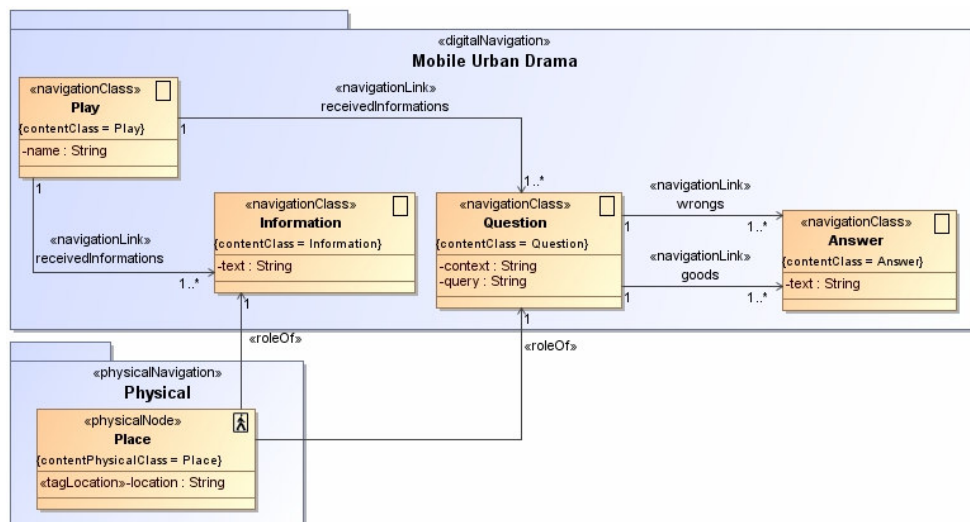


Figura 7.21: Modelo Navegacional creado producto de la transformación.

Al realizarse la transformación (del Modelo Unificado al Modelo Navegacional) la

misma detecta que hay una clase abstracta (*ReceivedInformation*), por como está definida la transformación esta clase no se pasa como nodo, ya que no tiene sentido que se genere un nodo que represente una clase abstracta. Las asociaciones que tiene la clase *ReceivedInformation* son agregadas a cada una de sus subclases, es por esto que surgen los links desde el nodo *Play* a los nodos *Information* y *Question*. También se duplica la relación de rol desde *Place* a *Information* y a *Question*. Es decir, la transformación del Modelo Unificado al Modelo Navegacional tiene en cuenta que, cuando hay una clase abstracta, las subclases heredan sus asociaciones como se muestra en este ejemplo.

Se puede observar además en la Figura 7.21 que los links conservan los nombres que tenían las asociaciones (*receivedInformations*, *wrongs* y *goods*) en el Modelo Unificado y la cardinalidad de las mismas.

Veamos cómo refinar el modelo *School Mobile UWE Navigation Diagram* (ver Figura 7.22). Supongamos que definimos la aplicación para que el juego educativo y el mismo este compuesto solamente de preguntas. Esto quiere decir, que el nodo *Information* no tiene sentido que esté definido en este modelo y por esa razón se elimina.

Otro refinamiento que se debe realizar es sobre el nodo *Answer*, no es cierto que el alumno tenga en la pregunta links a las repuestas, sino, que dentro del nodo de la pregunta tiene todas las posibles respuestas. Por lo tanto, se eliminó el nodo *Answer* y los links de la pregunta a este nodo. Para representar que el nodo *Question* conoce a sus posibles preguntas se definieron dos relaciones (*wrongs* y *goods*) a la clase *Answer* definida en el Modelo de Contenido. Es decir, es información que conoce el nodo de la pregunta pero no es información por la cual navega el alumno, ya que no son links. Cuando se crea la presentación del nodo *Question* se establece cómo visualizar las posibles respuestas relacionadas a cada pregunta.

En el nodo físico *Place* se agregó un link caminable predefinido, el cual indica que desde un determinado lugar se puede tener links caminables predefinidos a otros lugares. Estos links caminables predefinidos sólo son visualizados por los alumnos una vez que estos contesten la pregunta que tienen en un momento dado. Al contestar cada pregunta, el alumno puede ver los posibles links caminable predefinidos desde la pregunta que está respondiendo. Este funcionamiento para los links caminables difiere del presentado para la aplicación turística, que muestra los links caminables predefinidos sin condicionamientos.

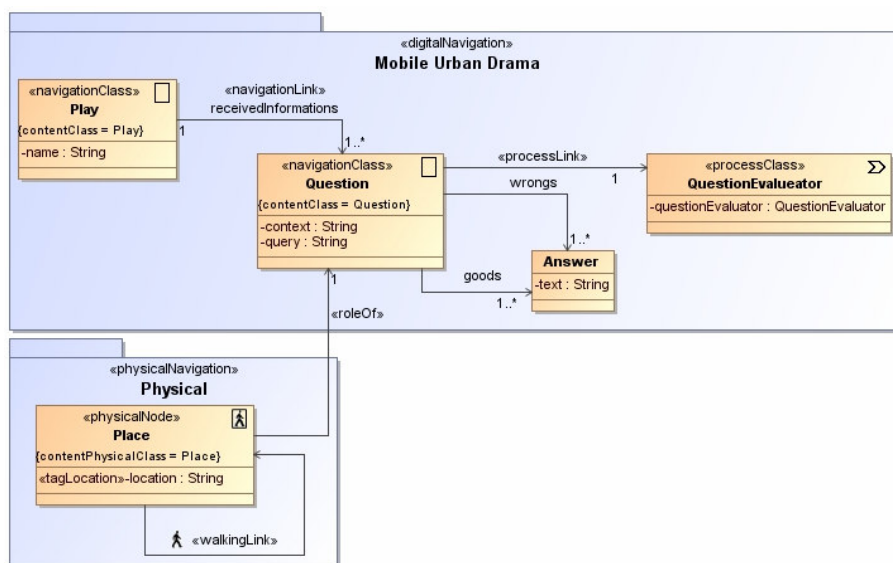


Figura 7.22: Modelo Navegacional del juego educativo.

Además, se puede observar en la Figura 7.22 que el nodo *Question* tiene una

asociación estereotipada como `<<processLink>>`⁵⁸ a una clase llamada *QuestionEvaluator* la cual esta estereotipada como `<<processClass>>`⁵⁸. El estereotipo `<<processClass>>` indica que la clase es un proceso que es invocado por un nodo, y como resultado de ejecutar este proceso se puede ejecutar otro proceso o devolver un nodo. En cuanto al estereotipo `<<processLink>>`, permite dar la semántica que la asociación tiene como destino un proceso.

QuestionEvaluator representa un proceso que se ejecuta para dar una respuesta al usuario o mostrarle como seguir en el juego. Esta clase conoce a un evaluador (*QuestionEvaluator*) definido en el Modelo de Contenido, el cual contiene la estrategia de corrección que a su vez sabe que acciones tomar. Este proceso es invocado cuando se evalúa la respuesta elegida por el alumno.

Para el Modelo Navegacional refinado de la Figura 7.22, se decidió no agregar la operación de links caminables derivados. Al no haber links digitales desde la pregunta no se van a poder derivar links caminables. Por esta razón, en *Place* no tiene sentido definir la operación `derivedWalkingLinks`.

Veamos ahora cómo refinar el Modelo Navegacional *Theater Mobile UWE Navigation Diagram* (Figura 7.23). Supongamos que se quiere tener una aplicación que sólo muestre información en los distintos lugares donde transcurre la obra. Es decir, no es necesario tener los nodos *Question* y *Answer* ni los links a estas clases, por lo tanto se sacan del Modelo Navegacional. De esta manera, la obra sólo va a contener información que recibe el usuario.

Al nodo *Place* no se le agregaron links caminables predefinidos ya que esta aplicación (de la obra de teatro) está focalizada en brindar información al usuario mientras ocurre la obra, pero sin indicarle los posibles lugares donde se debe dirigir para continuarla. Los actores dan el dinamismo necesario para que el usuario se mueva de un lugar a otro para que la historia transcurra.

Tampoco se define la operación para calcular links caminables derivados (`derivedWalkingLinks`), ya que al no tener links digitales en la información que se le brinda al usuario, no tiene sentido definir esta operación ya que no se va a poder derivar ningún link caminable.

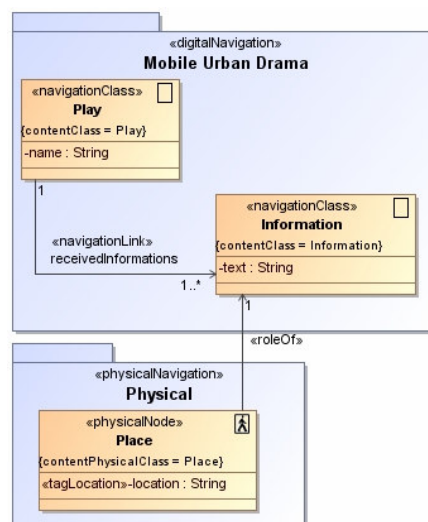


Figura 7.23: Modelo Navegacional de la obra de teatro.

Veamos ahora cómo crear un Modelo de Instancias Navegacional para cada una de las aplicaciones. A partir de los dos Modelos Navegacionales presentados en las

⁵⁸ Estereotipo provisto por UWE.

Figuras 7.22 y 7.23, se crearon dos Modelos de Instancias Navegacionales como se puede observar en la Figura 7.24. Cada uno de estos modelos creados tiene su correspondiente Modelo Navegacional de base.

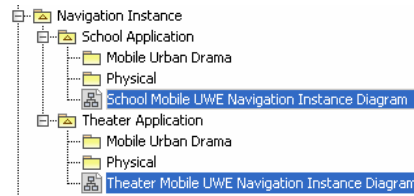


Figura 7.24: Los dos Modelos de Instancias Navegacional creados.

Cada uno de los Modelos de Instancias Navegacionales creados tiene sus propios nodos, acordes a las aplicaciones que se quiere representar (el juego educativo y la obra de teatro). En la Figura 7.25 muestra el modelo *School Mobile Navigation Instance Diagram*.

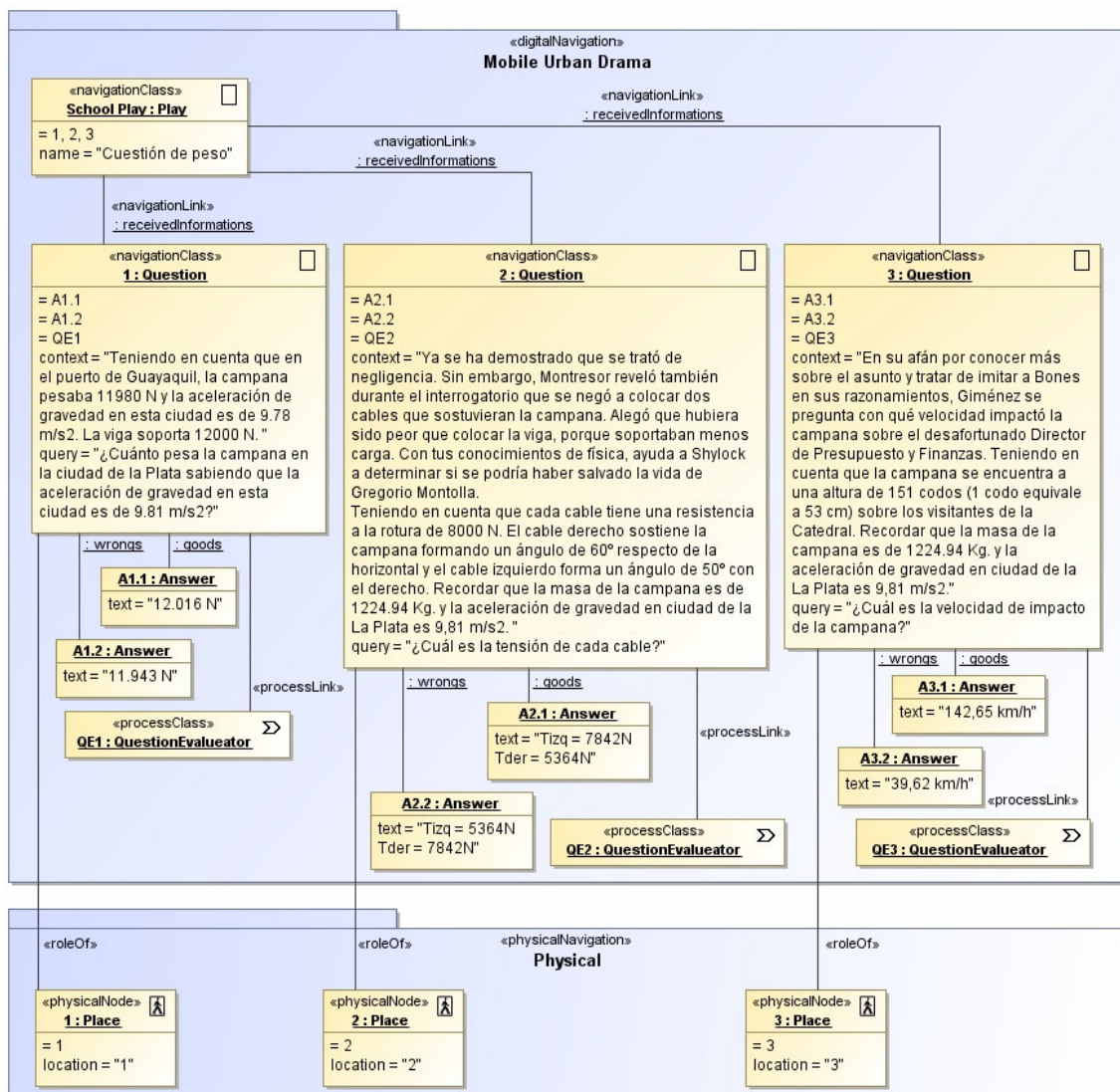


Figura 7.25: Modelo de Instancias Navegacional del juego educativo.

Se puede apreciar (en la Figura 7.2) la instanciación particular de un juego educativo,

este juego⁵⁹, consta de tres preguntas de física donde cada una tiene especificada una respuesta correcta y otra incorrecta. A cada una de estas preguntas se le definió su evaluador (por simplicidad no se agregó la instanciación de la estrategia de corrección ni las acciones que deben tomar las mismas).

Cada pregunta tiene asociado el lugar donde ocurre la misma, y cada lugar tiene el siguiente lugar donde se debe dirigir el alumno. Para este caso particular, se tiene un único posible lugar a donde continuar el juego. Esto es indicado en el modelo mediante los links caminables predefinidos. Estos links caminables (predefinidos) son visualizados por el alumno al responder la pregunta. Es decir, el alumno no sabe a donde se debe dirigir hasta que no responde la pregunta actual.

Se puede apreciar que cada uno de los lugares tiene definido un número que lo identifica, este número está ligado, por ejemplo, a un determinado código de barra.

El nombre del juego es usado para ser listado al alumno cuando se le ofrece más de un juego (al mismo tiempo) para jugar. Cuando el alumno está en un determinado lugar, recibe la pregunta asociada (como respuesta por ejemplo de leer un código de barra), cuando la contesta se invoca al evaluador el cual, una vez realizada la evaluación, le muestra al alumno el siguiente lugar donde se debe dirigir. Este lugar es calculado acorde a la definición de los links caminables predefinidos.

Se puede crear tantos juegos como se sean necesarios, sólo basta definir un Modelo de Instancias Navegacional para cada uno de los juegos que se quiere representar.

En la Figura 7.26 se muestra el modelo *Theater Mobile Navigation Instance Diagram*. Se pueden ver las instancias relacionadas a una obra de teatro, esta obra⁶⁰ define distinta información que recibe el jugador cuando está en cada uno de los lugares.

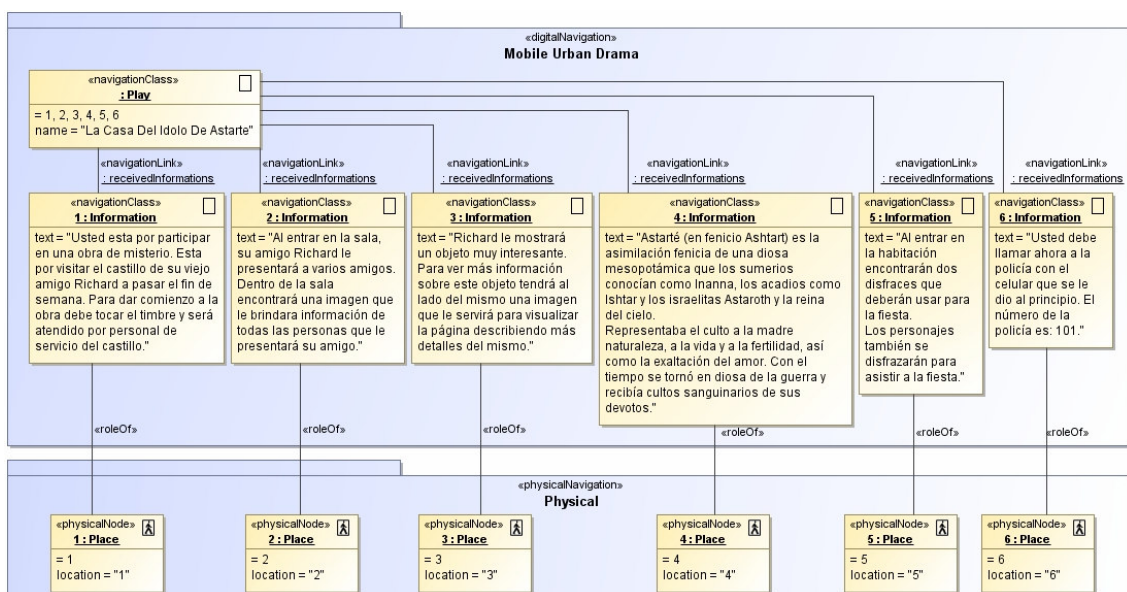


Figura 7.26: Modelo de Instancias Navegacional de la obra de teatro.

⁵⁹ Este ejemplo tiene una historia policial de base que los alumnos deben leer previamente antes de jugar. La historia deja planteada una muerte misteriosa, donde a uno de los personajes se le cae encima una campana de una iglesia y muere aplastado. El alumno mediante preguntas de física debe determinar si hubo negligencia, o no, cuando se coloco dicha campana. Las preguntas están pensadas para alumnos de cuarto o quinto año de una escuela técnica, los cuales cuentan con los conocimientos necesarios para realizar los cálculos planteados en estas preguntas.

⁶⁰ La obra está inspirada en el cuento de Agatha Christie llamado "La casa del ídolo de Astarté". A partir del cuento se creó la información que recibe el usuario en cada lugar. Además, los actores usan este cuento para saber cuál es la trama de la obra e improvisar sin perder el foco de la misma.

Tanto en la Figura 7.25 como 7.26, cada uno de los lugares (*Place*) tiene definido un número que lo identifica, este número es ligado, por ejemplo, a un determinado código de barra, que el jugador lee con su dispositivo móvil y como resultado recibe la información asociada a ese lugar.

Al tener definido dos Modelos Navegacionales diferentes, es necesario crear dos Modelos de Presentación como se muestra en la Figura 7.27. Cada uno tiene su correspondiente Modelo Navegacional de base.



Figura 7.27: Modelos de Presentación creados.

Cuando se realiza la creación (usando la herramienta *MagicMobileUWE*) de cada Modelo de Presentación se crea la presentación de cada uno de los nodos.

La Figura 7.28 muestra el Modelo de Presentación que se corresponde con el juego educativo.

Se puede ver que tanto la clase *QuestionEvaluation* como *Answer* no tienen su correspondiente clase de presentación. Esto se debe a que *QuestionEvaluation* es un proceso que se encarga de hacer la evaluación de la pregunta y, por lo tanto, no requiere definir una presentación ya que no va a ser visualizada por el alumno.

Answer representa una clase común que no es visualizada por el usuario como un nodo aparte, sino que su información está asociada al nodo *Question*. La clase de presentación *Question* debe especificar cómo se muestran las posibles respuestas asociadas. También esta clase (*Question*), debe tener definida la forma de poder indicar que se evaluó la pregunta, por ejemplo, mediante un botón y cuando el alumno lo selecciona se ejecuta la clase *QuestionEvaluation*.

Por lo tanto, hay que tener en cuenta que, cuando se realiza la creación de un Modelo de Presentación, sólo pasan a tener asociadas clases de presentación aquellas clases que están estereotipadas como `<<navigationClass>>` o `<<physicalNode>>` ya que está es la manera de determinar que realmente son nodos. Esta consideración es realizada cuando se crea un Modelo de Presentación en base a un Modelo Navegacional.

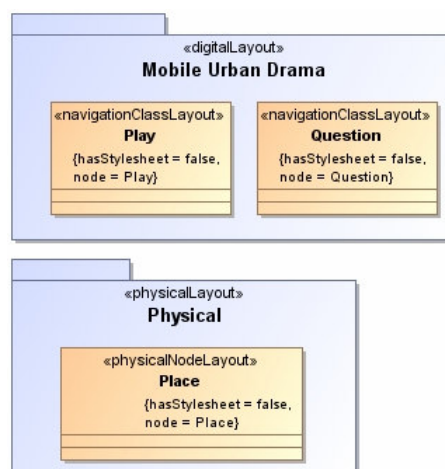


Figura 7.28: Clases de presentación del juego educativo.

La Figura 7.29 muestra el Modelo de Presentación creado para la aplicación de la obra de teatro. Como el Modelo Navegacional contaba solamente con tres nodos, el Modelo

de Presentación creado tiene sus tres clases de presentación correspondientes.

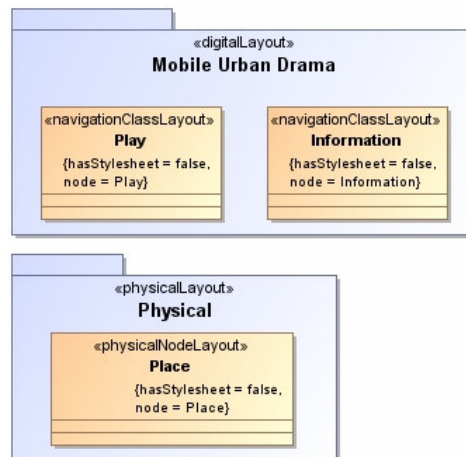


Figura 7.29: Clases de presentación de la obra de teatro.

Tanto para las Figuras 7.28 y 7.29 el estilo de presentación asociado a cada clase se define usando la herramienta, de la misma manera que el ejemplo del turista presentado en la Sección 6.5.

Al ser dos aplicaciones diferentes se deben crear dos Modelos del Usuario. La Figura 7.30 muestra el Modelo del Usuario para el juego educativo. Se puede apreciar que dicho modelo tiene definido su correspondiente Modelos de Instancias Navegacional y de Presentación. El Modelo del Usuario fue creado con las estrategias de navegación definidas por default, pero el diseñador decidió cambiarlas para que el modelo considere guardar estados. Es decir, una característica distintiva (del modelo presentado en la Figura 7.30) respecto de los anteriores Modelos del Usuario, es que la aplicación que se definió maneja estados. Esta decisión, se debe a que este tipo de aplicación puede considerar los estados, tanto a nivel del usuario, como a nivel de aplicación.

Manejar estados permite, por ejemplo, que una aplicación pueda estar a la espera de una cantidad determinada de alumnos logueados, para recién ahí dar comienzo al juego.

Guardar el estado de los alumnos puede ayudar a monitorearlos y, de esta manera, brindarles distintos tipos de asistencia. Por ejemplo, si la aplicación tiene almacenado el estado “caminando” para un alumno particular y, pasa una cantidad determinada de tiempo, la aplicación le brinda más ayuda para que pueda llegar a su destino. Lo mismo puede pasar si la aplicación tiene guardado que el alumno está contestando una pregunta y pasa más tiempo del estimado (para contestar esa pregunta), darle pistas que lo ayuden a responder la pregunta.

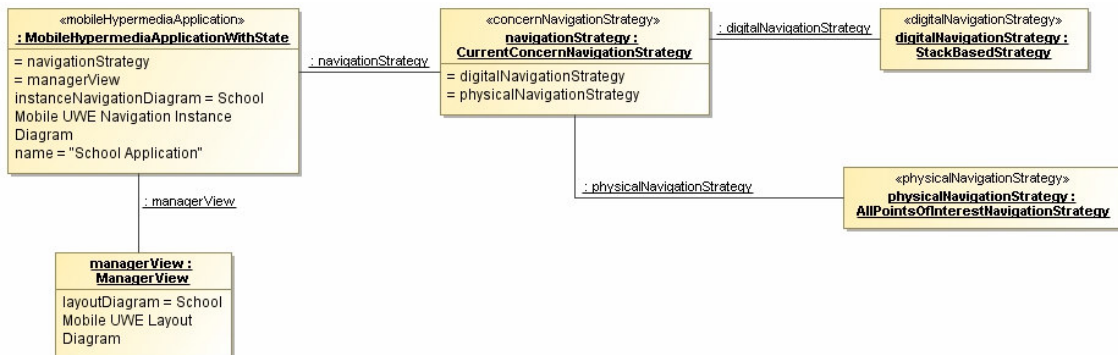


Figura 7.30: Modelo del Usuario para el juego educativo.

La Figura 7.31 define el Modelo del Usuario para la aplicación de la obra de teatro con sus correspondientes Modelos de Instancias Navegacional y de Presentación. Las estrategias de navegación son las creadas por default por la herramienta. La aplicación de la obra de teatro no necesita manejar estados, ya que sólo muestra información y la dinámica de la obra la dan los actores.

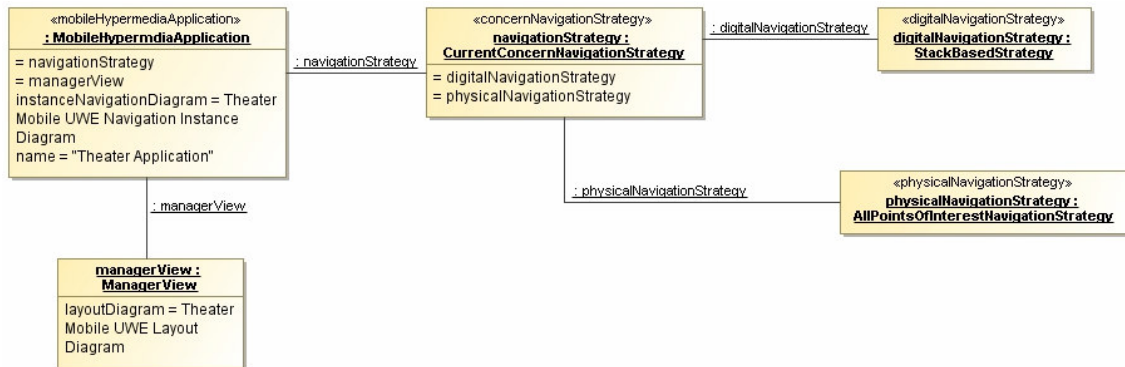


Figura 7.31: Modelo del Usuario para la obra de teatro.

En esta sección se mostró como crear a partir de un mismo Modelo de Contenido, que representa a las aplicaciones de *Mobile Urban Drama* distintos Modelos Navegacionales para dos aplicaciones con dominios diferentes. Se mostró cómo, a nivel de instancias, crear dos posibles aplicaciones con características diferentes. Se pudo apreciar cómo se refinaron los dos Modelos Navegacionales para ajustarse a las características de cada una de las aplicaciones.

7.3. Características particulares de los ejemplos presentados

En las Secciones 7.1 y 7.2, se presentaron tres ejemplos de dominios diferentes que pueden ser modelados con el enfoque. En esta sección se describe más en detalle las características de cada uno de los dominios mostrando como cada uno se puede volver más complejo y como esto puede ser modelado. Además, destacaremos aquellas características propias que tiene cada uno de estos dominios.

Comencemos analizando la aplicación turística, como pudimos ver tanto los ejemplos presentados en la Sección 6 como en la Sección 7.1, la aplicación turística puede estar destinada a varios tipos de usuarios diferentes, y brindar a cada uno información acorde a su perfil. Como mostramos, esto se modela teniendo diferentes Modelos del Usuario donde cada uno tenga asignado su propio Modelo de Instancias Navegacional.

La característica distintiva de este tipo de aplicaciones, es que la mayoría de las clases se modelan en varios concerns. Esto permite que el usuario pueda cambiar de concern desde la opción que recibe en la pantalla, y ver un mismo Punto de Interés desde distintas perspectivas (histórico, arquitectural, etc.).

Este tipo de aplicación no contiene sólo información descriptiva, sino que se puede modelar funcionalidad, por ejemplo, relacionadas a e-commerce (comercio electrónico). Usando la herramienta, esto se resuelve definiendo un concern separado y, al definir el Modelo Navegacional, se modelan clases de procesamiento (estereotipadas como <<processClass>>) para resolver los aspectos que necesiten verificación de datos. De esta manera, el usuario puede tener, por ejemplo, desde el nodo de un museo la posibilidad de comprar la entrada al mismo, o recibir información que el museo tiene una promoción asociada a su entrada.

También se puede pensar que los usuarios tuvieran la posibilidad de dejar comentarios respecto de los lugares que visita, para que luego otros usuarios los

podieran mirar. Esto forma parte de otro concern dentro de la aplicación. En el Modelo Navegacional se tienen aquellos nodos que representan el historial de comentarios asociados a un Punto de Interés. Además, desde el nodo se debe permitir agregar comentarios y que estos sean agregados al historial.

Se puede pensar dentro de la aplicación turística en otro concern relacionado a tareas que debe cumplir el usuario como por ejemplo, realizar la compra de regalos mientras visita una determinada ciudad.

En este tipo de aplicación muchas veces el usuario va a necesitar ayuda para completar los links caminables que eligió realizar, la aplicación debe ir monitoreando si el usuario sigue en el camino establecido. Si se desvió del camino, identificar cuál fue la causa para poder asistirlo. Por ejemplo, un usuario puede desviarse del camino, porque algo le llamo la atención o porque realmente se desorientó. Para poder asistirlo, es necesario saber los estados del usuario, en particular el estado actual y en base a este brindarle asistencia.

A partir de las características mencionadas se puede observar que las aplicaciones turísticas, pueden volverse más complejas necesitando considerar cuestiones de distinta naturaleza.

En cuanto a la aplicación del juego educativo, se mostró en la Sección 7.2 un juego simple que consistía en preguntas que tenían dos opciones posibles, y que a cada pregunta tenía un sólo posible lugar a donde debe dirigirse, una vez respondida la misma. Pero este tipo de juego puede ser más complejo, por ejemplo que las preguntas tengan múltiples opciones de respuesta y asociado a esto diferentes formas de evaluar al alumno. Las preguntas pueden tener links a información que ayuda a entender mejor la pregunta o a profundizar algún conocimiento sobre el tema.

Tener distintos lugares a donde se puede dirigir el alumno cuando contesta la pregunta, genera que el alumno deba decidir cuál de todos los caminos posibles elige. Se debe evaluar al alumno, acorde al camino que va eligiendo recorrer. Puede pasar, que dependiendo de cómo contesto el alumno reciba distintos caminos a donde se puede dirigir.

El diseñador puede plantear un juego más complejo donde, por ejemplo, los alumnos deban interactuar para poder contestar sus preguntas. En este caso, cada alumno conoce parte de la información, pero sólo mediante la interacción con otro alumno puede tener la información completa y de esta manera se resuelven las preguntas. También el diseñador puede crear juegos acordes a diferentes tipos de alumnos, es decir, con contenidos acorde al año que está cursando cada uno, esto lo puede realizar con el enfoque teniendo diferentes Modelos del Usuario, donde para cada uno tiene su correspondiente Modelo de Instancias Navegacional.

También se puede especificar que los alumnos se loguen al sistema, y de esta manera tener asociado todos sus juegos con sus evaluaciones. Para poder tener esto, en el Modelo Navegacional se deben agregar los nodos referentes a la parte del logueo y las clases de procesamientos que hacen la verificación del alumno en el sistema. Para poder tener registrado todos los juegos de cada alumno, se debe crear un Modelo del Usuario que registre la información referente a los juegos y cómo fue evaluado el alumno. Tener la información referente a los juegos de cada alumno, le permite a los docentes, ver cómo fue evolucionando cada uno. Para esto hay que crear también un Modelo del Usuario para el docente, el cual permita ver información de todos juegos de los alumnos y como estos contestaron cada pregunta.

También se puede pensar en que la aplicación brinde la posibilidad de pausar un juego para continuarlo en otro momento (para lo cual se necesita registrar en que estado estaba el alumno al momento de pausar el juego).

En este tipo de aplicaciones, es importante la asistencia que necesita el alumno para completar el juego y no pierda el interés en el mismo. El alumno puede necesitar ayuda para moverse de un lugar a otro para obtener cada una de las preguntas, como así también, orientación cuando no comprende la pregunta en cuestión.

Además, se puede pensar en un tutor que acompañe al alumno y, que cuando éste lee cada pregunta, el tutor reciba información relacionada con los consejos que le puede dar al alumno. El tutor evalúa cuando el alumno necesita estos consejos, o si necesita una explicación sobre algún tema relacionado con la pregunta. La figura del tutor permite que éste le brinde al alumno la información acorde a las necesidades que tiene cada uno. También puede hacer anotaciones de cómo ve el desarrollo del alumno dentro del juego, para que luego el docente visualice estos comentarios y, de esta manera, incorporar o rever temas para que los alumnos refuercen sus conocimientos. En este caso el tutor tiene una dinámica distinta a la del alumno respecto de los links caminables, ya que los mismos son activados cuando el alumno recibe un link caminable. El tutor no elige caminar un link, sino que debe hacer el mismo camino que el alumno. Por lo tanto, el tutor recibe el camino automáticamente cuando el alumno lo recibe. Estas cuestiones son propias de la dinámica de la aplicación, y se pueden considerar, por ejemplo, teniendo un Modelo del Usuario particular para el tutor.

Se puede ver en el Modelo de Contenido presentado en la Sección 7.2, que la división por concerns en este tipo de aplicación está relacionada a separar funcionalidad y no para tener diferentes vistas de un mismo Punto de Interés. La utilización de concerns en este caso, es para tener bien desacoplada cada una de las funcionalidades.

Por último veamos las características de la obra de teatro. En la Sección 7.2 se presento un ejemplo muy simple donde el usuario solamente recibe información en cada una de las escenas. Este tipo de aplicaciones se puede volver más compleja, si además el usuario tiene links caminables que seguir, y más aún, si desde un lugar se le plantea que puede dirigirse a varios lugares. Dependiendo del lugar a donde eligió dirigirse, la historia puede cambiar de rumbo.

También se puede pensar que el usuario tenga que ir respondiendo preguntas mientras va jugando la obra. Puede tener links digitales que le brinden más información para poder interactuar mejor dentro de la obra. La aplicación puede permitirle hacer anotaciones que luego lo por ejemplo, a resolver un misterio.

Dentro de la obra puede haber más de un jugador, y cada uno tener un rol diferente dentro de la obra. Para que cada uno tenga información acorde a su perfil, cada uno debe tener su propio Modelo del Usuario. Puede pasar que dependiendo del rol, el usuario tiene o no acceso a determinada información. Al tener varios jugadores dentro de la obra es más compleja la parte de improvisación, ya que muchas cuestiones pasan a ser decisiones de cada uno de los jugadores.

Se puede pensar, además, que los actores también cuenten con dispositivos móviles y saben donde esta cada usuario en un momento determinado y que fue realizando. De esta manera, la aplicación ayuda a los actores a poder improvisar mejor. Para que los actores reciban esta información, deben contar con su propio modelo que les permita acceder a toda la información, tanto de la obra, como la información de los usuarios. Los actores pueden tener links caminables disponibles, que los jugadores no conocen, por ejemplo, pasadizos secretos sólo habilitados para que ellos (los actores) puedan seguir la dinámica del juego.

La parte de asistencia para que cada jugador no se desvíe del juego corre más por parte de los actores, los cuales tienen que improvisar para poder lograr que la historia vuelva a su curso. Para esto es de mucha ayuda que cada uno de los actores cuente con un dispositivo móvil para detectar cuando el jugador se perdió, o no siguió la historia como estaba pautada.

Con la descripción de los tres dominios se puede apreciar que cada uno de los ejemplos presentados se puede volver más complejos. Veamos algunas cuestiones que son distintivas en cada aplicación.

Si bien los tres dominios mostrados tienen representado el concern físico, y dentro del mismo los links caminables, cada uno puede tener un mecanismo diferente para

accionarse. En el ejemplo del turista el usuario ve los links caminables predefinidos cuando está frente al objeto físico. Pero en el juego educativo la habilitación de los links caminables predefinidos, depende si el alumno contesto la pregunta. Es decir, cuando el alumno llega a un lugar no ve links caminables. Más aún, pueden pasar que dependiendo de cómo conteste la pregunta (bien o mal) recibe distintos links caminables predefinidos. Otro mecanismo diferente, es si se plantea tener un tutor (en el juego educativo), el mismo recibe el link caminable elegido por el alumno. Es decir, nunca tiene la posibilidad de elegir su camino sino que sigue al alumno.

Se puede ver que la forma de proveer los links caminables, depende de cada aplicación y del funcionamiento que tenga la misma. Sin embargo, a nivel de modelo, se especifican y luego en la etapa de implementación se decide que mostrar y cuando según el dominio que se este representando.

Otra característica que distingue la aplicación turística de las otras dos, es que las clases se representan en cada concerns. En el Modelo de Contenido de la aplicación turística, la mayoría de las clases se modelan en varios concerns. Esto permite que el usuario pueda cambiar de concerns desde la opción que recibe en la pantalla, y ver un mismo Punto de Interés visualizado desde distintas perspectivas. En cambio, en las otras dos aplicaciones la separación de concerns, se realiza para separar funcionalidad y no para tener diferentes vistas de un mismo Punto de Interés. La utilización de concerns en este caso, es para tener bien desacoplada cada una de las funcionalidades. Y van a surgir relaciones entre clases de diferentes concerns.

De esta manera, el enfoque permite el modelado de separación de concern acorde a las necesidades de cada uno de los dominios.

Cada una de estas tres aplicaciones, tiene diferentes tipos de usuarios y cada uno recibe información diferente. En la aplicación del turista, puede haber diferentes tipos de usuarios como mencionamos el turista o el historiador, ambos reciben información de los Puntos de Interés cada uno acorde a su perfil.

En el juego de la escuela, el alumno recibe la pregunta pero el tutor recibe consejos que puede dar al alumno. En cuanto a los links caminables, el tutor nunca los tiene disponibles para elegirlos sino que directamente le sale el camino que eligió recorrer el alumno. Además, el docente recibe información y links diferentes a los alumnos y tutores.

Por otro lado, en la obra de teatro la información que reciben los jugadores varían respecto de los actores ya que estos tienen acceso al historial de cada uno de los jugadores. También pueden variar los links digitales y caminables predefinidos que reciben cada uno.

Se puede ver que cada dominio tiene perfiles de usuarios diferentes con información y links disponibles acordes a cada aplicación. Donde para cada dominio las aplicaciones funcionan de manera particular.

7.4. Resumen

En este capítulo se presentaron aplicaciones con dominios diferentes, destacando cómo las características propias de cada una pueden ser modeladas usando el enfoque.

En la Sección 7.1 se describió una aplicación turística para la ciudad de *La Plata*, la cual se generó a partir del Modelo Unificado presentado en el Capítulo 5.

En la Figura 7.32 se puede apreciar el proyecto creado para la aplicación turística, que contiene los modelos relacionados a las aplicaciones particulares de las ciudades de Roma y de *La Plata*.

Se puede observar que ambas aplicaciones tienen en común los Modelos de Contenido y Unificados. En la figura se muestran que la transformación del Modelo Unificado al Navegacional se aplicó para generar el Modelo Navegacional de la ciudad de *Roma* y, además, para generar el Modelo Navegacional de la ciudad de *La Plata*. Se puede observar que desde el Modelo Navegacional de la ciudad de *La Plata*, se hicieron dos instanciaciones, una para el turista y otra para el historiador.

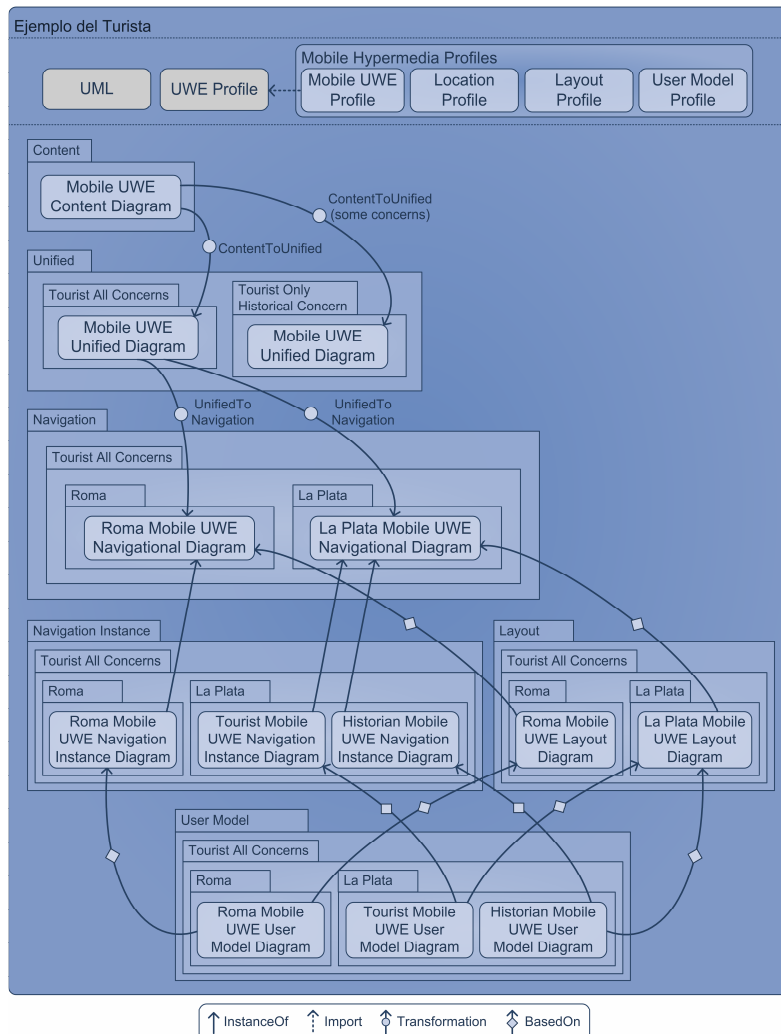


Figura 7.32: Proyecto con los modelos de la aplicación turística.

En la Sección 7.2 se describieron dos aplicaciones de *Mobile Urban Drama*, como son los juegos educativos y las obras de teatro. En la Figura 7.33 se puede apreciar el proyecto creado para estas aplicaciones. Se puede observar que desde el mismo Modelo Unificado se logran dos Modelos Navegacionales, los cuales tienen dominios con características totalmente diferentes (como se describió en la Sección 7.3). Si bien en este ejemplo, cada Modelo Navegacional se instanció una sola vez se pueden crear más Modelos de Instancias Navegacional para representar los datos concretos de otras aplicaciones.

Cabe destacar que dentro de los proyectos presentados en las Figuras 7.32 y 7.33 se usa nuestro perfil *Mobile Hypermedia* y el perfil de *UWE*, además de los elementos de *UML*. De esta manera, se presentó cómo usar la herramienta *MagicMobileUWE* para crear aplicaciones que tienen diferentes dominios.

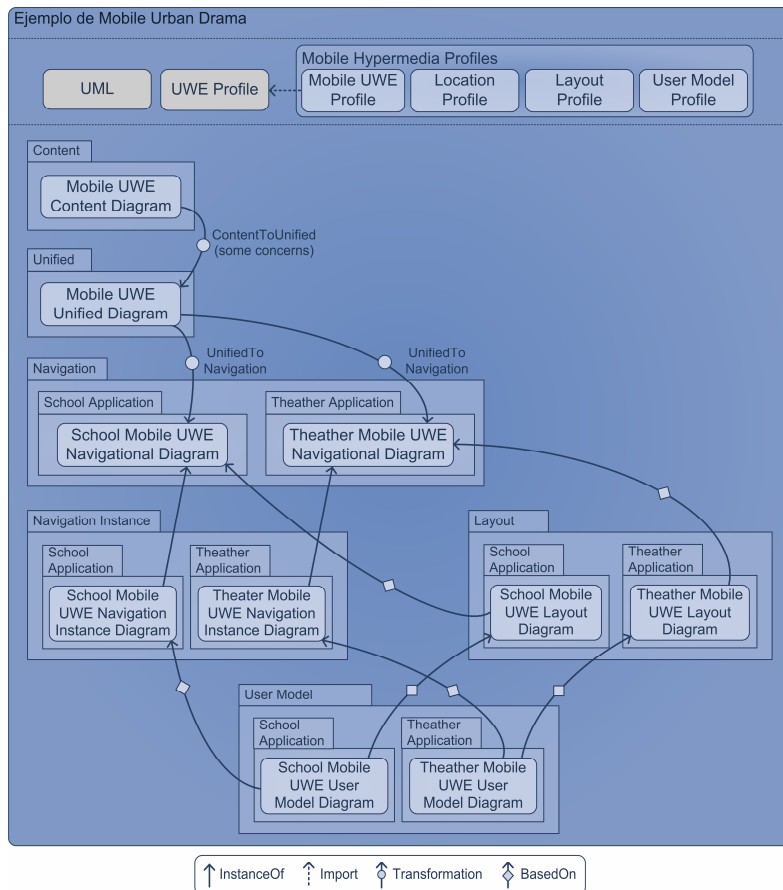


Figura 7.33: Proyecto con los modelos relacionados al juego educativo y la obra de teatro.

En este capítulo vimos algunos aspectos nuevos que aparecieron en los ejemplos de *Mobile Urban Drama* presentados en la Sección 7.2. A continuación se destacan como estos aspectos son considerados por la herramienta.

- Se pudo apreciar (en la Sección 7.2) como funciona la transformación del Modelo de Contenido al Modelo Unificado considerando sólo los concerns de interés, en particular, se mostró que pasa cuando existe una asociación (en el Modelo de Contenido) entre clases contenidas en dos concerns diferentes. En esta situación, esta transformación crea la asociación correspondiente en el Modelo Unificado solamente cuando los concerns involucrados fueron seleccionados como de interés.
- En la Sección 7.2 se pudo apreciar cómo funciona la transformación del Modelo Unificado al Modelo Navegacional cuando existe una clase abstracta en el Modelo Unificado. Como no tiene sentido tener un nodo para representar una clase abstractas, esta transformación no genera el nodo correspondiente a esta clase (abstracta) y las relaciones que tenía la clase abstracta son tomadas por las subclasses y pasadas al Modelo Navegacional como links de los nodos correspondientes a las subclasses.
- Se pudo observar en el Modelo Navegacional de Figura 7.22 (Sección 7.2) la necesidad de tener clases que representen procesamiento. En particular, se modeló una clase para procesar las respuestas de los alumnos.
- Se mostró cómo se pueden tener clases en el Modelo Navegacional que no representan nodos en el Modelo Navegacional, como eran las respuestas. Estas clases son necesarias para representar información asociada a los nodos y en la presentación del nodo se establece cómo mostrar esta información.

Los ejemplos presentados sobre *Mobile Urban Drama* permitieron mostrar otros aspectos del funcionamiento del enfoque que no se habían podido apreciar con el ejemplo de las aplicaciones turísticas (tanto para las ciudades de *Roma* y de *La Plata*).

8. DERIVACIÓN DE APLICACIONES MÓVILES

En este capítulo se presentan dos tipos de derivaciones de aplicaciones que se pueden generar con la herramienta *MagicMobileUWE*. Una de las derivaciones crea una aplicación *J2ME (Java Micro Edition)* [Flanagan, 2002] a partir de la información definida en un Modelo de Instancias Navegacional. Esta aplicación se instala en el celular y luego se ejecuta.

La otra derivación crea una aplicación Web Móvil que cuenta también con la información definida en un Modelo de Instancias Navegacional. La aplicación Web Móvil derivada se debe instalar en un servidor Web y queda funcionando para ser accedida por el usuario mediante un Browser Web desde su celular.

Para ambas derivaciones se definió un modelo de aplicación básico para representar tanto el Modelo Navegacional como el Modelo del Usuario, a nivel de código *Java*. Además, este modelo básico tiene representada una capa para el manejo de mapas.

En este capítulo se muestra cómo el diseñador puede usar la herramienta *MobileMagicUWE* para generar aplicaciones concretas a partir de los modelos del enfoque.

8.1. Modelo básico de la aplicación de Hipermedia Móvil

El modelo básico de la aplicación para ambas derivaciones cuenta con la representación de todos los conceptos del Modelo Navegacional y del Modelo del Usuario del enfoque. En la Figura 8.1 se puede apreciar las clases que representan los conceptos del Modelo Navegacional, este modelo se instancia (a nivel de código *Java*) luego con la información de un Modelo de Instancias Navegacional creado con *MagicMobileUWE*.

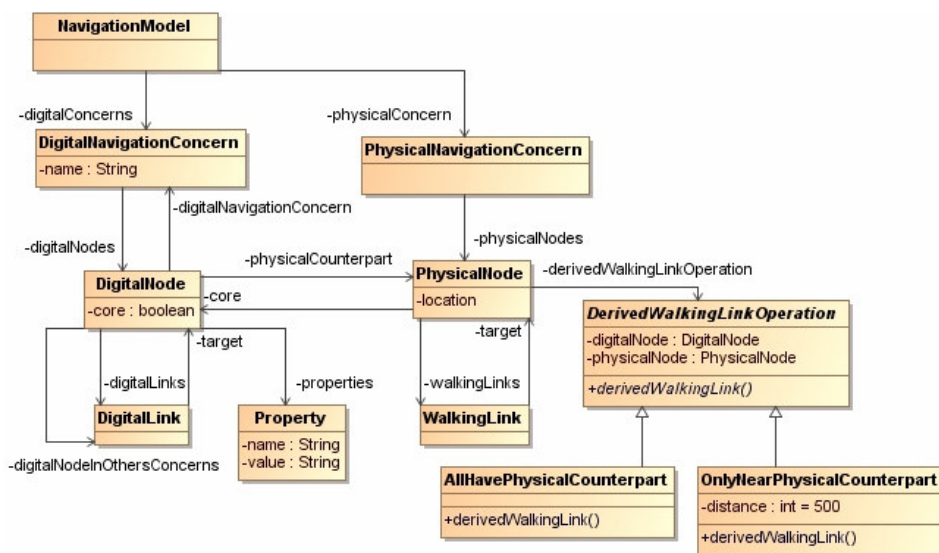


Figura 8.1: Modelo Navegacional implementado en las aplicaciones que deriva la herramienta.

Se puede observar que el Modelo Navegacional (Figura 8.1) tiene una clase *NavigationModel* que lo representa, este modelo está compuesto de concerns digitales (*DigitalNavigationConcern*) y un concern físico (*PhysicalNavigationConcern*). Cada uno de los concerns está compuesto por nodos, ya sean digitales (*DigitalNode*) o físicos (*PhysicalNode*) respectivamente. A su vez cada nodo cuenta con sus correspondientes links digitales (*DigitalLink*) o caminables (*WalkingLink*).

Cada nodo digital tiene definido sus propias propiedades, las cuales se especifican

instanciando la clase *Property* (que tiene definido un nombre y un valor). Esto permite tener representado nodos que tengan propiedades con diferentes nombres. El nodo físico tiene definido la ubicación y la operación de links caminables derivados. Por ahora están definidas dos operaciones concretas, las cuales se corresponden con los dos estereotipos provistos por la herramienta.

Para tener representado el concepto de rol, se optó por definir, a nivel del nodo digital si este es *Core* o no. Además, se definió la relación *digitalNodeInOtherConcerns* en los nodos digitales para relacionar a todos aquellos nodos que tienen un *Core* común. A su vez, cada nodo digital tiene su contraparte física, como así también cada nodo físico conoce su *Core* (un nodo digital). Estas relaciones no solamente permiten tener representado el concepto de rol, sino que además facilitan cambiar de concern digital y pasar a ver la contraparte digital o la contraparte física de un Punto de Interés según corresponda.

La Figura 8.2 muestra el Modelo del Usuario (el cual es parte del modelo básico de la aplicación). Se puede observar que se representa con una clase al usuario, el cual cuenta con un nombre, una ubicación y la lista de aplicaciones asociadas (una de las cuales es la aplicación actual que está usando el usuario).

Cada aplicación del usuario (*UserApplicationInformation*) conoce una aplicación de HM concreta (*MobileHypermediaApplication*) junto a los historiales de navegación digital (*DigitalNavigationHistory*) y de navegación en el mundo real (*PhysicalNavigationHistory*) que se realizaron para esa aplicación. La clase *MobileHypermediaApplication* conoce un Modelo Navegacional (definido en la Figura 8.1), el cual contiene toda la información que puede visualizar el usuario cuando utiliza la aplicación. Los historiales registran tanto los nodos digitales como físicos y además registran cuáles son los nodos, digital y físico, actuales que está visualizando el usuario.

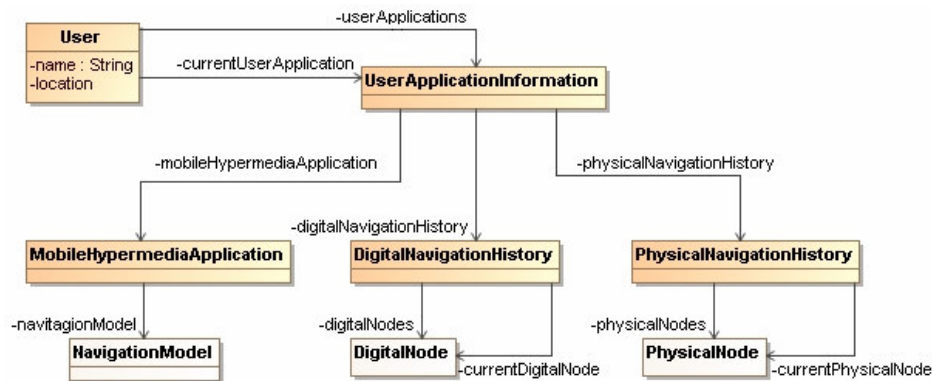


Figura 8.2: Modelo del Usuario implementado en las aplicaciones que deriva la herramienta.

Además del Modelo Navegacional y el Modelo del Usuario, el modelo básico de la aplicación tiene la capa de manejo de mapas. Por ahora, esta capa está definida para proveer sólo mapas de *Google Static Maps*⁶¹. En el caso de querer representar aplicaciones con otro tipo de mapa el diseñador debe modificar las aplicaciones derivadas.

En la Figura 8.3 se pueden observar las clases definidas para la capa de manejo de mapas. La clase *GoogleStaticMap* contiene el ubicación central que debe tener el mapa, tiene el *marker* (*GoogleStaticMapMarker*) que representa la ubicación del

⁶¹ Página de la *Google Static Map API*: <http://code.google.com/intl/es-ES/apis/maps/documentation/staticmaps/>

usuario, los markers que representan los Puntos de Interés y los caminos (*GoogleStaticMapPath*) a mostrar en el mapa (si es que existieran).

Además, se tienen dos clases *GoogleStaticMapsAPI* y *GoogleDirectionsAPI* que representan el principal comportamiento de la API de *Google Static Map* y la API de *Google Directions*⁶² respectivamente.

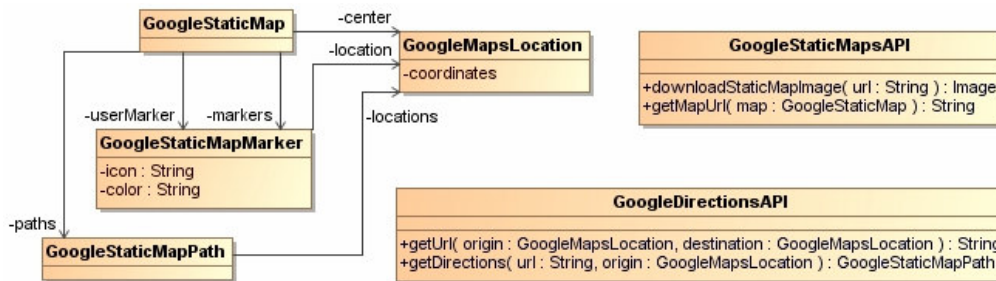


Figura 8.3: Modelo para el manejo de mapas implementado en las aplicaciones que se deriva la herramienta.

Para obtener la imagen que representa la información de un mapa se debe invocar de la clase *GoogleStaticMapsAPI* el método `getMapUrl()`. Este método toma los datos del mapa (*GoogleStaticMap*) que le llega como parámetro y genera un *String* que representa la URL (*Uniform Resource Locator*) con la información que debe mostrar el mapa.

Luego se debe invocar (de la clase *GoogleStaticMapsAPI*) el método `downloadStaticMapImage()` pasándole como parámetro la URL generada. Este método realiza un requerimiento *HTTP* (*Hypertext Transfer Protocol*) con la URL (que recibe como parámetro) y obtiene como resultado la imagen⁶³ que representa al mapa con todos sus datos. Esta imagen es la que visualiza el usuario en la aplicación de HM.

Para la búsqueda de caminos se invoca primero el método `getUrl()` de la clase *GoogleDirectionsAPI*. Este método arma una URL que respeta la especificación de la API para búsqueda de caminos entre dos puntos. Esta URL indica que se quiere obtener un camino entre un punto origen y un punto destino. Luego, esta URL se utiliza como parámetro del método `getDirections()`, el cual envía un requerimiento *HTTP* con la URL y obtiene como resultado un path (*GoogleStaticMapPath*) con la información del camino. Para que este camino sea mostrado al usuario, se debe crear un mapa (*GoogleStaticMap*) y agregar este path. Luego, usando la API de *Google Static Map* se obtiene la imagen del mapa (el cual contiene el camino). Para esto se debe invocar el método `getMapUrl()` enviándole como parámetro el mapa generado (el cual contiene el camino), como resultado se obtiene una URL. A continuación se debe invocar el método `downloadStaticMapImage()` de la clase *GoogleDirectionsAPI*, enviándole como parámetro la URL obtenida y como resultado se obtiene la imagen del mapa. Esta imagen contiene toda la información del mapa, y además tiene dibujado el camino que debe realizar el usuario.

El modelo básico de la aplicación presentado (Figuras 8.1, 8.2 y 8.3) es utilizado por los dos tipos de aplicaciones (*J2ME* y *Web Móvil*) que deriva la herramienta.

⁶² Página de *Google Directions* API:

<http://code.google.com/intl/es-ES/apis/maps/documentation/directions/>

⁶³ Esta imagen se genera por barrido de arriba hacia abajo, en el caso de haber elementos posicionados en el mismo lugar puede que uno se superponga al otro y sólo se visualice en la imagen resultante uno de ellos.

8.2. Aplicación de Hipermedia Móvil – J2ME

J2ME (*Java Micro Edition* o *Java Mobile Edition*, ambas son validas en la bibliografía) [Flanagan, 2002] es una plataforma que permite el desarrollo de aplicaciones para dispositivos pequeños usando *Java*. Esta plataforma posee dos tipos de configuraciones *CLDC* (*Connected Limited Device Configuration*) y *CDC* (*Connected Device Configuration*). Cada configuración soporta distintos perfiles como se muestra en la Figura 8.4. En particular, nos vamos a focalizar en las aplicaciones para celulares (o smartphones), para esto se debe utilizar la configuración *CLDC* y el perfil *MID* (*Mobile Information Device*).

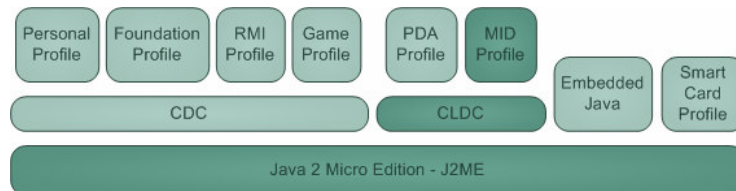


Figura 8.4: Plataforma J2ME.

Las aplicaciones que utilizan la configuración *CLDC* y el perfil *MID* se denominan aplicaciones *MIDlets*. Estas aplicaciones están conformadas por dos archivos, un archivo descriptor con extensión *.jad* y un archivo *.jar* conteniendo las clases de la aplicación y los recursos necesarios (por ejemplo: imágenes, archivos, etc.). En particular, al menos una de las clases de la aplicación debe extender de la clase *MIDlet*, esta clase es el punto de entrada de la aplicación. Una vez desarrollada la aplicación, se debe instalar el archivo *.jad* y *.jar* en el celular para que la aplicación se pueda ejecutar.

J2ME se define como un subconjunto de *J2SE* (*Java Standard Edition*), que a su vez es un subconjunto de *J2EE* (*Java Enterprise Edition*). *J2ME* además agrega una librería propia (*javax.microedition.**) para todos los conceptos relacionados con dispositivos móviles, que no están considerados en las aplicaciones desktop tradicionales. En la Figura 8.5 se puede apreciar la relación entre las distintas plataformas. La librería *javax.microedition.** contiene la clase *MIDlet* de la cual debe extender al menos una clase de la aplicación.

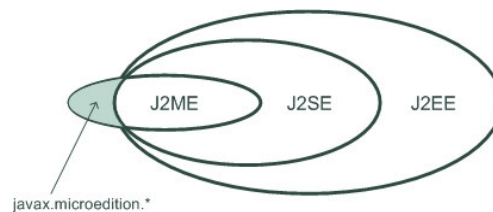


Figura 8.5: Relación entre J2ME, J2SE y J2EE.

La aplicación *J2ME* muestra la información en pantalla usando formularios (*javax.microedition.lcdui.Form*). Estos formularios contienen todos los elementos a ser mostrados en la pantalla. Para controlar las acciones que realiza el usuario, se definen comandos que se muestran en la pantalla como opciones. Además, se crean listeners (clases dedicadas a escuchar eventos) asociados a los comandos, los cuales esperan que el usuario seleccione una opción para realizar alguna acción.

En base a la configuración (*CLDC*) y el perfil (*MID*) además se pueden usar *APIs* propias de cada vendedor de celular (*Samsung, Nokia, Sony Ericsson, etc.*) y de esta manera poder desarrollar aplicaciones con funcionalidad específica para cada modelo de celular. Todas las *APIs* contienen librerías estándar y además agregan

funcionalidad propia. En particular las aplicaciones *J2ME* que deriva la herramienta usan sólo librerías estándar, para no tener problemas de portabilidad entre distintas marcas de celulares.

Una librería estándar es la *API* de Ubicación (*Location API – JSR 179*⁶⁴), esta librería permite el manejo de los datos del *GPS* para crear aplicaciones sensibles a la ubicación. En la Figura 8.6 se pueden apreciar las clases involucradas en la *API* de Ubicación. Para recibir la información de cambio de posición, una clase debe implementar la interfaz *LocationListener*. Si se quiere recibir información de cercanía, la clase debe definir la interfaz *ProximityListener*. El punto de entrada de esta *API* es a través del *LocationProvider*.

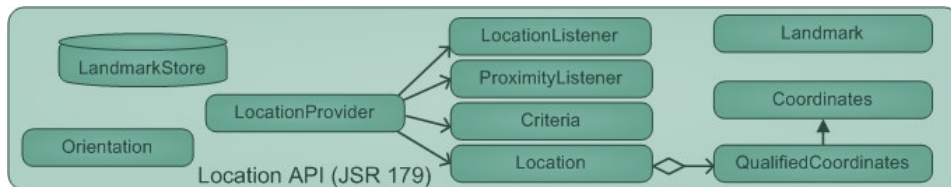


Figura 8.6: Clases de la API de Ubicación.

En la Figura 8.7 se puede apreciar las clases de una aplicación *J2ME* creadas para representar los conceptos específicos de una aplicación de HM. El punto de entrada de la aplicación es la clase *MobileHypermediaMIDlet*, la cual extiende de la clase *MIDlet*.

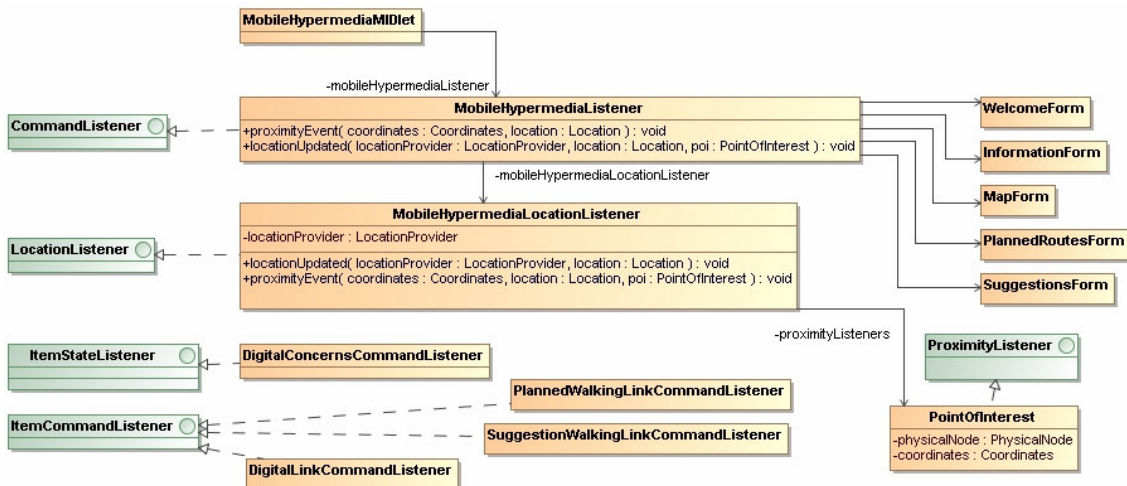


Figura 8.7: Clases específicas de la aplicación *J2ME*.

Se puede observar, en la Figura 8.7, que se definen distintos formularios acordes a la información que se le debe mostrar al usuario en cada situación, estos son:

- **WelcomeForm**, sirve para mostrar el mensaje de bienvenida.
- **InformationForm**, muestra la información digital del Punto de Interés, junto a sus links digitales y los concerns digitales del mismo (los cuales son usados para cambiar de concern).
- **MapFrom**, sirve para visualizar el mapa que se le muestra al usuario.
- **PlannedRoutesForm**, muestra la lista de links caminables predefinidos.
- **SuggestionsForm**, muestra la lista de links caminables derivados.

Además, se definen distintos listeners que reaccionan a diferentes eventos, como son:

⁶⁴ Página de *JSR 179*: <http://jcp.org/en/jsr/detail?id=179>

- **MobileHypermediaListener**, es un listener que centraliza todas las acciones de la aplicación. El resto de los listener le pasan el control a este listener ya que es el encargado de mostrar la información adecuada al usuario según el evento que ocurrió.
- **DigitalConcernsCommandListener**, está a la espera de que el usuario cambie de concern digital. Cuando el usuario cambia de concern, envía un aviso al listener *MobileHypermediaListener* para que éste le actualice la información acorde al concern elegido.
- **DigitalLinkCommandListener**, está a la espera de que el usuario seleccione un link digital, para enviar un aviso al listener *MobileHypermediaListener* de manera que éste le actualice la información con el target del link digital.
- **PlannedWalkingLinkCommandListener**, está a la espera de que el usuario seleccione un link caminable predefinido, para enviar un aviso al listener *MobileHypermediaListener* y éste le calcule el camino y se lo muestre en un mapa.
- **SuggestionWalkingLinkCommandListener**, ídem al anterior pero para los links caminables derivados.
- **MobileHypermediaLocationListener**, está a la espera de que el usuario cambie de ubicación (para enviar un aviso al listener *MobileHypermediaListener* y que actualice la información al usuario). Esta clase implementa la interfaz *LocationListener* (de la API de ubicación).
- **PointOfInterest**, esta clase implementa la interfaz *ProximityListener* (de la API de ubicación). Cuando el usuario entra en el radio de alcance de este Punto de Interés, esta clase recibe una notificación. Como respuesta de la notificación envía un aviso al listener *MobileHypermediaListener* para que actualice la información del usuario acorde al nuevo Punto de Interés.

En la Figura 8.8 se puede apreciar la relación que existe entre las clases específicas de la aplicación *J2ME* (Figura 8.7) y la API de Ubicación. Además, se pueden visualizar las relaciones con las clases del modelo básico de la aplicación (Figuras 8.1, 8.2 y 8.3), por ejemplo, el formulario del mapa conoce un *GoogleStaticMap* para generar a partir del mismo la imagen que se le debe mostrar al usuario.

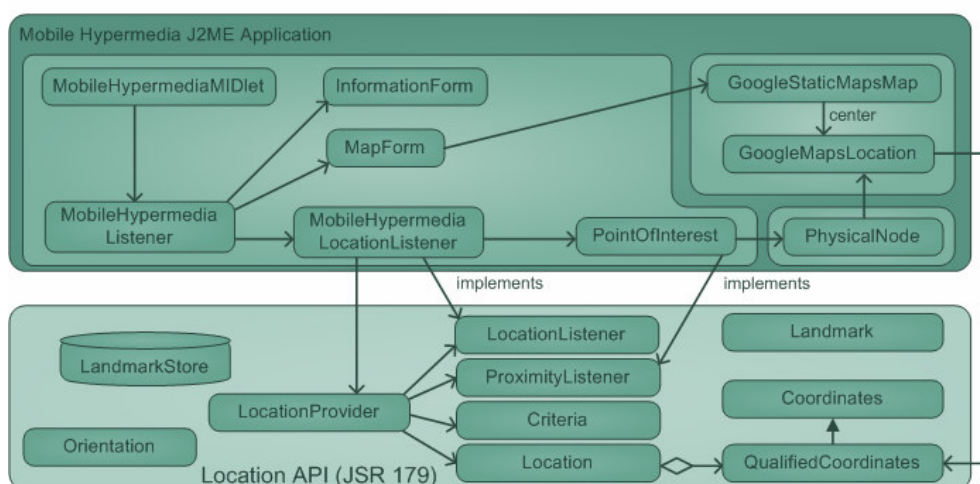


Figura 8.8: Relación entre la aplicación *J2ME* y la API de Ubicación.

La clase *PointOfInterest* implementa la interfaz *ProximityListener* definiendo la cercanía en base a una determinada coordenada. Esta coordenada la toma de la ubicación del nodo físico (*PhysicalNode*) que conoce. La clase *PointOfInterest* actúa como un *Wrapper* [Gama et al., 1995] del nodo físico agregando la funcionalidad para

detectar cuándo el usuario está cerca del Punto de Interés.

Se puede observar que la aplicación *J2ME* se definió en forma general para representar las características de las aplicaciones de HM, independientemente del contenido (o información) a mostrar. Este tipo de aplicación *J2ME* está definida para permitir que cuando un usuario esté cerca de un Punto de Interés visualice:

- Su información digital, sus links digitales y sus concerns digitales. Permitiéndole navegar digitalmente y cambiar de concerns.
- Un mapa junto a los links caminables predefinidos y derivados. Permitiéndole navegar en el mundo real (seleccionar un link caminable y el sistema le calcula el camino al target, cuando llega al target se completa esta navegación).

La aplicación descrita en esta sección se usa como base para la derivación de cada una de las aplicaciones *J2ME* que genera la herramienta.

8.3. Aplicación de Hipermedia Móvil – Web Móvil

Para el desarrollo de la aplicación Web Móvil (que cumpla con las características de las aplicaciones de HM) se optó por una solución simple como es el uso de *Servlets*⁶⁵ y *JSPs*⁶⁶ (*Java Server Pages*). Los *Servlets* son los encargados de resolver los requerimientos (Web) realizados por los usuarios mediante un Browser, mientras que las *JSPs* permiten la visualización de la información adecuada.

Este tipo de aplicación (Web Móvil) requiere que los usuarios envíen un requerimiento al servidor para obtener la información del Punto de Interés. Para realizar esto se puede, por ejemplo, leer un código de barra que tenga la *URL* correspondiente al Punto de Interés y recibir como respuesta la información del mismo. La ubicación del usuario se actualiza cada vez que lee un código de barra. Es decir, hasta que no se lee un código de barra no se puede determinar donde se encuentra el usuario. Esta forma de posicionamiento es imprecisa cuando, por ejemplo, el usuario realiza una navegación en el mundo real ya que no se lo puede asistir mientras realiza el camino, a menos que lea un código de barra intermedio.

La aplicación Web Móvil se desarrolló para correr sobre un servidor *Tomcat*⁶⁷ (versión 5.5), por simplicidad se optó por crear un *Servlet* por cada requerimiento diferente que realiza el usuario. Esto facilita la resolución del requerimiento ya que, según la *URL* del requerimiento, es el *Servlet* que se ejecuta. Cada *Servlet* es el punto de entrada a la aplicación, luego se le pasa el control al *Dispatcher*, el cual arma el *DTO* (*Data Transfer Object*) correspondiente y luego redirecciona el control a la *JSP* adecuada.

Los *DTOs* representan clases que no tienen comportamiento sino que definen variables con setters y getters. Estas clases sirven para el intercambio de información entre los diferentes componentes de la aplicación.

Las *JSPs* toman los datos del *DTO* para generar la página que visualiza el usuario. Algunas *JSPs* le muestran al usuario sólo información, otras, imágenes de mapa.

En la Figura 8.9 se pueden observar todos los *Servlets* creados para la aplicación (Web Móvil), todos extienden de la clase *Servlet*, la cual define el comportamiento común que deben tener los *Servlets*. Se pueden visualizar las cuatro *JSPs* que se crearon y los *DTOs* que se necesitaron especificar, también se puede observar en el diagrama la clase *Dispatcher*.

⁶⁵ Página con información sobre *Servlet*:

<http://www.oracle.com/technetwork/java/servlet-138661.html>

⁶⁶ Página con información sobre *JSP*:

<http://www.oracle.com/technetwork/java/javaee/jsp/index.html>

⁶⁷ Página de *Tomcat*: <http://tomcat.apache.org/>

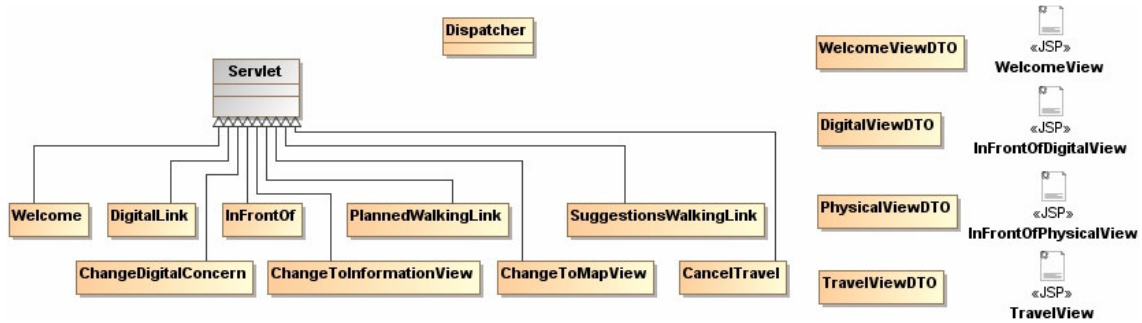


Figura 8.9: Clases creadas para la aplicación Web Móvil.

Veamos en detalle que función cumple cada *Servlet* dentro de la aplicación:

- **Welcome**, es el punto inicial de la aplicación y se encarga de crear todos los datos necesarios para que la aplicación funcione correctamente. Este *Servlet* le pasa el control al *Dispatcher*, éste arma el *WelcomeViewDTO* y redirecciona el control a la *JSP WelcomeView*.
- **InFrontOf**, es invocado cada vez que el usuario está frente a un Punto de Interés y lee, por ejemplo, un código de barra. Este *Servlet* busca la información del nuevo Punto de Interés (guardándolo en el historial del usuario) y luego le pasa el control al *Dispatcher* que se encarga de armar el *DTO* correspondiente según la vista actual del usuario.
 - Si el usuario estaba en la vista de información, el *Dispatcher* arma un *DigitalViewDTO* y redirecciona el control a la *JSP InFrontOfDigitalView*.
 - Si el usuario estaba en la vista del mapa (sin un camino establecido), el *Dispatcher* arma un *PhysicalViewDTO* y redirecciona el control a la *JSP InFrontOfPhysicalView*.
 - Si el usuario estaba en la vista del mapa (con un camino establecido), el *Dispatcher* arma un *TravelViewDTO* y redirecciona el control a la *JSP TravelView*.
- **ChangeDigitalConcern**, cuando el usuario cambia de concern digital, se invoca este *Servlet* que busca la información del concern seleccionado (guardándolo en el historial del usuario) y le pasa el control al *Dispatcher*, el cual arma un *DigitalViewDTO* con la información del concern seleccionado y redirecciona el control a la *JSP InFrontOfDigitalView*.
- **DigitalLink**, es invocado cuando el usuario selecciona un link digital, este *Servlet* busca la información del target (guardándolo en el historial del usuario) para luego pasarle el control al *Dispatcher*, que arma con esta información un *DigitalViewDTO* y luego redirecciona a la *JSP InFrontOfDigitalView*.
- **PlannedWalkingLink**, se ejecuta cuando el usuario selecciona un link caminable predefinido, este *Servlet* busca el target (guardándolo en el historial del usuario) y redirecciona el control al *Dispatcher* que arma un *TravelViewDTO* e invoca a la *JSP TravelView*.
- **SuggestionWalkingLink**, se ejecuta cuando el usuario selecciona un link caminable derivado, este *Servlet* busca el target (guardándolo en el historial del usuario) y redirecciona el control al *Dispatcher* que arma un *TravelViewDTO* e invoca a la *JSP TravelView*.
- **CancelTravel**, cuando el usuario decide cancelar un camino, se ejecuta este *Servlet*, dejando el Modelo del Usuario en un estado consistente y redirecciona el control al *Dispatcher*, este arma un *PhysicalViewDTO* con la información del Punto de Interés actual y luego pasa el control a la *JSP*

InFrontOfPhyscialView.

- **ChangeToInformationView**, este *Servlet* es ejecutado cada vez que el usuario decide pasar de la vista del mapa a la vista de información. Este *Servlet* busca la contraparte digital y le pasa el control al *Dispatcher* que arma un *DigitalViewDTO*, con la información de la contraparte digital y luego redirecciona el control a la *JSP InFrontOfDigitalView*.
- **ChangeToMapView**, este *Servlet* es ejecutado cada vez que el usuario decide pasar de la vista de información al mapa. Este *Servlet* busca la contraparte física, y le pasa el control al *Dispatcher* que arma el mapa acorde a última información física que estuvo visualizando el usuario.
 - Si el usuario tenía la vista del mapa sin un camino establecido, el *Dispatcher* arma un *PhysicalViewDTO*, y redirecciona el control a la *JSP InFrontOfPhyscialView*.
 - Si el usuario tenía la vista del mapa con un camino establecido, el *Dispatcher* arma un *TravelViewDTO*, y redirecciona el control a la *JSP TravelView*.

El *DigitalViewDTO* contiene la información digital del Punto de Interés junto a sus links digitales y los concerns digitales del mismo. El *PhysicalViewDTO* contiene la información de un mapa con la posición actual del usuario, junto a los links caminables predefinidos y derivados. El *TravelViewDTO* contiene la información del camino que debe realizar el usuario junto a su posición actual.

Tanto *PhysicalViewDTO* como *TravelViewDTO* interactúan con la clase *GoogleStaticMapAPI* (presentada en la Figura 8.3) para crear la imagen con el mapa que se le muestra al usuario. Además, *TravelViewDTO* interactúa con la clase *GoogleDirectionsAPI* (presentada en la Figura 8.3) para realizar la búsqueda de caminos.

Se puede observar que la descripción realizada (de la aplicación Web Móvil) se define en forma general para cualquier aplicación que cumpla con las características de una aplicación de HM. Este tipo de aplicación está definida para permitir que cuando el usuario está en un Punto de Interés, visualice su información digital, sus links digitales y sus concerns digitales. Además, cuando está en un Punto de Interés puede visualizar un mapa junto a los links caminables predefinidos y derivados, permitiéndole navegar digitalmente, cambiar de concerns y navegar en el mundo real (seleccionar un link caminable y el sistema le calcula el camino al target, cuando llega al target se completa esta navegación).

La aplicación descrita en esta sección se usa como base para la derivación de cada una de las aplicaciones de Web Móvil que genera la herramienta. En este tipo de aplicación, se detecta que el usuario está en un Punto de Interés, por ejemplo, porque leyó un código de barra que representa una *URL*. A partir de la misma se hace un requerimiento al servidor y como resultado se obtiene una página con la información del Punto de Interés.

8.4. Derivación de aplicaciones

Tanto en las Secciones 8.2 como 8.3 se mostraron dos tipos de aplicaciones genéricas que soportan las características básicas de las aplicaciones de HM. Estas aplicaciones sirven de base para hacer los dos tipos de derivaciones (*J2ME* y Web Móvil), la herramienta toma estas aplicaciones de base (presentadas en las Secciones 8.2 y 8.3) y le agrega la información de un Modelo de Instancias Navegacional particular para que la aplicación derivada quede completa.

8.4.1. Derivación de aplicaciones J2ME

Para derivar una aplicación J2ME, primero se obtuvieron dos archivos, uno llamado *MobileHypermediaApplication.jad* y otro llamado *MobileHypermediaApplication.jar*, de la aplicación J2ME básica presentada en la Sección 8.2, los cuales contienen toda la funcionalidad descrita. Estos archivos son la base para hacer la derivación, y forman parte de la herramienta, se encuentran ubicados dentro del directorio: *MagicMobileUWE\GeneratedCode_aux\J2MEApplication*.

El diseñador visualiza la opción *Generate J2ME Application*, como se puede observar en la Figura 8.10. Una vez que el diseñador selecciona esta opción, se le muestra la lista de posibles Modelos del Usuario que puede derivar y el diseñador debe elegir uno para generar la aplicación J2ME (a partir de la información del mismo).

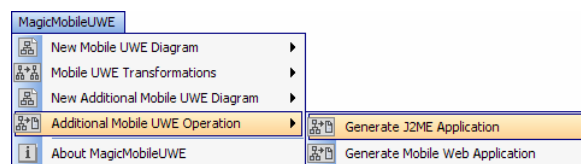


Figura 8.10: Generación de una aplicación J2ME.

La herramienta toma del Modelo del Usuario, el Modelo de Instancias Navegacional asociado y genera⁶⁸ un archivo llamado *NavigationModel.xml* con la información del mismo. A su vez se toman los archivos de la aplicación base *MobileHypermediaApplication.jad* y *MobileHypermediaApplication.jar* para crear dos archivos nuevos con la misma información, pero se los nombra acordes al nombre de la aplicación definida por el diseñador en el Modelo del Usuario. Al nuevo archivo .jar se le agrega el archivo *NavigationModel.xml*, para tener dentro de la aplicación la información propia del modelo que se derivó.

Si el diseñador eligió derivar un Modelo del Usuario cuya aplicación tiene como nombre “*Roma Application*”, la herramienta crea los dos nuevos archivos denominándolos *RomaApplication.jad* y *RomaApplication.jar*⁶⁹. Estos archivos se almacenan en una carpeta con el nombre *RomaApplication*. Esta carpeta es creada dentro del directorio: *MagicMobileUWE\GeneratedCode\J2MEApplications*.

Para que la aplicación base J2ME funcione en base a los datos contenidos en el archivo *NavigationModel.xml*, se debió agregar a la aplicación base un parser (procesador) que lea este archivo e instancie el Modelo Navegacional presentado en la Figura 8.1 con toda la información del mismo.

Además, a la aplicación base J2ME se le agregaron dos archivos más con sus correspondientes parsers. Uno de los archivos (*location.ini*) sirve para representar la ubicación inicial del usuario, este archivo por default, no tiene ningún dato en particular y el diseñador debe especificar el valor acorde a la aplicación derivada. Esto sirve particularmente para realizar la simulación de la aplicación.

El otro archivo se llama *GUIInformation.xml* y especifica toda la información textual que se debe mostrar al usuario en la aplicación, por ejemplo, el mensaje de bienvenida de la aplicación. Este archivo tiene definido información por default a mostrar, pero el diseñador una vez realizada la derivación puede modificar este archivo y cambiar los mensajes para ajustar la aplicación.

⁶⁸ Para pasar los elementos del Modelo de Instancias Navegacional a archivos se utilizan los estereotipos de los elementos. De esta manera se puede identificar las características de cada uno y agregar así su información en el archivo XML.

⁶⁹ El archivo *MobileHypermediaApplication.jar* de la aplicación J2ME base no cuenta internamente con un archivo llamado *NavigationModel.xml*.

En la Figura 8.11 se visualizan tanto los dos archivos (*location.ini* y *GUIInformation.xml*) agregados a la aplicación base *J2ME* como el archivo *NavigationModel.xml* agregado por la herramienta. Además, se detallan sus correspondientes parsers (*GUIInformationParser* y *NavigationModelParser*⁷⁰) para que dinámicamente se cargue toda la información tanto de la ubicación inicial, los datos del Modelo Navegacional, como la información propia de la *GUI* (*Graphical User Interface*).

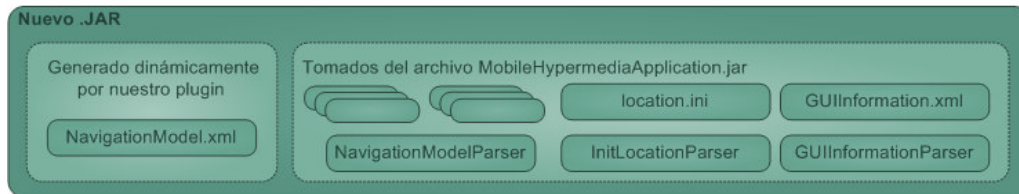


Figura 8.11: Archivos con información y sus correspondientes parsers.

Una vez generada la derivación, el diseñador recibe un mensaje indicando si la derivación se realizó, o no, en forma correcta, en caso afirmativo el diseñador cuenta con los archivos .jar y .jar con la información propia del Modelo del Usuario seleccionado, y estos archivos son los que se utilizan para correr la aplicación en un celular.

El diseñador debe indicar en el archivo *location.ini* (contenido en el archivo .jar), cuál es la posición inicial de la aplicación, caso contrario inicialmente el usuario no se puede posicionar en el mapa. Esta información es fundamental cuando se utilizan simuladores para probar la aplicación derivada.

8.4.2. Derivación de aplicaciones Web Móvil

La herramienta cuenta con un archivo .war que contiene toda la funcionalidad descrita en la aplicación Web Móvil base presentada en la Sección 8.3. El archivo .war se denomina *MobileHypermediaApplication.war* y se encuentra ubicado en el directorio: *MagicMobileUWE\GeneratedCode_aux\MobileWebApplications*.

El diseñador visualiza, en el menú principal la opción *Generate Mobile Web Application* como se puede observar en la Figura 8.12, cuando selecciona esta opción, se le muestra una lista de posibles Modelos del Usuario que puede derivar, el diseñador debe elegir uno de estos para generar la aplicación *Web Móvil*.

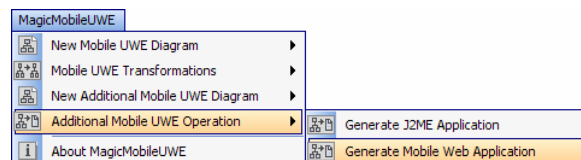


Figura 8.12: Generación de una aplicación Web móvil.

La derivación se realiza de manera muy similar a la derivación de la aplicación *J2ME*. La herramienta toma del Modelo del Usuario, el Modelo de Instancias Navegacional asociado y los datos del mismo son pasados al archivo llamado *NavigationModel.xml*. A su vez se toma el archivo de la aplicación base *MobileHypermediaApplication.war*⁷¹,

⁷⁰ Este parser lee los datos del archivo *NavigationModel.xml* e instancia objetos del modelo presentado en la Figura 7.1.

⁷¹ El archivo *MobileHypermediaApplication.war* de la aplicación Web Móvil de base no cuenta internamente con un archivo llamado *NavigationModel.xml*.

para crear otro archivo nuevo con la misma información. Al nuevo archivo .war se lo nombra acorde al nombre de la aplicación definida por el diseñador en el Modelo del Usuario, y se le agrega el archivo *NavigationModel.xml* para tener dentro de la aplicación la información propia del modelo que se derivó.

Si el diseñador eligió derivar un Modelo del Usuario cuya aplicación se llama *Theater Application*, la herramienta crea el nuevo archivo denominándolo *TheaterApplication.war*. Este archivo se almacena en una carpeta con el nombre *TheaterApplication*. Esta carpeta es creada dentro del directorio: *MagicMobileUWE\GeneratedCode\MobileWebApplications*.

Para que la aplicación Web Móvil base funcione con los datos contenidos en el archivo *NavigationModel.xml*, se debió agregar, al igual que para la aplicación base *J2ME*, un parser que lea este archivo e instancie el Modelo Navegacional presentado en la Figura 8.1 con toda la información del mismo.

También se agregaron a la aplicación Web Móvil base, los archivos *location.ini* y *GUIInformation.xml* (como en la aplicación *J2ME* base), con sus correspondientes parsers.

En particular se agregó a la aplicación Web Móvil base, el archivo *ResourcesURL.xml* que describe:

- El nombre de la aplicación. Tener desacoplado el nombre de la aplicación permite que luego las *URLs* que se generan para encontrar los recursos de la aplicación sean dinámicas y se construyan en base al valor especificado. De esta manera no se restringe a un determinado nombre de aplicación.
- La ubicación de la carpeta donde están ubicadas las *JSPs*. Si bien se define un valor por default, el diseñador puede variar la ubicación de las mismas.

Supongamos que se asigna en el archivo *ResourcesURL.xml* como nombre de aplicación *TheaterApplication*. Esto quiere decir que las *URLs* relacionadas a los recursos de la aplicación comienzan de la siguiente manera:

```
http://xx.xx.xx.xx:8080/TheaterApplication/
```

La aplicación Web Móvil base accede a los *Servlets* y *JSPs*, construyendo de manera dinámica la *URL* de los mismos usando el nombre de la aplicación definido en el archivo *ResourcesURL.xml*, por ejemplo, el *Servlet* de bienvenida se encuentra de la siguiente manera:

```
http://xx.xx.xx.xx:8080/TheaterApplication/Welcome
```

De esta manera, al configurar el nombre de la aplicación desde un archivo el diseñador puede asignarle a la misma, el nombre que considere adecuado. Además tener desacoplada esta información en este archivo (*ResourcesURL.xml*) permite que sea información que puede modificar el diseñador sin necesidad de compilar código *Java*.

El diseñador debe tener en cuenta que el nombre de la aplicación que se especifique en el archivo *ResourcesURL.xml*, debe corresponderse con el nombre del archivo .war que se copie en el servidor *Tomcat*. En el caso de no coincidir los nombres no se van a encontrar los recursos de la aplicación y se generará un error en ejecución (cuando el usuario use la aplicación).

En la Figura 8.13 se pueden observar los archivos que contienen la aplicación resultante y sus parser correspondientes. Se puede observar que también se agregó el parser correspondiente para leer estos datos dinámicamente del archivo *ResourcesURL.xml*.

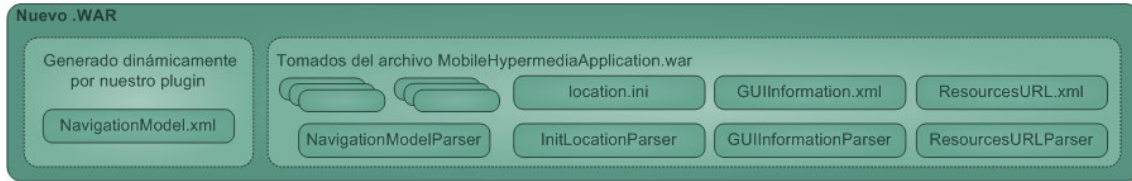


Figura 8.13: Archivos con información y sus correspondientes parsers.

Una vez generada la derivación el diseñador recibe un mensaje indicando si la derivación se realizó, o no, en forma correcta. En el caso afirmativo, el diseñador cuenta con un archivo .war con la información propia del Modelo del Usuario seleccionado. Este archivo es el que se utiliza para almacenarlo en la carpeta `webapps` del *Tomcat*, para que la aplicación quede funcionando y sea accedida mediante un Browser desde los dispositivos móviles.

En este caso el diseñador también debe indicar en el archivo *location.ini* (contenido en el archivo .war) cuál es la posición inicial, caso contrario inicialmente el usuario no se puede posicionar en el mapa. Esto es fundamental para poder realizar las simulaciones. Cuando el usuario accede a la *JSP* de bienvenida de la aplicación se le va a indicar que está parado en la ubicación indicada en ese archivo (caso contrario se le muestra un mapa sin una ubicación en particular).

Además, el diseñador debe verificar la información del archivo *ResourcesURL.xml* en el caso de que modifique la ubicación de las *JSPs* o cambie el nombre de la aplicación.

8.4.3. Consideraciones en ambas derivaciones

La creación del archivo *NavigationModel.xml* en ambos tipos de derivaciones (aplicaciones *J2ME* y aplicaciones Web Móvil) se realiza de la misma manera. Se recorren los elementos del Modelo de Instancias Navegacional⁷² (que se corresponde con el Modelo del Usuario que seleccionó el diseñador para derivar la aplicación) y según el estereotipo del elemento es la información que se especifica en el archivo (*NavigationModel.xml*).

Durante la creación del archivo *NavigationModel.xml*, se realizan diferentes controles y, en el caso de encontrar posibles problemas en el Modelo Navegacional, se genera un archivo llamado *warnings.txt* (contenido en la misma carpeta donde se crean los archivos de la aplicación generada) describiendo las problemáticas encontradas. Durante la creación del archivo *NavigationModel.xml* se realizan los siguientes controles:

- Si los concerns digitales tienen nombre, caso contrario se detalla en el archivo *warnings.txt* que un concern no tiene nombre y por lo tanto sus elementos no son pasados al archivo *NavigationModel.xml*. Esto sirve para prevenir futuros problemas, ya que sus elementos no pueden indicar el nombre del concern al que pertenecen. Esto afecta visualmente cuando se listan los nombres de los concerns para que el usuario pueda cambiar a uno de ellos.
- Si las instancias tienen nombre, caso contrario la instancia no es pasada al archivo *NavigationModel.xml* ya que no hay forma de referenciarla posteriormente.
- Si las instancias tienen definido un nodo base, caso contrario no es pasada al archivo *NavigationModel.xml* ya que no se puede determinar el

⁷² Sólo son pasados al archivo *NavigationModel.xml*: los concerns, las instancias de los nodos (con sus propiedades) y sus links. Cualquier otro elemento no es incorporado en este archivo.

comportamiento de la misma como así tampoco las propiedades que tiene.

- Si las instancias de un nodo físico tienen más de una propiedad de ubicación estereotipada como geométrica, se le indica esto al diseñador para que lo considere (esta situación puede afectar el funcionamiento de la aplicación resultante).
- Si las instancias de un nodo físico tienen más de una propiedad de ubicación estereotipada como etiquetada, se le indica esto al diseñador para que lo considere (esta situación puede afectar el funcionamiento de la aplicación resultante).
- Si las instancias de un nodo físico tienen al menos definida una propiedad de ubicación, caso contrario se le indica al usuario que ese elemento no va a poder posicionarse en la aplicación resultante.
- Si las instancias de los nodos digitales tienen definidas la propiedad `name` con un determinado valor, caso contrario se le avisa al diseñador para que tenga en cuenta esta situación. No se puede establecer links a esta instancia ya que no hay forma de listársela al usuario. Esta situación también trae problemas si hay links caminables en su contraparte física, ya que no se puede determinar que nombre listarle al usuario en el link caminable.

El diseñador debe tener en cuenta que actualmente la herramienta sólo pasa al archivo *NavigationModel.xml* propiedades de los nodos definidas como *Strings*. Cualquier otra propiedad definida con otro tipo no es pasada a este archivo.

Los modelos base presentados en la Sección 8.1 fueron ajustados, acorde a la plataforma sobre la cual se desarrolla la aplicación. Esto quiere decir, que la plataforma *J2ME* no provee la misma funcionalidad que la plataforma *J2EE* usada para el desarrollo de la aplicación Web Móvil. La principal diferencia se ve en el manejo de colecciones, en *J2ME* se utiliza la clase *Vector* mientras que en *J2EE* se deben usar colecciones que implementen la interfaz *List* para no tener conflicto cuando se muestran los datos en las *JSPs*. Otra cuestión es que *J2EE* usa *Generic* para la definición de colecciones, mientras que en *J2ME* no existen. Por otro lado, el manejo de conexiones *HTTP* varía de una plataforma a otra (es decir, varía de *J2EE* a *J2ME*), por lo tanto se hicieron los ajustes necesarios, tanto en la clase *GoogleStaticMapsAPI* como en la clase *GoogleDirectionsAPI*.

8.4.4. Ejemplos de aplicaciones derivadas

A continuación se muestran distintos ejemplos de aplicaciones derivadas. Para cada ejemplo, se indica cuál fue el Modelo de Instancias Navegacional utilizado en la derivación, y las pantallas resultantes que visualiza el usuario en su dispositivo móvil, cuando utiliza las aplicaciones generadas.

8.4.4.1. Aplicaciones *J2ME*

En la Figura 8.14 se puede observar un ejemplo de un Modelo de Instancias Navegacional que se corresponde⁷³ con el Modelo Navegacional de la aplicación de la ciudad de *Roma* presentado en la Sección 5.4.

Se puede observar que las instancias contienen tanto información propia como links. En particular, las instancias de los nodos físicos tienen definidas la propiedad de `location` como geométrica.

⁷³ Se modificó en el Modelo Navegacional el estereotipo especificado para la propiedad de ubicación, definiéndola como geométrica.

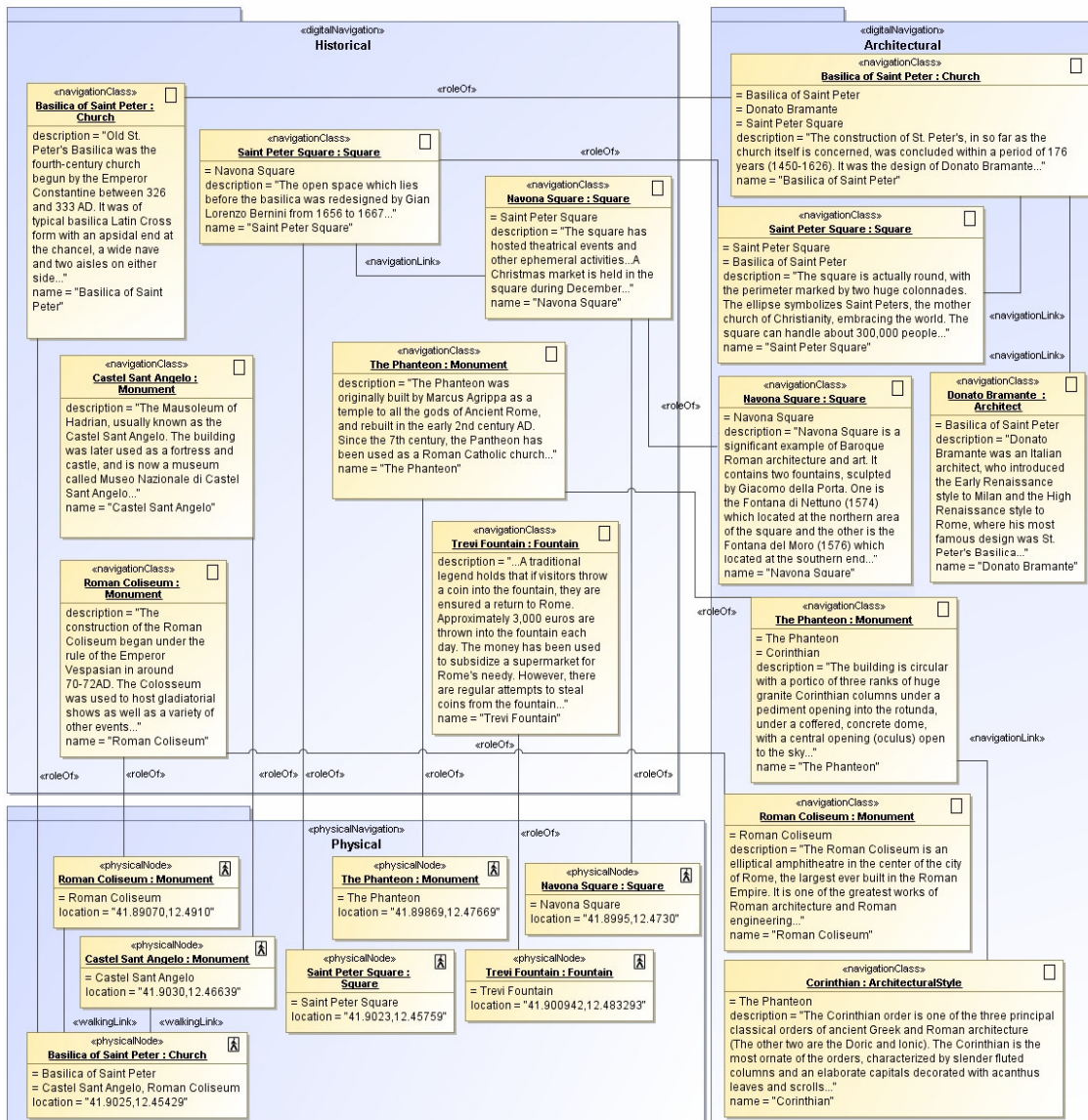


Figura 8.14: Un Modelo de Instancias Navegacional del ejemplo de la ciudad de Roma.

Supongamos que se crea luego un Modelo del Usuario que contiene el Modelo de Instancias Navegacional presentado en la Figura 8.14. En este Modelo del Usuario, se especifica como nombre de la aplicación "*Roma Application*" y luego se elige la opción para generar una aplicación *J2ME*, seleccionando este Modelo del Usuario, el cual es usado para derivar la aplicación. Como resultado de la derivación, se obtienen dos archivos, uno llamado *RomaApplication.jad* y otro *RomaApplication.jar*, los cuales especifican la aplicación *J2ME* que contiene la información definida en el Modelo de Instancias Navegacional (Figura 8.14). Estos archivos creados se copian directamente en el celular (que debe permitir correr aplicaciones *Java*) y luego se instala la aplicación. El celular debe tener *GPS* para poder detectar que el usuario está cerca de un Punto de Interés y mostrarle la información del mismo.

Para mostrar en esta tesis que las aplicaciones *J2ME* generadas por la herramienta funcionan correctamente se eligió probarlas usando un emulador provisto por *Samsung*⁷⁴. Este emulador tiene la característica de permitir definir emular el

⁷⁴ Página del emulador de *Samsung* (La versión utilizada para esta tesis es la que provee la SDK de *Samsung* 1.2.1): <http://innovator.samsungmobile.com>

movimiento del usuario y, por lo tanto, simular el funcionamiento del *GPS*⁷⁵ cuando el usuario se mueve con su dispositivo.

A continuación se muestran distintas pantallas que reflejan la interacción del usuario con la aplicación funcionando⁷⁶.

Supongamos que el usuario está caminando⁷⁷ por la ciudad de *Roma*, cuando pasa cerca⁷⁸ de la *Basílica de San Pedro* recibe la pantalla de la Figura 8.15.a. Cuando el usuario está en la vista de información tiene la opción *View Map* que le permite cambiar a la vista del mapa, imaginemos que elige cambiar a esta vista (del mapa), pasa a ver una imagen como se muestra en la Figura 8.15.b. Una vez que está en la vista del mapa (Figura 8.15.b) el usuario tiene disponible la opción *Menú*, que le provee tres alternativas: volver a la vista de información (Figura 8.15.a), ver los links caminables predefinidos (Figura 8.15.c) o ver los links caminables derivados (Figura 8.15.d). Para este ejemplo se puede visualizar que hay dos links predefinidos relacionados a la *Basílica de San Pedro*, uno al *Castillo San Ángel* y otro al *Coliseo*. No se pudieron derivar links caminables porque no hay links digitales.



Figura 8.15: El usuario pasa cerca de la *Basílica de San Pedro*.

El usuario decide volver a la vista de información de la *Basílica* (Figura 8.16a⁷⁹) porque desea ver más información de la misma. Como la *Basílica* tiene información arquitectural asociada, el usuario puede cambiar de concern y pasar a ver esta información como se muestra en la Figura 8.16.b (cambio al concern arquitectural). Luego el usuario decide cambiar a la vista del mapa (Figura 8.16.c) y al visualizar el

⁷⁵ Se define un archivo .xml que contiene una lista de pares (latitud, longitud) que simulan el cambio de posición del usuario, cuando el emulador lee cada par, se actualiza con esta información la posición actual del usuario. La integración de la *API* de Ubicación (JSR-179) a la aplicación *J2ME* permite que cada vez que se cambia la posición del usuario se actualice la información de la aplicación según corresponda.

⁷⁶ Para una mejor comprensión de la simulación se destacaron todos los Puntos de Interés de la aplicación con un globo gris claro, el Punto de Interés donde está el usuario se marca con un globo gris oscuro, los links caminables predefinidos se destacan en azul y los links caminables derivados se destacan en amarillo.

⁷⁷ Para la simulación se creó un archivo .xml que contiene una lista de pares (latitud, longitud) que permiten emular el caminar del usuario por la ciudad de *Roma*.

⁷⁸ La cercanía al Punto de Interés lo detecta la aplicación a través de la clase *PointOfInterest*, que es listener de una determinada coordenada. Cuando el usuario está cerca de esa coordenada, esta clase recibe un aviso y le pasa el control al *MobileHypermediaListener* para mostrar en este caso la información de la *Basílica de San Pedro*.

⁷⁹ Es la misma que la Figura 7.15.a se repitió para que sea más fácil seguir el ejemplo.

mapa el usuario puede observar que le apareció (en la Figura 8.16.d) un link caminable derivado a la *Plaza de San Pedro* (ya que hay un link digital a la *Plaza de San Pedro* y además este lugar tiene una contraparte física, por lo tanto se puede derivar un link caminable). Los links caminables predefinidos no cambian (ya que el usuario sigue ubicado en la *Basilica*).



Figura 8.16: El usuario cambia de concern en la *Basilica de San Pedro*.

Si el usuario vuelve a ver la vista de información de la *Basilica* (Figura 8.17.a⁸⁰) puede ver dos links digitales (en el concern arquitectural). El usuario decide navegar digitalmente a la *Plaza de San Pedro*, en este caso recibe la información mostrada en la Figura 8.17.b. Si cambia a la vista del mapa (Figura 8.17.c) puede apreciar que no tiene ningún link caminable derivado. Esto se debe a que si bien hay un link digital (Figura 8.17.b) es a la *Basilica*, la aplicación detecta esta situación, y no agrega ningún link caminable derivado (Figura 8.17.d). Los links caminables predefinidos no cambian ya que el usuario sigue parado en el mismo lugar (*Basilica*).



Figura 8.17: El usuario navega digitalmente a la *Plaza de San Pedro*.

El usuario vuelve a ver la información de la *Plaza de San Pedro* (Figura 8.18.a⁸¹) y decide cambiar de concern para ver el concern histórico de este Punto de Interés (Figura 8.18.b). Luego al cambiar a la vista del mapa (Figura 8.18.c) puede visualizar

⁸⁰ Es la misma que la Figura 7.16.b se repitió para que sea más fácil seguir el ejemplo.

⁸¹ Es la misma que la Figura 7.17.b se repitió para que sea más fácil seguir el ejemplo.

que se le agregó un link caminable derivado a la *Plaza Novona* (Figura 8.18.d). En cuanto a los links caminables predefinidos, no cambian ya que el usuario sigue ubicado en la *Basílica*.



Figura 8.18: El usuario cambia de concern en la Plaza de San Pedro.

Luego el usuario elige realizar el camino hacia la *Plaza Novona* (Figura 8.19a⁸²) entonces el sistema le calcula un camino y se lo muestra como se visualiza en la pantalla de la Figura 8.19.b. Se puede observar que el target destino está destacado y además se marca en el mapa el camino sugerido.

A medida que el usuario avanza en su camino, se le actualiza su posición en el mapa como se puede apreciarse en la Figura 8.19.c. La información del *GPS* permite actualizar la posición del usuario en el mapa.



Figura 8.19: El usuario elige caminar a la Plaza Novona.

El usuario sigue caminando hasta que en un momento determinado está cerca del destino, el sistema detecta esta situación y le muestra la información del destino. Finalizando de esta manera la navegación en el mundo real.

Veamos ahora otro ejemplo de derivación de una aplicación *J2ME*. En la Figura 8.20 se puede observar un Modelo de Instancias Navegacional que se corresponde con el

⁸² Es la misma que la Figura 7.18.d se repitió para que sea más fácil seguir el ejemplo.

Modelo Navegacional⁸³ presentado en la Sección 7.1 (relacionado con la aplicación turística de la ciudad de *La Plata*).

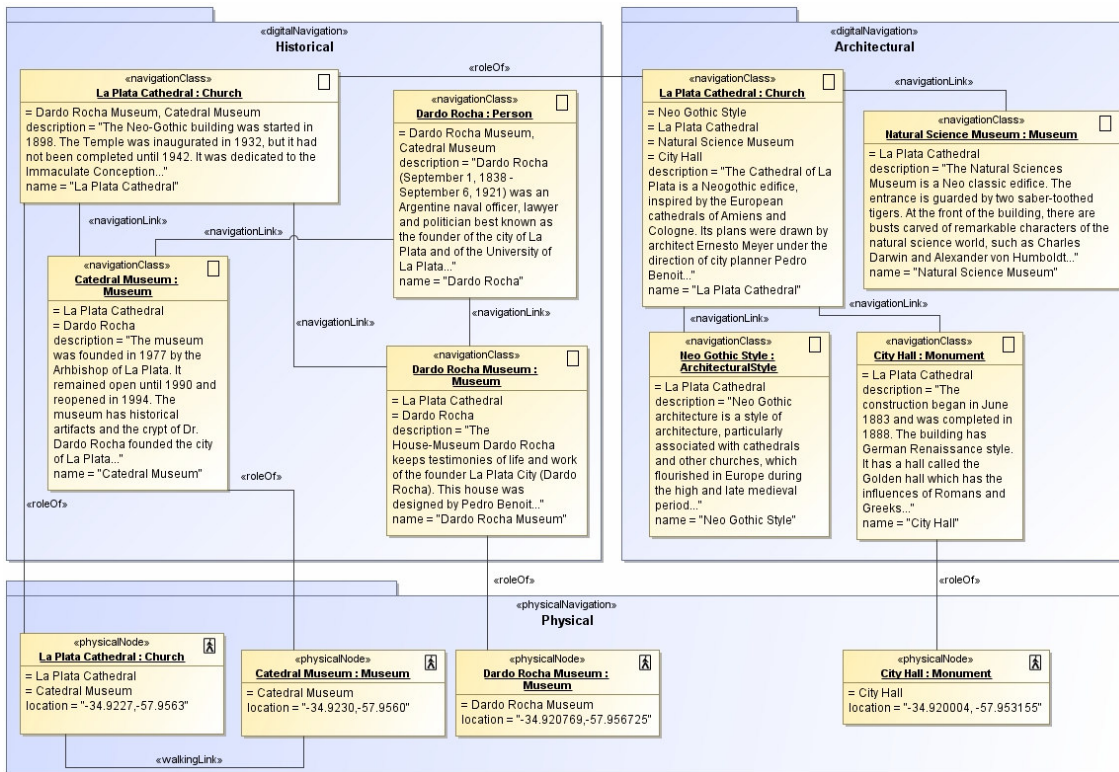


Figura 8.20: Un Modelo de Instancias Navegacional del ejemplo de la ciudad de *La Plata*.

Supongamos que el diseñador crea un Modelo del Usuario que contiene el Modelo de Instancias Navegacional presentado en la Figura 8.20. Como nombre de la aplicación (en el Modelo del Usuario) se especificó "*La Plata Application*". Luego el diseñador elige la opción para generar una aplicación *J2ME*, teniendo que seleccionar el Modelo del Usuario, el cual es usado para derivar la aplicación.

Los archivos creados como resultado de la derivación son *LaPlataApplication.jad* y *LaPlataApplication.jar*⁸⁴, este último contiene el archivo *NaviagionModel.xml* con la información definida en el Modelo de Instancias Navegacional de la Figura 8.20.

Nuevamente usaremos el emulador provisto por *Samsung* para probar la aplicación generada. Supongamos que el usuario está caminando⁸⁵ por la ciudad de *La Plata*, al pasar cerca del *Museo Dardo Rocha* recibe la pantalla de la Figura 8.21.a. En ese momento el usuario puede pasar de la vista de información a la vista del mapa (Figura 8.21.b) usando la opción *View Map* provista en la pantalla.

Una vez que está visualizando el mapa (Figura 8.21.b) el usuario tiene disponible la opción *Menú*, que le provee tres alternativas: volver a la vista de información (Figura 8.21.a), ver los links predefinidos (Figura 8.21.c) o ver los links caminables derivados (Figura 8.21.d).

⁸³ Se modificó en el Modelo Navegacional el estereotipo especificado para la propiedad de ubicación, definiéndola como geométrica.

⁸⁴ En este ejemplo se cambio la configuración por default de la aplicación que muestra el mapa de calles, indicando que además muestre la imagen satelital, es decir indicando que sea hibrido el mapa. Esto se realiza cambiando el archivo *GUIInformation.xml*. También se cambio el zoom por default (14) que tiene la aplicación creada, por un zoom de 16 así se puede apreciar mejor la información del ejemplo.

⁸⁵ Para la simulación se creó un archivo .xml que contiene una lista de pares (latitud, longitud) que permiten emular el caminar del usuario por la ciudad de *La Plata*.

En este ejemplo no hay links caminables predefinidos (para el *Museo Dardo Rocha*) y hay un link caminable derivado a la *Catedral de La Plata* (ya que hay un link digital a la *Catedral de La Plata* y además este lugar tiene una contraparte física, por lo tanto se puede derivar un link caminable). Notar que, el link digital a *Dardo Rocha* va a mostrar información del fundador de *La Plata*, por lo tanto no tiene contraparte física.



Figura 8.21: El usuario pasa cerca del *Museo Dardo Rocha*.

El usuario decide volver a la vista de información (Figura 8.22.a⁸⁶) y navegar digitalmente a la *Catedral de La Plata* (Figura 8.22.b). Luego pasa a visualizar la vista del mapa (Figura 8.22.c).

Se puede apreciar en la Figura 8.22.d que se listó un link caminable derivado al *Museo de La Catedral* (ya que hay un link digital al *Museo de La Catedral* y éste lugar tiene una contraparte física, por lo tanto se puede crear un link caminable derivado). Cabe recordar, que el *Museo Dardo Rocha* no tiene links caminables predefinidos.



Figura 8.22: El usuario navega digitalmente a *La Catedral de La Plata*.

El usuario decide caminar hacia el *Museo de La Catedral* (Figura 8.23.a⁸⁷), el sistema le calcula el camino y le muestra una pantalla como se visualiza en la Figura 8.23.b. A medida que el usuario avanza en su camino, el sistema actualiza su posición como se puede apreciar en la Figura 8.23.c y nunca pierde de vista el camino pautado.

⁸⁶ Es la misma que la Figura 7.21.a se repitió para que sea más fácil seguir el ejemplo.

⁸⁷ Es la misma que la Figura 7.22.d se repitió para que sea más fácil seguir el ejemplo.



Figura 8.23: El usuario elige caminar al *Museo de la Catedral*.

En estos dos ejemplos se puede apreciar cómo, a partir de dos modelos de instancias diferentes, la herramienta puede derivar dos aplicaciones *J2ME*. Estas aplicaciones tienen en común el mismo funcionamiento pero cada una muestra la información acorde a su Modelo de Instancias Navegacional. Es decir, tienen en común el funcionamiento de mostrar la información de un Punto de Interés cercano, cambiar de concerns, navegar digitalmente, visualizar en el mapa la posición del usuario y, en caso de estar en un Punto de Interés, visualizar sus links caminables predefinidos y links caminables derivados teniendo la posibilidad navegar en el mundo real (indicando el camino que debe realizar el usuario y cuál es el target destino). Toda la funcionalidad común a ambos ejemplos viene dada por el archivo *MobileHypermediaApplciation.jar*, el cual se usa en las derivaciones de este tipo.

8.4.4.2. Aplicaciones Web Móvil

En esta sección se muestran algunos ejemplos de derivación de aplicaciones Web Móvil usando la herramienta *MobileMagicUWE*. Veamos primero cómo usando la misma información definida en los dos ejemplos presentados en la sección anterior (8.4.4.1) se pueden derivar aplicaciones Web Móvil. En la Figura 8.24 se puede observar que se agregó al ejemplo presentado en la Figura 8.14, información referente a la etiqueta que identifica a cada Punto de Interés de *Roma*, el resto de la información del Modelo de Instancias Navegacional se mantiene igual.

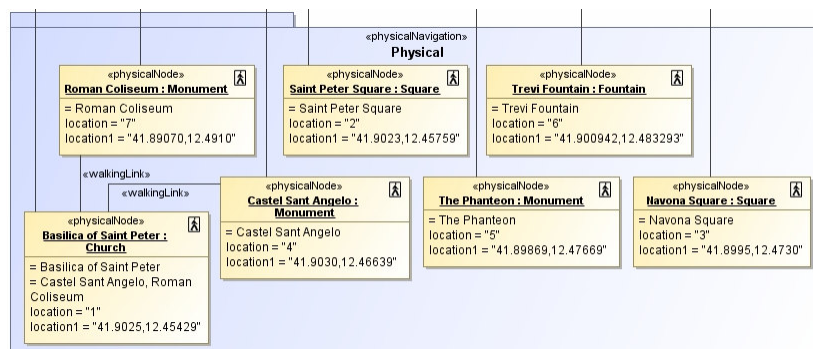


Figura 8.24: Un Modelo de Instancias Navegacional del ejemplo de la ciudad de *Roma*.

El diseñador crea un Modelo del Usuario que contiene el Modelo de Instancias Navegacional presentado en la Figura 8.24. En este modelo (del Usuario) se

especifica como nombre de la aplicación “*Roma Application*”. Luego el diseñador eligió la opción para generar una aplicación *Web Móvil* (a partir de este Modelo del Usuario). Como resultado, se obtiene un archivo llamado *RomaApplication.war*, el cual contiene la información definida en el Modelo de Instancias Navegacional (Figura 8.24). Este archivo se copia en la carpeta `webapps` del *Tomcat*⁸⁸ y cuando se enciende (o se reinicia) este servidor la aplicación queda funcionando para que el usuario la acceda mediante un Browser Web desde su celular.

Una forma de probar este tipo de aplicaciones, es tener por un lado instalado en el celular un programa que lea códigos de barra, y por otro, impreso distintos códigos de barras que describan la *URL* que se corresponde con cada Punto de Interés⁸⁹. Cuando el usuario usa el programa para leer el código de barra, éste envía un requerimiento al servidor (*Tomcat*) indicando en la *URL* el nombre de la aplicación y como parámetro el identificador del Punto de Interés. Como resultado (del requerimiento) se abre un Browser con la página correspondiente al Punto de Interés.

Para mostrar en esta tesis que las aplicaciones Web Móvil generadas por la herramienta funcionan correctamente se eligió probarlas usando el Browser *Opera Mobile*⁹⁰. Este Browser es específico para dispositivos móviles, y posee una versión desktop para realizar pruebas. Se indica en este Browser la *URL* que se corresponde con cada Punto de Interés para poder realizar así la simulación de la lectura del código de barra.

A continuación se pueden apreciar distintas pantallas que reflejan la interacción del usuario con la aplicación funcionando⁹¹. La distribución que recibe el usuario es muy similar a la presentada para las aplicaciones *J2ME*.

En la pantalla de la Figura 8.25.a el usuario recibe la información de la *Basílica de San Pedro*. Cuando el usuario pasar de la vista de información a la vista del mapa, visualiza la Figura 8.25.b donde además del mapa se le listan los links caminables predefinidos (para este ejemplo se puede visualizar los links *Castillo San Angelo* y *Coliseo*). Se puede apreciar que no hay ningún link caminable derivados.

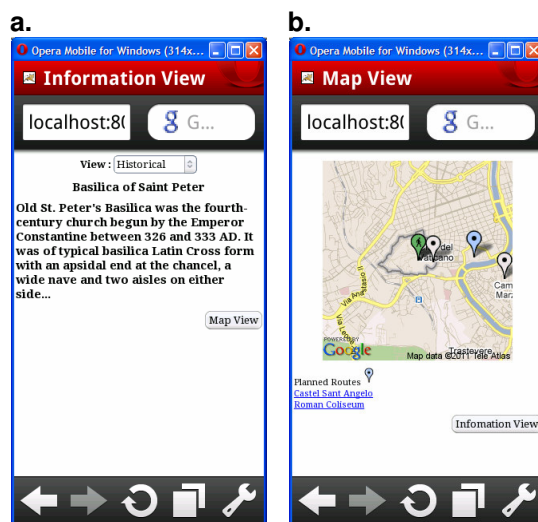


Figura 8.25: El usuario lee el código de barra de la *Basílica de San Pedro*.

⁸⁸ La versión de *Tomcat* usada para probar los ejemplos de esta tesis es 5.5.31.

⁸⁹ La *URL* tiene definido como parámetro el identificador del Punto de Interés.

⁹⁰ Página de *Opera Mobile*: <http://www.opera.com>

⁹¹ Para una mejor comprensión de la simulación se destacaron todos los Puntos de Interés de la aplicación con un globo gris claro. Cuando el usuario está cerca a un Punto de Interés, éste se marca con un globo gris oscuro. Los links caminables predefinidos se destacan en azul y los links caminables derivados se destacan en amarillo.

El usuario decide volver a la vista de información de la *Basílica* (Figura 8.26.a⁹²) y cambiar al concern arquitectural (Figura 8.26.b). Luego decide ver la vista del mapa (Figura 8.21.c) y puede observar que le apareció un link caminable derivado a la *Plaza de San Pedro* mientras que los links caminables predefinidos no cambian (ya que el usuario se encuentra ubicado en la *Basílica*).

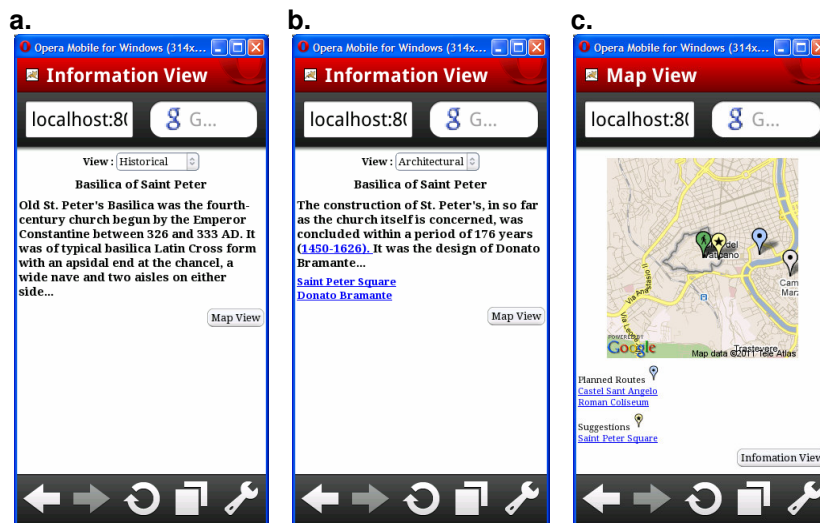


Figura 8.26: El usuario cambia de concern en la *Basílica de San Pedro*.

Al volver a la vista de información (Figura 8.27.a⁹³), el usuario decide navegar digitalmente a la *Plaza de San Pedro* (Figura 8.27.b) y luego cambiar al concern histórico de la plaza (Figura 8.27.c).

Al pasar a la vista del mapa (Figura 8.27.d) puede visualizar que se le listan los dos links caminables predefinidos relacionados a la *Basílica de San Pedro* y se derivó un link caminable a la *Plaza Novona*.

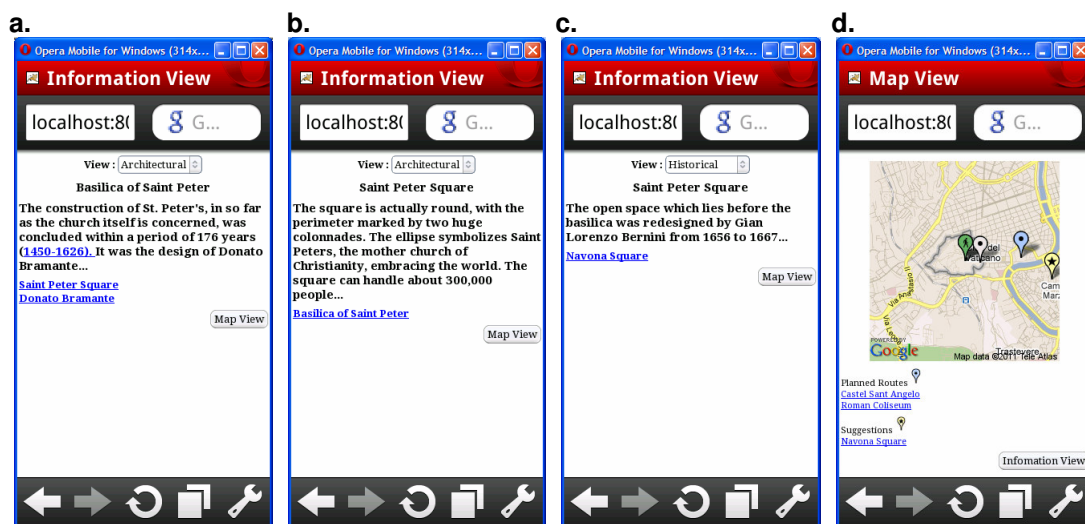


Figura 8.27: El usuario navega digitalmente a la *Plaza de San Pedro* y luego cambia de concern.

Supongamos que cuando el usuario está en la vista del mapa (Figura 8.28.a⁹⁴), elige caminar hacia la *Plaza Novona*, entonces el sistema le calcula un camino y se lo muestra como se visualiza en la pantalla de la Figura 8.28.b.

⁹² Es la misma que la Figura 7.25.a se repitió para que sea más fácil seguir el ejemplo.

⁹³ Es la misma que la Figura 7.26.b se repitió para que sea más fácil seguir el ejemplo.

⁹⁴ Es la misma que la Figura 7.27.d se repitió para que sea más fácil seguir el ejemplo.

En esta aplicación, la única manera de registrar que el usuario se mueve es que lea códigos de barras. Cuando lee un código de barra que se corresponde con su destino el usuario completa su navegación en el mundo real.

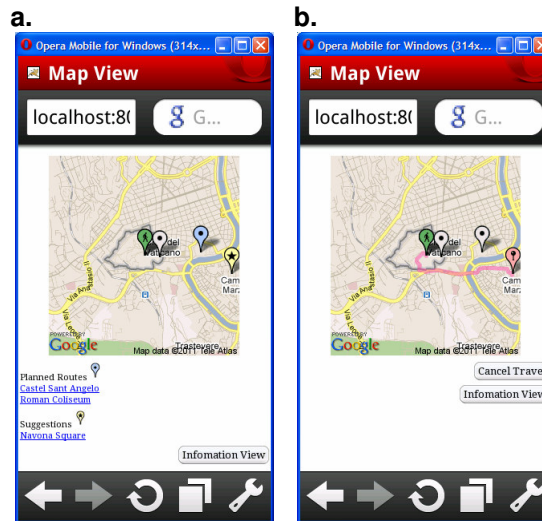


Figura 8.28: El usuario elige caminar a la *Plaza Novona*.

Veamos ahora otro ejemplo de derivación de una aplicación Web Móvil usando como base el ejemplo presentado en la Figura 8.20 al cual se le agrega la información correspondiente a la etiqueta que identifica a cada Punto de Interés de *La Plata*. Esta modificación se puede visualizar en la Figura 8.29, el resto de la información del modelo y los links se mantienen igual.

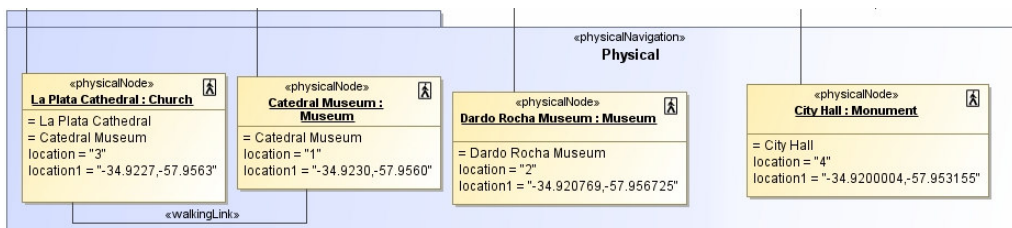


Figura 8.29: Un Modelo de Instancias Navegacional del ejemplo de la ciudad de *La Plata*.

El diseñador crea un Modelo del Usuario que contiene el Modelo de Instancias Navegacional presentado en la Figura 8.29 y como nombre de aplicación específica "*La Plata Application*". Al elegir la opción para generar una aplicación Web Móvil, se crea un archivo llamado *LaPlataApplication.war*. Este archivo contiene al archivo *NavigationModel.xml*, que se definió con la información del Modelo de Instancias Navegacional de la Figura 8.29.

El archivo *LaPlataApplication.war* se copia en la carpeta *webapps* del *Tomcat* y cuando se enciende (o se reinicia) este servidor (*Tomcat*) la aplicación queda funcionando para que el usuario la acceda mediante un Browser Web desde su celular.

A continuación, se muestran distintas pantallas que reflejan la información que recibe el usuario en su Browser cuando interactúa con la aplicación. Para la simulación, se uso el Browser *Opera Mobile* en el cual se ingresa la *URL* donde se indica como parámetro el Punto de Interés que se tiene en frente el usuario.

Supongamos que ingresamos la *URL* que se corresponde con el *Museo Dardo Rocha* (simulando que el usuario leyó un código de barra) el usuario visualiza la pantalla que se muestra en la Figura 8.30.a. Cuando decide visualizar la vista del mapa (Figura 8.30.b) puede ver un link caminable derivado a la *Catedral* de *La Plata*. Además,

puede observar que el *Museo Dardo Rocha* no tiene links caminables predefinidos.

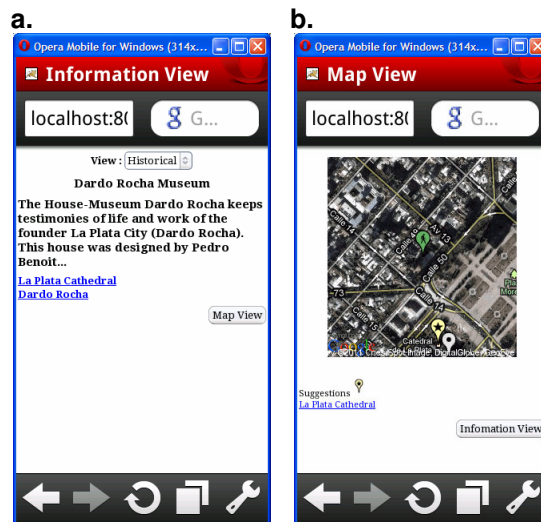


Figura 8.30: El usuario lee el código de barra del *Museo Dardo Rocha*.

El usuario vuelve a la vista de información (Figura 8.31.a⁹⁵) y decide navegar digitalmente a la *Catedral de La Plata* (Figura 8.31.b). Si pasa a la vista del mapa puede observar una pantalla como se muestra en la Figura 8.31.c. No se muestran links caminables predefinidos porque el *Museo Dardo Rocha* no tiene ninguno definido. Se puede observar que a partir de la información digital de la *Catedral de La Plata*, se derivó un link caminable al *Museo de La Catedral*.

El usuario puede decidir navegar en el mundo real, en este caso, la aplicación se comporta de la misma manera que para el ejemplo presentado en la Figura 8.28. Le calcula el camino y se lo muestra en el mapa.

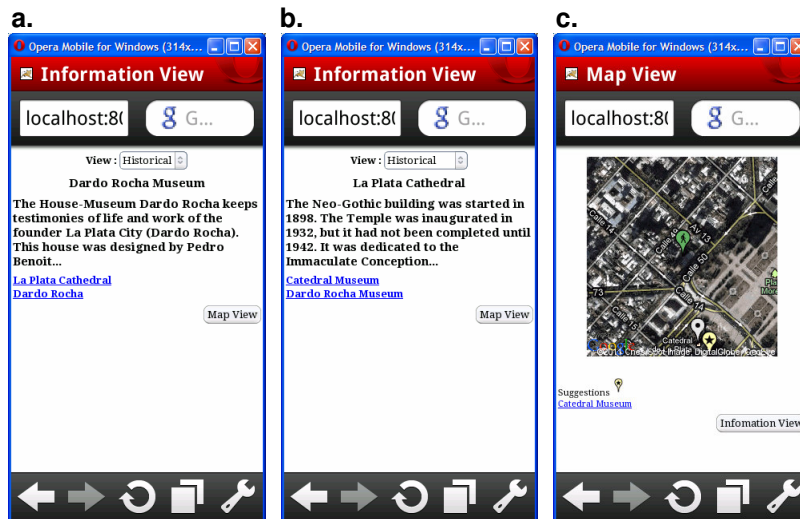


Figura 8.31: El usuario navega digitalmente a la *Catedral de La Plata*.

Veamos ahora un ejemplo distinto al del turista (tanto de *Roma* como de *La Plata*). En la Figura 8.32 se puede apreciar un Modelo de Instancias Navegacional⁹⁶ que se

⁹⁵ Es la misma que la Figura 7.30.a se repitió para que sea más fácil seguir el ejemplo.

⁹⁶ Este modelo no tiene definido caracteres especiales (o acentos) en los valores de las variables, ya que sino cuando se quieren visualizar en con el Browser *Opera Mobile* tira un error porque no los soporta.

corresponde⁹⁷ con el Modelo Navegacional de la obra de teatro presentado en la Sección 7.2.

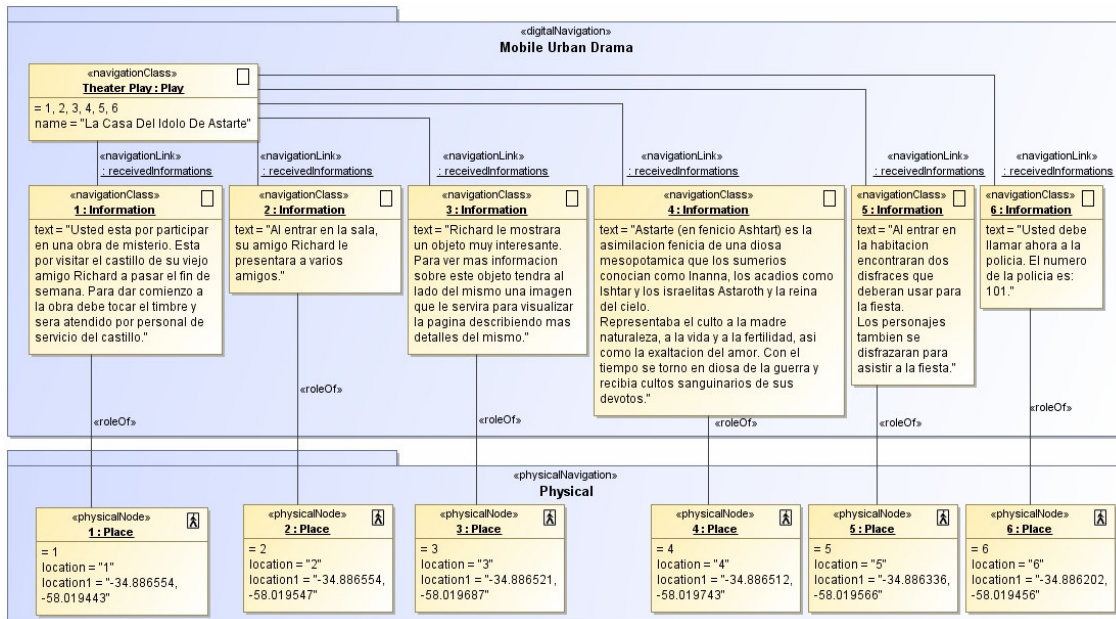


Figura 8.32: Un Modelo de Instancias Navegacional del ejemplo de la obra de teatro.

A partir del Modelo de Instancias Navegacional presentado en la Figura 8.32, el diseñador crea un Modelo del Usuario, al cual le definió como nombre de aplicación "Theather Application". Supongamos que el diseñador selecciona la opción de crear una aplicación Web Móvil a partir de este Modelo del Usuario. Como resultado de la creación se obtiene un archivo llamado *TheatherApplication.war*⁹⁸ que contiene el archivo *NavigationModel.xml* con la información del Modelo de Instancias Navegacional de la Figura 8.32.

El archivo *TheatherApplication.war* se copia en la carpeta *webapps* del *Tomcat* y cuando se enciende (o se reinicia) este servidor la aplicación queda funcionando para que el usuario la acceda mediante un Browser Web desde su dispositivo.

En la Figura 8.33, se pueden apreciar algunas pantallas que reflejan la información que recibe el usuario cuando usa la aplicación de la obra de teatro.

Supongamos que se el usuario lee el código de barra que se corresponde con el inicio de la obra (para simular esta situación se ingresa en el Browser una URL), el usuario pasa a visualizar la pantalla que se muestra en la Figura 8.23.a⁹⁹.

Al pasar a la vista del mapa¹⁰⁰ (Figura 8.23.b) sólo se indica la posición del usuario ya que no hay ni links caminables predefinidos ni derivados. Esto se debe a que en el Modelo de Instancias Navegacional, no hay links caminables predefinidos ni links digitales para derivar links caminables.

⁹⁷ Se modificó el Modelo Navegacional agregando una propiedad de ubicación, definida como geométrica.

⁹⁸ Se modificó el archivo *GUIInformation.xml* para que muestre el mapa como solamente satelital. Además, se modificó el archivo *NavigationModel.xml*, renombrando la propiedad *text* (derivada del modelo de instancias) por *description*, que es el nombre con que la aplicación recupera ese dato.

⁹⁹ Se puede observar que el usuario no tiene disponible la opción de cambiar de concerns, esto se debe a que la *JSP InFrontOfDigitalView* fue modificada por el diseñador para que no liste el nombre del concerns y así ajustar la aplicación a sus características particulares.

¹⁰⁰ Para este ejemplo se eligió una vista de mapa usando de *Google Static Maps* pero también el diseñador puede crear un modelo interno del lugar.

Luego el usuario lee el código de barra que se corresponde con el segundo acto de la obra y recibe la información que se visualiza en la Figura 8.33.c. Si pasa a visualizar la vista del mapa (Figura 8.33.d) puede apreciar que cambio su posición.

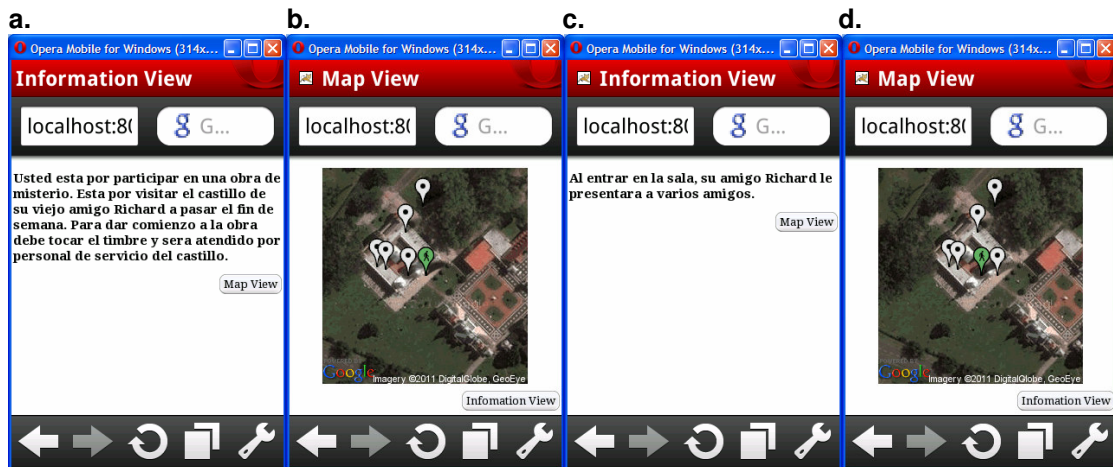


Figura 8.33: Aplicación de la obra de teatro funcionando usando Opera Mobile.

Se puede observar en estos tres ejemplos cómo a partir de tres Modelos de Instancias Navegacional diferentes, la herramienta puede derivar tres aplicaciones *Web Móvil*. Si bien estas aplicaciones tienen en común el mismo funcionamiento cada una muestra la información correspondiente a su Modelo de Instancias Navegacional. Se pudo ver que todas tienen en común: mostrar la información de un Punto de Interés, cambiar de concerns, navegar digitalmente, visualizar en el mapa la posición del usuario, navegar en el mundo real (tanto los links caminables predefinidos como derivados). Toda la funcionalidad común a ambos ejemplos viene dada por el archivo *MobileHypermediaApplication.war*, el cual se usa en las derivaciones de este tipo.

8.5. Resumen

En este capítulo se mostró el funcionamiento de la herramienta *MagicMobileUWE* y cómo se pueden generar dos tipos de aplicaciones de HM a partir de los modelos del enfoque. De esta manera, completamos nuestro desarrollo dirigido por modelo (para aplicaciones de HM).

Los dos tipos de aplicaciones que actualmente permite derivar la herramienta son: aplicaciones *J2ME* y aplicaciones *Web Móvil*. Se puede ver en la Figura 8.34 un esquema general que muestra las distintas derivaciones que se realizaron en este capítulo.

Se puede apreciar en la Figura 8.34 las dos derivaciones de las aplicaciones *J2ME* para lograr dos aplicaciones turísticas una de *Roma* y otra de *La Plata*. Estas derivaciones generan, a partir del Modelo de Instancias Navegacional, un archivo llamado *NavigationModel.xml* y luego usan los archivos *MobileHypermediaApplication.jad* y *MobileHypermediaApplication.jar* para generar la aplicación.

En la Figura 8.34 también se pueden ver las tres derivaciones de las aplicaciones *Web Móvil* realizadas, una para *Roma*, otra para *La Plata* y otra para la obra de teatro. Estas derivaciones generan, a partir del Modelo de Instancias Navegacional, un archivo llamado *NavigationModel.xml* y usando el archivo *MobileHypermediaApplication.war* se genera la aplicación.

Las dos tipos de derivaciones (*J2ME* y *Web Móvil*) tienen en común la generación del archivo *NavigationModel.xml* a partir de un Modelo de Instancias Navegacional. Pero cada una tiene una aplicación base diferente, una derivación usa los archivos

MobileHypermediaApplication.jad y *MobileHypermediaApplication.jar* (que contienen la aplicación *J2ME* básica) mientras que la otra derivación usa el archivo *MobileHypermediaApplication.war* (que contiene la aplicación *Web Móvil* básica). Ambas derivaciones generan aplicaciones que tienen las características básicas de las aplicaciones de HM.

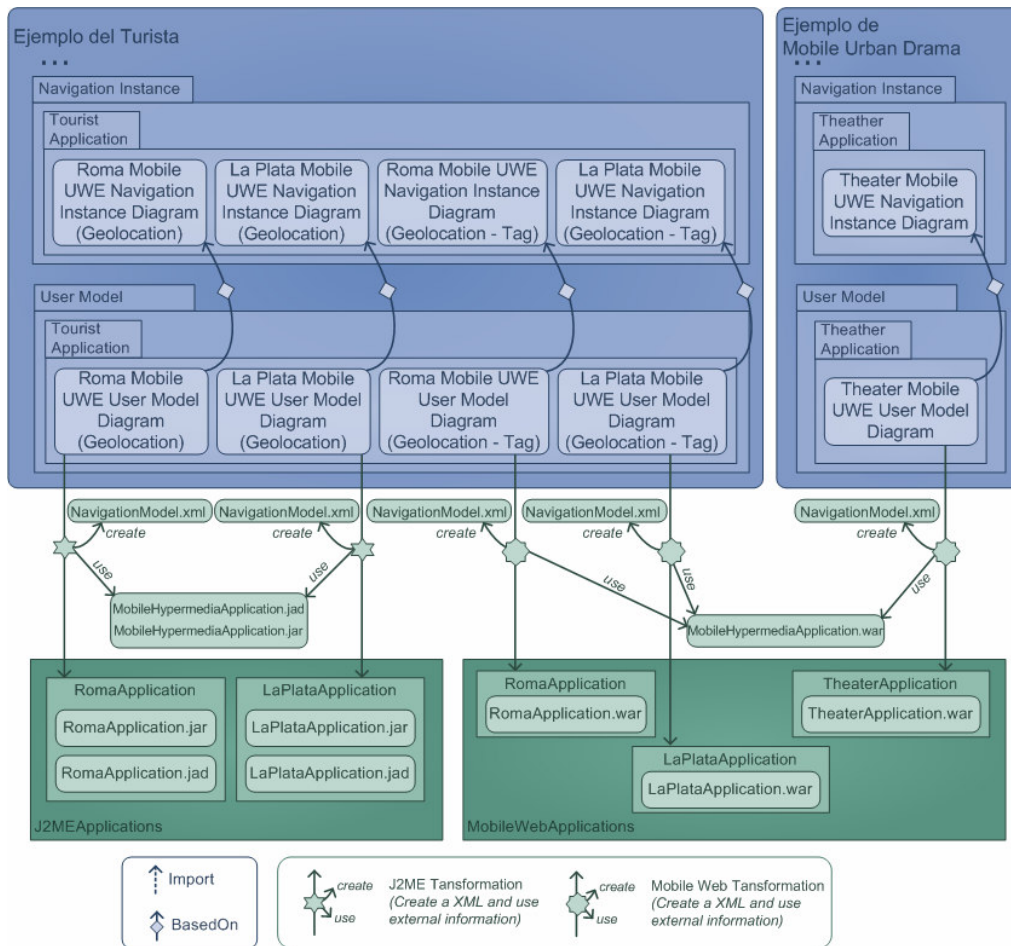


Figura 8.34: Esquema general de las aplicaciones generadas.

De esta manera, queda presentada la herramienta *MagicMobileUWE* y cómo llevar a cabo un desarrollo dirigido por modelos para obtener como resultado una aplicación de HM (ya sea *J2ME* o *Web Móvil*). Mostramos además que se pueden crear aplicaciones para dominios diferentes como son las aplicaciones turísticas y la obra de teatro.

9. TRABAJOS RELACIONADOS

En este capítulo se presentan distintos trabajos relacionados con el trabajo presentado. En la Sección 9.1 se detallan trabajos relacionados desde el punto de vista de las aplicaciones que permite derivar nuestro enfoque, especificando sus características principales y comparándolos con las aplicaciones (de HM) que permite modelar nuestro enfoque. En la Sección 9.2, se presentan las principales características de dos trabajos relacionados con nuestro enfoque y se realiza una comparación con los mismos.

9.1. Trabajos relacionados con las aplicaciones de Hipermedia Móvil

En los dos últimos años gracias al avance de las características de los nuevos teléfonos celulares, se dió comienzo a nuevas aplicaciones de realidad aumentada. Una de las más nuevas para celulares es *Layar*¹⁰¹, que ofrece diversas capas de contenido digital superpuesta a la imagen real que captura la cámara, esto se realiza usando la brújula y el *GPS* del celular. Mediante el *GPS* y la brújula se determina que información se debe mostrar superpuesta a la imagen real que captura la cámara. Gracias al uso de la brújula se logra mayor precisión de la información mostrada, dado que con el *GPS* sólo se logra determinar una ubicación pero no se puede determinar el lugar para dónde está apuntando el celular.

En la Figura 9.1 se puede apreciar un ejemplo de la aplicación *Layar*, donde se muestra la imagen real aumentada con información del lugar. En la pantalla se indica con una grilla los objetos que visualiza el usuario según la profundidad (distancia) a la que se encuentran. Además se muestra (abajo a la derecha), un círculo que representa la brújula del celular, indicando el ángulo de visualización que tiene el usuario en ese momento. Para este ejemplo particular, se pueden observar tres puntos (en el ángulo de la brújula), los cuales representan los tres círculos destacados en la imagen. Estos círculos indican los objetos de interés detectados en el ángulo de visión del usuario. Estos se encuentran a distintas distancias respecto del usuario y esto se refleja acorde al tamaño en que se dibuja cada círculo.



Figura 9.1: Imagen de la aplicación *Layar*.

¹⁰¹ Página de *Layar*: <http://www.layar.com>

La Figura 9.2 muestra las distintas vistas que puede ver el usuario cuando utiliza *Layar*. El usuario puede ver una imagen real de lo que está visualizando (a través de la cámara del celular), aumentada con la información de los objetos de interés que tiene en su rango de visión. Puede ver además, un mapa que muestra la ubicación de los objetos de interés cercanos, o puede ver la información de estos objetos de interés como una lista, junto con una breve descripción de los mismos.

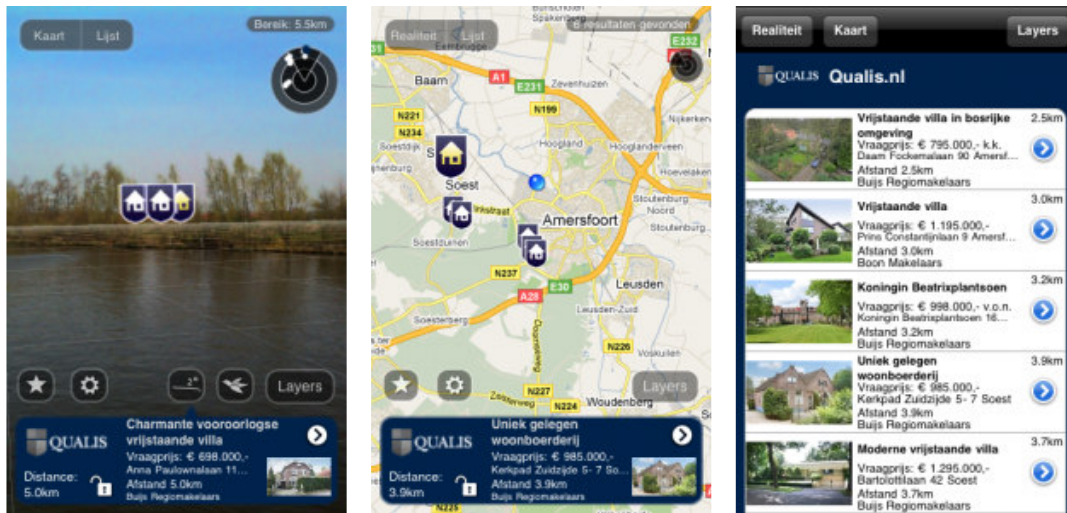


Figura 9.2: Distintas vistas de la información provista por *Layar*.

Existen varias capas de información que provee *Layar*, algunas de éstas están acotadas a determinadas ciudades o regiones. Las capas de información abarcan diferentes temas como pueden ser: información de transporte, universidades, comercios, centros asistenciales, redes sociales, recreación, etc. A medida que se fue incrementando el uso de esta aplicación fue creciendo la cantidad y variación de información que provee.

Los desarrolladores pueden crear nuevas capas y cargar información para luego publicarlas en el sitio de *Layar*, permitiendo que otros usuarios las puedan usar. La creación de las capas se realiza mediante el uso de herramientas provistas por *Layar*, utilizando un mapa donde se indica las posiciones de los lugares cargando la información correspondiente a cada lugar.

Layar trabaja integrado con *Google Maps*¹⁰² para proveer mapas al usuario, y además brindarle búsqueda de caminos a los lugares que se visualizan en pantalla.

Actualmente *Layar* corre en celulares *Android* e *iPhone 3GS*. Se encuentra en etapa de desarrollo para celulares que tienen sistema operativo *Symbian*. El celular debe tener conexión a Internet, cámara, *GPS* y brújula para que la aplicación *Layar* pueda correr.

Actualmente existe otra empresa llamada *Acrossair*¹⁰³ que se dedica a realizar aplicaciones de realidad aumentada para *iPhone*. Una de las aplicaciones que tiene desarrollada esta empresa es sobre el metro de París¹⁰⁴. Esta aplicación utiliza la cámara del teléfono, su *GPS*, la brújula y el acelerómetro para brindar la información correspondiente al usuario. Esta aplicación funciona de forma muy similar a la aplicación *Layar* descrita anteriormente, pero cuenta con el adicional de utilizar el acelerómetro para brindar información diferente, dependiendo de la inclinación del celular. Es decir, el acelerómetro puede detectar si el usuario está apuntando hacia el piso o no, dependiendo de esta orientación, la aplicación se adapta para mostrar

¹⁰² Página de *Google Maps* para celulares: <http://www.google.es/mobile/maps/>

¹⁰³ Página de *Acrossair*: <http://www.acrossair.com>

¹⁰⁴ Página de la aplicación del metro de París: http://www.metroparisiphone.com/index_en.html

información diferente. Si el usuario apunta el celular hacia el piso se muestra información relacionada al mapa mientras que si apunta hacia el frente ve la información de realidad aumentada.

En la Figura 9.3 se puede apreciar la imagen aumentada de la aplicación del metro de Paris que recibe el usuario, en la cual se muestran información de los distintos lugares de interés cercanos y la distancia que hay hasta estos. En este caso el usuario está apuntando su celular hacia el frente.

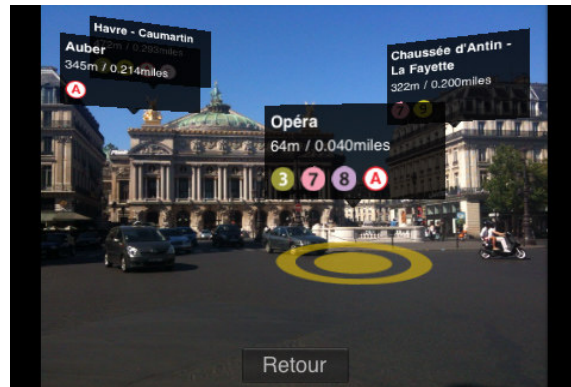


Figura 9.3: Aplicación del metro de Paris.

En esta aplicación, el usuario tiene la posibilidad de elegir caminar a uno de los lugares que se le muestra en pantalla, al elegir uno de los lugares el sistema lo guía para llegar a su destino.

En la Figura 9.4.a se puede apreciar como el usuario visualiza en forma destacada la opción elegida (*Chaussée d'Antin – La Fayette*). Como parte de la información se muestra qué línea de metro se puede tomar para llegar más rápido al lugar indicado. Si el usuario apunta el celular hacia el piso, el sistema detecta esto mediante el acelerómetro del celular, y le muestra al usuario una flecha indicando la dirección del lugar elegido, esta acción puede visualizarse en la Figura 9.4.b. Cuando el usuario mueve su celular, el sistema toma la información de la brújula del mismo y adapta la flecha según el movimiento que realizó el usuario. A medida que el usuario camina hacia el lugar, el sistema le va actualizando los datos mostrados y la distancia que le falta recorrer para llegar al mismo.

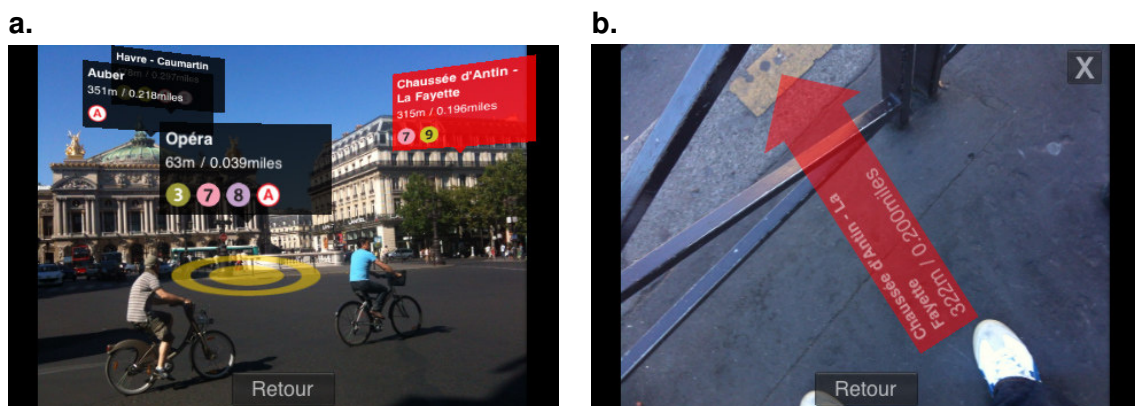


Figura 9.4: El usuario elige un destino en la aplicación del metro de Paris.

Esta aplicación además, cuenta con la opción de brindar información de los negocios disponibles en la ciudad. En la Figura 9.5 se puede apreciar cómo el usuario recibe información de los lugares que lo rodean y a su vez conoce la distancia a los mismos. Se puede ver que estos lugares son mostrados al usuario con sus logos característicos, permitiendo la identificación de los mismos de forma inmediata. El usuario puede elegir ir a cualquiera de estos lugares y el sistema lo guiará para que

pueda llegar al destino seleccionado.

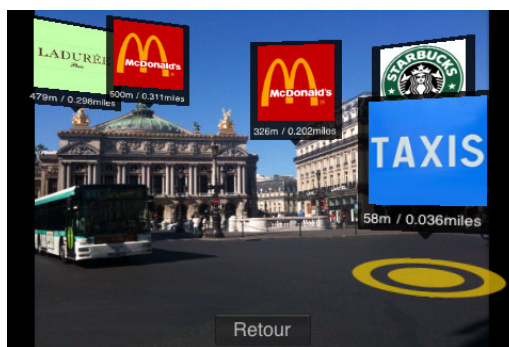


Figura 9.5: Información de los negocios en la aplicación del metro de Paris.

Este sistema también usa *Google Maps* para proveerle al usuario toda la funcionalidad relacionada a los mapas, indicando sobre el mismo, los lugares que el usuario visualiza en la aplicación según donde está ubicado (información del *GPS*).

Retomemos las dos aplicaciones ya introducidas en el Capítulo 2, ambas enmarcadas dentro del concepto *Mobile Urban Drama*. Una de estas aplicaciones denominadas por [Hansen et al., 2008] como *AudioMove*, en donde el usuario se convierte en el actor principal de una obra y se usa el mundo real como escenario. El usuario recibe una historia inicial y luego debe moverse por los distintos lugares para conocer cómo transcurre la obra. En esta aplicación, el usuario puede recibir distintas pistas para poder descubrir cuál es el próximo lugar a donde se debe dirigir. Mientras va caminando de un lugar a otro, va escuchando diferentes audios acordes a la parte de la obra que va transcurriendo. Estos audios tienen una duración estratégica para que acompañen al usuario cuando se mueve de un lugar a otro, de esta manera se logra que el usuario mantenga la atención en la obra. Los audios fueron grabados previamente por actores del teatro *Katapult*¹⁰⁵ de la ciudad de *Aarhus (Dinamarca)*.

En los diferentes lugares (donde transcurre la obra) el usuario tiene disponibles códigos de barra que puede leer con su dispositivo, al leerlos, recibe los audios de la historia como se puede visualizar en la Figura 9.6. También puede recibir información de actividades que debe realizar, como así también, llamadas a su celular para recibir indicaciones. Los celulares usados en esta aplicación deben contar con cámara para leer los códigos de barra y el usuario debe contar con auriculares para escuchar los audios.

Las escenas transcurren de manera lineal y están preestablecidas en la aplicación, pero sólo las acciones del usuario hacen que la obra avance. Es decir, a medida que el usuario va leyendo cada uno de los códigos de barra y se mueve de un lugar a otro genera que la obra avance.

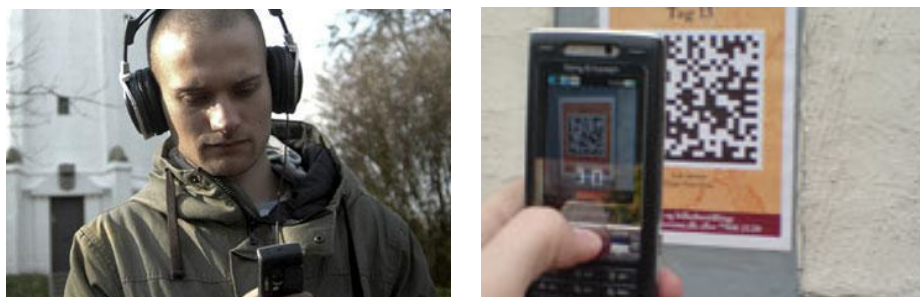


Figura 9.6: Aplicación AudioMove.

¹⁰⁵ Página del teatro *Katapult*: <http://www.katapult.dk>

La otra aplicación de *Mobile Urban Drama* mencionada en el Capítulo 2, es una aplicación educativa llamada *HasleInteractive* [Hansen and Bouvin, 2009]. En [Hansen et al., 2010] los autores describen más en profundidad esta aplicación.

Los alumnos van leyendo los códigos de barras y recibiendo, tanto información como actividades que deben realizar, por ejemplo, tomar fotos o grabar un video de la zona donde se encuentran ubicados describiendo lo que observan, como se visualiza en la Figura 9.7. A medida que se mueven de un lugar a otro, van recibiendo audios que los ayudan a concentrarse en la historia y no perder el foco de atención. Estos audios fueron realizados por los actores del teatro de *Katapult*.

Al finalizar el juego, los alumnos deben crear una presentación con las fotos y los videos que realizaron, describiendo la experiencia y el análisis que pudieron realizar en cada lugar visitado. Este material es analizado en clase junto con el docente y el resto de los alumnos.

Esta aplicación (*HasleInteractive*) está orientada a aplicaciones de biología, matemática y geografía, en donde se plantea una historia de base que se usa para proponer un conjunto de tareas que deben realizar los alumnos. Una de las historias envía a los alumnos al año 2022 y se les cuenta que el planeta está infectado por un virus muy peligroso que se va expandiendo por toda la tierra. Los alumnos deben analizar un área en particular que todavía no fue afectada por este virus e investigar cada lugar según lo que le vaya indicando el sistema. Simultáneamente, pueden jugar cuatro grupos, cada grupo puede tener entre dos a cuatro alumnos. Si bien los lugares a recorrer son los mismos, cada grupo debe analizar características distintas del mismo. En cada lugar, todos los integrantes del grupo deben leer el código de barra para recibir la tarea a realizar. El sistema maneja secciones para mantener sincronizada la información del grupo, esta información es persistida, para poder ser recuperada cuando alguno de los miembros del grupo lo necesite. Los grupos inician todos en un determinado lugar, donde reciben la historia y la consigna que deben resolver, y desde ahí se los guía para cuatro lugares diferentes así no hay superposición de grupos en cada lugar. El grafo de cada una de las cuatro historias está preestablecido.

La aplicación está desarrollada en *Java Micro Edition* para celulares *Sony Ericsson W910* y *W715*.



Figura 9.7: Aplicación *HasleInteractive*.

Comparemos ahora cada una de las cuatro aplicaciones presentadas con las aplicaciones que permite modelar nuestro enfoque. Hay que considerar que estas aplicaciones están creadas para fines específicos. Analizaremos el funcionamiento de las dos aplicaciones turísticas *Layar* y *Acrossair* comparándolas con nuestros ejemplos del turista, sólo nos focalizaremos en la funcionalidad que proveen ya que ambas son de código cerrado. Compararemos también la aplicación de *AudioMove* con nuestro ejemplo de la obra de teatro como así también compararemos la aplicación *HasleInteractive* con nuestro juego educativo.

La principal diferencia que tenemos respecto de las aplicaciones *Layar* y *Acrossair*, es

que éstas están focalizadas en brindar realidad aumentada. Por lo tanto, el usuario ve la imagen real con información superpuesta. Las aplicaciones que podemos modelar con nuestro enfoque sólo brindan información del Punto de Interés que senso al usuario. En el caso de querer proveer la imagen real superpuesta, el diseñador tiene que adaptar la derivación del código que genera nuestro plugin (*MagicMobileUWE*) para poder procesar la imagen real y superponerle la información.

Las aplicaciones de *Layar* y *Acrossair* hacen uso del acelerómetro y brújula, para brindar información más precisa ya que por las características de la aplicación con la información del *GPS* no es suficiente. Nuestras aplicaciones no consideran la tecnología del acelerómetro como parte del modelado, ya que las aplicaciones que modelamos se focalizan en mostrar información de un Punto de Interés en particular sin considerar la orientación que tenga el dispositivo.

Las aplicaciones que podemos modelar con nuestro enfoque, no solamente pueden usar posicionamiento geométrico sino también posicionamiento a través de etiquetas o simbólico. Esto permite que las aplicaciones resultantes puedan ser ejecutadas en celulares que no cuentan con *GPS*. *Layar* y *Acrossair* no brindan el sistema de lectura de códigos de barras como método de posicionamiento.

Layar cuenta con la característica de; modelar el espacio como una grilla, marcando con un punto cada objeto de interés, al ser marcado el punto en la grilla se puede percibir la distancia que hay hasta estos. *Acrossair* dibuja los objetos de interés con un logo o un recuadro superpuestos en la imagen, el tamaño del dibujo varia dependiendo de la distancia que haya a los mismos. Mostrar la profundidad de ubicación de los Puntos de Interés, no es una característica modelada en nuestras aplicaciones ya que no nos focalizamos en brindar realidad aumentada, a lo sumo el usuario puede ver en el mapa los lugares a los que puede ir caminando pero no percibe la profundidad de los mismos.

Tanto *Layar* como *Acrossair*, brindan la realización de caminos asistidos mediante mapas, ambos usan la búsqueda de caminos de *Google Maps* como así también sus mapas. En nuestros ejemplos (Sección 8), se mostró cómo las aplicaciones derivadas por la herramienta usan la *API* de *Static Google Maps* para brindar mapas y la *API* de *Google Directions* para buscar caminos.

Comparemos ahora la aplicación de *AudioMove* descrita anteriormente y las aplicaciones de teatro que derivamos con nuestro enfoque. Una diferencia es que nuestras aplicaciones no cuentan con una historia prefijada sino que involucran actores reales que improvisan acorde a la actitud y reacción que tenga el usuario. Es decir, la historia se va formando dinámicamente y puede variar cada vez que se juega. En *AudioMove* sólo participa un usuario en la historia, en nuestras aplicaciones puede haber más de un usuario al mismo tiempo en la obra y cada uno puede tener roles diferentes dentro de la misma.

Nuestra aplicación de teatros puede ser representada como grafos, donde el usuario debe elegir que camino seguir cuando se le presenta una bifurcación. En cambio la aplicación *AudioMove* establece una secuencia lineal.

En ambas aplicaciones (*AudioMove* y nuestras aplicaciones de teatro) el usuario se debe mover de un lugar al otro para que la historia transcurra. Tanto en nuestro ejemplo, como en *AudioMove*, el usuario debe leer un código de barra para poder recibir información e ir avanzando en la obra.

La información que recibe el usuario en nuestra aplicación de teatro es textual en su dispositivo o interactuando con los actores reales, contrario a *AudioMove* que brinda la información mediante audio. Recibir la información textual, trae como problemática que el usuario debe detenerse a leer la información. Esto puede traer aparejado perder la interacción con los actores reales, los cuales tienen que esperar que el usuario termine de leer para poder interactuar y continuar con la obra. Escuchar un audio es más ágil que leer la información, pero trae como problemática que el usuario queda totalmente aislado de la realidad y no escucha los ruidos que hay a su alrededor, los cuales en

algunas obras son de gran importancia.

Por último comparemos la aplicación *HasleInteractive* con nuestro juego educativo. Nuestro juego educativo está pensado para que el alumno sea evaluado mientras lo realiza, es decir, las preguntas que contiene el juego permiten saber el grado de conocimiento del alumno. *HasleInteractive* está orientado a la recolección y análisis de situaciones, para luego ser analizadas en clase junto al docente. Es decir, *HasleInteractive* está pensado para que el alumno genere fotos y videos permitiendo que los almacene en su dispositivo para luego ser utilizados.

Nuestro juego puede plantear preguntas que sean evaluadas por un docente en tiempo real, permitiendo una evaluación más compleja de cada pregunta ya que depende de cómo responde el alumno es el puntaje que se le asigna.

Tanto en *HasleInteractive* como en nuestro juego educativo, se representa un grafo de lugares que el alumno debe recorrer. *HasleInteractive* está pensado para grupos de alumnos mientras que, por ahora, nuestro enfoque sólo deriva juegos individuales, pero el diseñador puede adaptar el código generado para que sea una aplicación grupal.

Tanto las historias presentadas por *HasleInteractive*, como nuestros juegos educativos pueden ser variadas, como así también la temática de las preguntas o actividades que se pueden realizar. Ambos (*HasleInteractive* como nuestros juegos educativos) necesitan de los docentes calificados, para la creación del material que contenga el juego acorde al grupo de alumnos que lo jueguen.

Nuestros juegos educativos son más generales y no están restringidos a determinados teléfonos, ya que en el caso de hacer una derivación a una aplicación Web no hay restricciones en el celular mientras el mismo tenga instando un Browser Web.

Nuestro enfoque permite crear, mediante la transformación de modelos, diferentes tipos de aplicaciones educativas simplemente variando el Modelo de Instancias Navegacional de las mismas.

De esta manera quedan planteadas las diferencias y similitudes entre las cuatro aplicaciones presentadas (*Layar*, *Acrossair*, *AudioMove* y *HasleInteractive*) y los ejemplos que se pueden generar con nuestro enfoque.

9.2. Trabajos relacionados con el enfoque propuesto

Para comparar nuestro enfoque con otras metodologías relacionadas, primero veamos las características particulares del trabajo presentado en esta tesis para luego poder realizar una comparación con otras metodologías.

Como se ha mencionado en el Capítulo 4, *UWE* es una metodología para modelar aplicaciones de Hipermedia tradicional; se define como una extensión conservativa de *UML*, es decir los elementos del metamodelo de *UML* no se modifican sino que definen un perfil con los elementos propios de *UWE*. Como se mencionó en el Capítulo 6, actualmente *UWE* provee una herramienta denominada *MagicUWE* (plugin para *MagicDraw*) que permite el desarrollo dirigido por modelo para aplicaciones de Hipermedia.

Como ya mencionamos en el Capítulo 4, nuestro enfoque es una extensión conservativa de *UWE*, para lograr esto definimos un perfil con los elementos necesarios para modelar aplicaciones de HM. Principalmente *UWE* se usó como base del Modelo de Contenido y el Modelo Navegacional, como así también la forma en que *UWE* realiza sus transformaciones.

La necesidad de incorporar los nuevos elementos para modelar las características

propias de HM, nos llevó a crear nuestros propios modelos (Capítulo 5). El enfoque presentado en esta tesis cuenta con seis modelos: el Modelo de Contenido, el Modelo Unificado, el Modelo Navegacional, el Modelo de Instancias Navegacional, el Modelo de Presentación y el Modelo del Usuario. Todos estos elementos son diagramas de clases *UML*, los cuales contienen elementos que usan los estereotipos definidos en nuestro perfil. Sólo en nuestro Modelo Navegacional se usa para la parte del modelado Hipermedial (tradicional), es decir el concern digital, los elementos definidos por *UWE* para su Modelo Navegacional. De esta manera se aprovecha las definiciones de los elementos Hipermediales ya definidas por el perfil de *UWE*.

Las características principales de nuestros modelos y los elementos que se definieron en nuestro perfil son:

- **Modelo de Contenido:** Este modelo permite modelar los conceptos relevantes del dominio de la aplicación. Para este modelo se definieron, en nuestro perfil, dos estereotipos para paquetes, `<<digitalContent>>` y `<<physicalContent>>` para representar los concerns digitales y el concern físico respectivamente. Además, se crearon tres estereotipos para poder especificar la propiedad de ubicación: `<<tagLocation>>`, `<<geometricLocation>>` y `<<symbolicLocation>>`.
- **Modelo Unificado:** Mediante este modelo se logra unificar el Modelo de Contenido, definiendo cuál de todos los concerns digitales es el concern *Core*. En nuestro perfil, se definió el estereotipo de asociación `<<roleOf>>` para representar la relación de role entre las clases del concern *Core* y las clases de los otros concerns.
- **Modelo Navegacional:** Permite el modelado de la HM de nuestra aplicación. En nuestro perfil, se definieron los estereotipos (`<<digitalNavigation>>` y `<<physicalNavigation>>`) para representar el concern navegacional digital y el concern navegacional físico. Para el concern físico se definieron los estereotipos `<<physicalNode>>` y `<<walkingLink>>` para representar el nodo físico y el link caminable respectivamente. Además, se definieron en nuestro perfil dos estereotipos de operación para representar la función de links caminables derivados como son `<<allHavePhysicalCounterparts>>` y `<<onlyNearPhysicalCounterparts>>`. En los concerns digitales, se utilizan los elementos definidos por el perfil de *UWE* para su Modelo Navegacional.
- **Modelo de Instancias Navegacional:** Representa la instanciación de un Modelo Navegacional en particular (usado como modelo de base) y no se agrega ningún estereotipo nuevo en nuestro perfil para este modelo.
- **Modelo de Presentación:** Permite la asignación de una visualización particular para un determinado Modelo Navegacional. Se definió, para la implementación particular mostrada, un perfil con dos estereotipos para representar los concerns de presentación digital y el concern de presentación físico (`<<digitalLayout>>` y `<<physicalLayout>>`). Además, se definieron en este perfil los estereotipos `<<navigationClassLayout>>` y `<<physicalNodeLayout>>` para representar las clases de presentación de los nodos digitales y físicos.
- **Modelo del Usuario:** Este modelo se define las clases mínimas para proveer el funcionamiento de aplicación de HM. En particular, se mostró una implementación de este modelo, para el cual se creó un estereotipo `<<mobileHypermediaApplication>>` para modelar la clase que representa la aplicación de HM. Esta clase conoce con un Modelo de Instancias Navegacional particular con su correspondiente presentación

asociada.

Para nuestro desarrollo dirigido por modelo se definen dos tipos de transformaciones diferentes:

- Transformación del Modelo de Contenido al Modelo Unificado: definimos dos transformaciones diferentes una que permite elegir cuál es el concern *Core* del Modelo de Contenido para derivar el Modelo Unificado. Considerando en la transformación todos los concerns definidos en el Modelo de Contenido. La otra transformación permite elegir cuál es el concern *Core* del Modelo de Contenido y, además, cuáles son los concerns de interés para realizar la transformación, es decir filtrar (en la transformación) los concerns que no son de interés. Según el concern elegido como *Core* agrega la relación de rol según corresponda.
- Transformación del Modelo Unificado al Modelo Navegacional: Crea un Modelo Navegacional a partir del Modelo Unificado respetando el concern *Core* del mismo. El modelo generado queda expresado en términos de nodos (digitales y físicos) y links (digitales y caminables).

Para el enfoque presentado en esta tesis, se provee una herramienta que permite la creación de nuestros modelos, como así también la realización de las transformaciones mencionadas anteriormente. La herramienta, como ya se mencionó en el Capítulo 6, es una extensión para *MagicDraw* llamado *MagicMobileUWE*. Esta elección se debe a que *UWE* provee un plugin para *MagicDraw* llamado *MagicUWE* y, de esta manera, podemos rehusar los elementos del Modelo Navegacional definidos por el plugin de *UWE* en nuestro Modelo Navegacional, ya que nuestra extensión es compatible.

La herramienta define diferentes menús contextuales para facilitar al diseñador la definición de los elementos de cada uno de nuestros modelos. Además, se permite que el diseñador realice la transformación, y luego ajuste el resultado de la misma a las necesidades de la aplicación que está modelando.

Además, permite la derivación de un Modelo del Usuario a aplicaciones, como se especificó en el Capítulo 8, por ahora se pueden hacer derivaciones de aplicación *J2ME* (*Java Micro Edition*) que corren directamente en el celular o aplicaciones Web Móvil, que corren en un servidor y son accedidas desde el celular mediante un Browser Web. Las aplicaciones *J2ME* derivadas con nuestro plugin cuentan con una librería de ubicación, que permite leer los datos generados por el *GPS* del celular, mientras que las aplicaciones Web Móvil generadas, ubican al usuario mediante el valor de la etiqueta pasada como parámetro.

Ahora se detallan dos metodologías de desarrollo dirigido por modelos y luego se comparan las mismas con nuestro enfoque.

Una de las metodologías se denomina *CAUSE* (*Context-Aware Applications for Ubiquitous Computing Enviroments*), está descripta en detalle en [Tesoriero, 2009] y [Tesoriero et al., 2010], es un desarrollo dirigido por modelos para aplicaciones sensibles al contexto en ambientes ubicuos. Esta metodología está dividida en tres capas:

- Capa de Análisis (*Analysis Layer*): Esta capa sirve para representar las características de las aplicaciones sensibles al contexto, para esto la metodología define tres modelos dentro de esta capa:
 - *Social Model*, este modelo permite modelar las entidades del sistema y sus relaciones. Para representar estos conceptos definen instancias y luego extienden su funcionalidad en base a la relación de rol.

- *Space Model*, define el ambiente físico de las entidades del sistema, es decir modela las características de ubicación e infraestructura, como así también, las condiciones físicas de las entidades. Este modelo permite definir las posiciones de las entidades como así también las relaciones espaciales entre las mismas.

Define dos tipos de espacios: los físicos y los virtuales, los primeros representan a aquellas entidades que se pueden ubicar en una determinada posición mientras que los segundos representan entidades que no tienen una posición determinada sino que representan un concepto dentro del sistema.

- *Task Model*, define el conjunto de tareas de las distintas entidades del sistema, y en qué espacios se deben llevar a cabo. Además, este modelo define las pre y post condiciones necesarias para cada tarea, expresando de esta manera las restricciones de ejecución de las tareas. Este modelo está íntegramente relacionado al modelo espacial y social.

Para estos tres modelos, la metodología define sus propios metamodelos para definir cada uno de los elementos que se deben representar.

- Capa de Información (*Information Layer*): Esta capa permite al diseñador definir las características en cuanto a desarrollo, arquitectura y comunicación propias del software de aplicaciones sensibles al contexto. Para esta capa, la metodología define tres modelos: *Entity Context*, *Referencial Space* e *Information Flow*.
 - *Entity Context*, define la estructura interna de las entidades de contexto como así también sus relaciones. Este modelo representa situaciones junto a las acciones que se deben tomar.
 - *Referencial Space*, representa la estructura del sistema en término de las entidades definidas en el modelo social.
 - *Information Flow*, representa el intercambio de información entre las entidades, definiendo la estructura de la información y cómo ésta es transportada entre entidades.

La metodología define para estos tres modelos sus propios metamodelos para definir cada uno de los elementos que se deben representar.

- Mapeo a Código (*Mapping*): Esta capa define las relaciones entre los modelos de la capa de información con el correspondiente pasaje a código.

Esta metodología define dos transformaciones, una permite generar a partir de los tres modelos de la capa de análisis, los tres modelos de la capa de información. Esta transformación se define como una transformación multi-model ya que a partir de tres modelos se generan tres modelos nuevos. Es decir, no es una simple transformación de un modelo a otro sino que es una transformación compleja que involucra tres modelos de entrada y tres modelos de salida. Esta transformación se define en *ATL*¹⁰⁶ (*Atlas Transformation Language*). La otra transformación, se establece entre los tres modelos de la capa de información y los mapeos para generar el código resultante. Esta transformación se define en base a *MOFScript*¹⁰⁷ y es multi-model ya que involucra tres modelos de entrada junto a los mappings, generando como salida el código correspondiente. El código generado es representado con clases *Java*. Para esta metodología se desarrolló un plugin para *Eclipse* usando *EMF*¹⁰⁸ (*Eclipse*

¹⁰⁶ Página de *Eclipse* sobre *ATL*: <http://www.eclipse.org/atl>

¹⁰⁷ Página de *Eclipse* sobre *MOFScript*: <http://www.eclipse.org/gmt/mofscript>

¹⁰⁸ Página de *Eclipse* sobre *EMF*: <http://www.eclipse.org/modeling/emf>

Modelling Framework) y GMF¹⁰⁹ (Graphical Modelling Framework) para poder crear cada uno de los modelos de la metodología, como así también, realizar las dos transformaciones mencionadas.

Un esquema general de la metodología descrita puede visualizarse en la Figura 9.8, donde se muestran los diferentes modelos mencionados junto a las transformaciones asociadas a esta metodología.

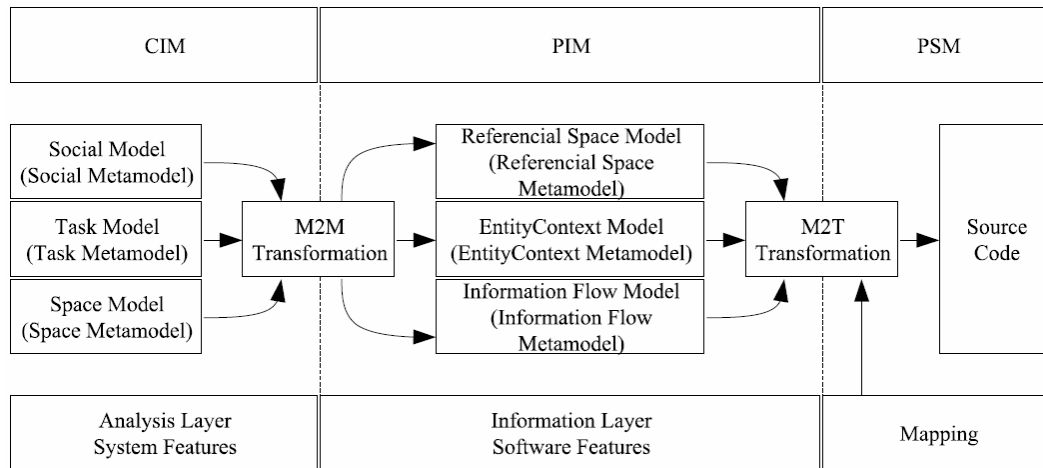


Figura 9.8: Esquema general de CAUSE [Tesoriero et al., 2010].

Comparemos ahora la metodología presentada (*CAUSE*) con nuestro enfoque. La principal diferencia entre nuestro enfoque y *CAUSE* es que nosotros nos focalizamos en aplicaciones de HM, y *CAUSE* en aplicaciones sensibles al contexto, es por esta razón que no tiene representado el modelo Hipermedial de la aplicación, es decir, no tiene los conceptos de nodos y links.

CAUSE tiene representado el flujo de la información con las pre y post condiciones, mientras que en nuestro enfoque esto no está establecido, ya que cada usuario puede recorrer libremente su instancia del Modelo Navegacional. *CAUSE* establece cuándo la información de contexto se debe mostrar, mientras que las aplicaciones modeladas con nuestro enfoque la información es visualizada porque el usuario cliqueo un link (digital o caminable) o porque fue sentido por un Punto de Interés. Es decir, en nuestras aplicaciones los mecanismos que actualizan la información que visualiza el usuario, son propios del funcionamiento de las aplicaciones de HM.

CAUSE define un metamodelo propio para definir los elementos de su metodología y no utiliza la definición de un perfil *UML* como lo hace nuestro enfoque. Para modelar y dibujar sus elemento *CAUSE* creó un plugin para *Eclipse*, el cual fue definido en base a dos frameworks de *Eclipse* (*EMF* y *GMF*). Nuestro enfoque, al tener definido un perfil de *UML*, genera modelos que pueden ser representados con cualquier herramienta de modelado *UML*. Los diagramas aparecen dibujados con las características básicas de los elementos de *UML*, y a lo sumo, el estereotipo definido puede agregar algún icono particular, como en nuestro caso, para el link caminable y el nodo físico. Nosotros proveemos una herramienta (*MagicMobileUWE*), la cual permite crear los diagramas usando *UML* y nuestros estereotipos.

Las transformaciones involucradas en *CAUSE* son complejas, ya que involucran considerar varios modelos para generar un resultado. Nuestras transformaciones, toman un único modelo de entrada y generar un único modelo de salida.

Otra metodología que se describe para compararla con nuestro enfoque, es un desarrollo dirigido por modelos para aplicaciones Web sensibles al contexto. Esta metodología es presentada en [Ceri et al., 2007] y descrita con más nivel de detalle

¹⁰⁹ Página de *Eclipse* sobre *GMF*: <http://www.eclipse.org/modeling/gmp>

en [Daniel, 2009] y [Daniel, 2010]. Este desarrollo extiende la metodología *WebML*¹¹⁰ (Web Modeling Language) [Ceri et al., 2003], la cual modela aplicaciones de Hipermedia. La extensión agrega los conceptos particulares para soportar cambios de la información de contexto. Es decir, usan los modelos definidos por *WebML* e incorporan los elementos necesarios para representar la información de contexto. Veamos cuáles son los conceptos básicos de *WebML*. Esta metodología (*WebML*) define tres modelos:

- Un modelo de datos que permite modelar el contenido de la aplicación, representándolo con un modelo de Entidad-Relación.
- Un modelo de Hipertexto expresado en términos de sitio, áreas, páginas y unidades de contenido (visualización de una o más entidades del modelo de datos). Las páginas y las unidades de contenido están conectadas mediante links.
- Un Modelo de Presentación, expresado en términos textuales mediante un *XML*¹¹¹ en el cual se detallan propiedades que no pueden ser expresadas mediante una notación gráfica. Este *XML* es el punto de entrada para la generación de código.

WebML genera sus modelos usando la herramienta *WebRatio*¹¹² [Brambilla et al., 2008], esta herramienta ofrece un editor para todos los modelos *WebML* permitiendo expresar visualmente todos elementos de sus modelos. Además, *WebRatio* provee derivación a código creando aplicaciones que cumplen con el estándar *Java/JSP 2.0* y éstas se pueden instalar en cualquier servidor Web.

La extensión de *WebML* para aplicaciones Web sensibles al contexto, representa en el modelo de datos las distintas características de contexto, por ejemplo, el usuario y su posición todo en término de entidades y relaciones. Además, en este modelo se pueden representar, por ejemplo, las áreas geográficas de la aplicación, los edificios, las calles, etc.

La extensión de *WebML* incorpora, al modelo de Hipertexto, una etiqueta *c* a las páginas que necesitan adaptarse, de esta manera, visualmente se puede identificar aquellas páginas que necesitan adaptación acorde al contexto. A estas páginas etiquetadas se las denomina *Page Context*, las cuales deben adaptarse según alguna característica de contexto, es decir si cambia la característica de contexto que muestra la página, se debe mostrar con el nuevo valor de la misma.

Las páginas que muestran información de contexto, son observadas mediante un *Context Monitor* del lado del cliente y otro desde el lado del servidor (*Context Monitor Client* y *Context Monitor Server* en la Figura 9.10). El *Context Monitor* del lado del cliente, es un mecanismo de monitoreo embebido en la página *HTML*¹¹³ que visualiza el usuario en su Browser. Mientras que el *Context Monitor* que está en el servidor, es el que tiene acceso a la información de contexto del usuario. Estos dos elementos trabajan en forma conjunta para mantener actualizadas las páginas que muestra información contextual. El *Context Monitor* del lado del cliente, periódicamente toma el valor por ejemplo del *GPS* y se comunica con el *Context Monitor* que está en el servidor, para que esté controle si hubo un cambio respecto de la información de contexto del usuario. En el caso que la respuesta sea afirmativa, el *Context Monitor* que está en el servidor guarda el nuevo valor del contexto y le envía al *Context Monitor* del lado del cliente la respuesta afirmativa. Cuando el *Context Monitor* del lado del cliente recibe una respuesta afirmativa, la página cuenta con un mecanismo para generar un requerimiento automático al servidor, y de esta manera, actualizar la

¹¹⁰ Página de *WebML*: <http://www.webml.org>

¹¹¹ Página de *XML*: <http://www.w3.org/XML>

¹¹² Página de *WebRatio*: <http://www.webratio.com/>

¹¹³ Página de especificación de *HTML4*: www.w3.org/TR/html4/

información contextual que está mostrando en la página.

En la Figura 9.9 se pueden visualizar los elementos representados en esta arquitectura (que permite el desarrollo dirigido por modelos para aplicaciones Web sensibles al contexto) tanto en el cliente como del servidor.

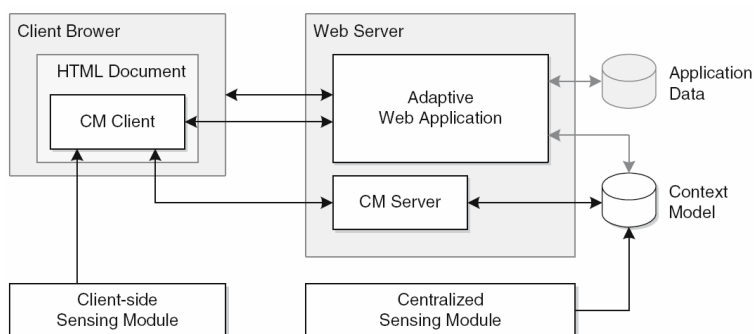


Figura 9.10: Arquitectura de la aplicación Web sensible al contexto [Daniel, 2009].

Los ejemplos presentados en [Ceri et al., 2007], [Daniel, 2009] y [Daniel, 2010], describen una aplicación móvil que toma los datos del *GPS* y, en base a esta información muestran la página adecuada, para esto necesitan tener conexión *WIFI*. Para que el *Context Monitor* del lado del cliente se comunique con el módulo del *GPS* del celular usan la librería *Chaeron*¹¹⁴. Además obtienen la intensidad de la señal de *WIFI* usando *Place Lab*¹¹⁵ para alertar al usuario cuando la intensidad es baja y así detectar que la aplicación puede dejar de funcionar.

Los elementos que incorpora la extensión de *WebML* para aplicaciones Web sensibles al contexto, pueden ser representados usando una extensión de *WebRatio*¹¹⁶ y luego derivar el código correspondiente.

La extensión presentada de *WebML* para aplicaciones Web sensibles al contexto, tiene en común con nuestro enfoque, que ambos extienden de metodologías de aplicaciones Web como son *WebML* y *UWE* respectivamente, por lo tanto ambas tienen un modelo para representar la parte Hipermedial de la aplicación.

Una diferencia entre la extensión de *WebML* y nuestro enfoque, es que nosotros modelamos los elementos en el Modelo de Contenido usando el paradigma orientado a objetos, mientras que la extensión de *WebML* define su modelo de datos como un modelo de entidad-relación. Nuestro enfoque tiene representado el Modelo del Usuario como un modelo particular mientras que la extensión de *WebML* representa al usuario y sus características de contexto en el modelo de datos junto a otros datos de la aplicación.

Si bien ambos (la extensión de *WebML* y nuestro enfoque) tienen un Modelo Navegacional, cada uno define sus propios elementos, la extensión de *WebML* representa a este modelo en base a la definición de los sitios, las áreas, las páginas y las unidades de contenido, definiendo links entre las páginas y las unidades de contenido. En cambio nuestro enfoque define este modelo en base a nodos y links, los nodos pueden ser digitales o físicos y los links pueden ser digitales o caminables.

La extensión presentada de *WebML*, no tiene representado el concepto de nodos físico o links caminables, sino que modela los datos sólo a nivel digital y esta información a mostrar puede variar según los datos de contexto.

La información de contexto relevante en nuestro enfoque, es la posición del usuario y si éste es sensado por un Punto de Interés recibe la información digital y física del

¹¹⁴ Página de *Chaeron*: <http://www.chaeron.com>

¹¹⁵ Página de *Place Lab*: <http://www.placelab.org>

¹¹⁶ Los autores indican en [Daniel, 2010] que están trabajando en extender *WebRatio* para soportar la extensión para aplicaciones Web sensibles al contexto.

mismo. La extensión presentada de *WebML*, se focaliza en modelar los aspectos relevantes de contexto y además, incorpora la idea de mostrar información dependiente del contexto en las páginas. Para esto tienen embebido en estas páginas los mecanismos necesarios para controlar si la información de contexto cambia o no, y cuando cambia enviar un requerimiento automático para recalculación de la página.

Tanto la extensión de *WebML* como nuestro enfoque, tienen herramientas para el modelado de sus modelos y la derivación a código, nuestro enfoque provee una extensión para *MagicDraw* mientras que la extensión de *WebML* extiende *WebRatio* para soportar las características de las aplicaciones Web sensibles al contexto. Sin embargo estas herramientas están orientadas a distintos fines, *MagicDraw* está focalizado al modelado *UML*, mientras que *WebRatio* permite el desarrollo dirigido por modelos para la generación de aplicaciones Web.

La metodología para aplicaciones Web sensibles al contexto (extensión de *WebML*), sólo cuenta con una transformación de modelos a código no tiene especificada una transformación de modelo a modelo como si lo hace nuestro enfoque.

9.3. Resumen

En la Sección 9.1 se presentaron trabajos relacionados con nuestra tesis desde el punto de vista de las aplicaciones que podemos modelar con nuestro enfoque.

En la Tabla 9.1 se puede apreciar una comparación entre *Layar*, *Acrossair* y nuestras aplicaciones turísticas. La principal diferencia entre nuestras aplicaciones turísticas y las provistas por *Layar* y *Acrossair*, es que nuestras aplicaciones no muestran la imagen real superpuesta con información. Esto se debe a que, si bien las aplicaciones de HM pueden ser vistas como aplicaciones de Realidad Aumentada, no poseen todas las características de las mismas.

Todas (*Layar*, *Acrossair* y nuestras aplicaciones turísticas) usan *GPS*, sólo *Layar* y *Acrossair* usan los datos de la brújula, sólo *Acrossair* usa el acelerómetro para proveer diferente información y sólo nuestras aplicaciones turísticas permiten el uso de códigos de barra para obtener información.

Tanto las aplicaciones de *Layar* y *Acrossair* como nuestras aplicaciones turísticas, usan *Google Maps* para brindar mapas y búsqueda de caminos.

Tabla 9.1: Comparación de nuestras aplicaciones turísticas con las aplicaciones de *Layar* y *Acrossair*.

Característica a comparar	Aplicación <i>Layar</i>	Aplicación <i>Acrossair</i>	Nuestras aplicaciones turísticas
Usa <i>GPS</i>	Si	Si	Si
Usa Brújula	Si	Si	No
Usa Acelerómetro	No	Si	No
Usa lectura de código de barra	No	No	Si
Tipo de Aplicación	Realidad Aumentada	Realidad Aumentada	Hipermedia Móvil
Uso de Mapas	<i>Google Maps</i>	<i>Google Maps</i>	<i>Google Maps</i>

Se puede apreciar en la Tabla 9.2 una comparación de las principales características de *AudioMove* y nuestras obras de teatro. Ambas usan código de barra para poder recibir información e ir avanzando en la obra. Sólo en nuestras aplicaciones hay actores reales que interactúan con los jugadores. *AudioMove* posee una historia lineal que el usuario va siguiendo mediante audio. Nuestras historias no son lineales ni

proveen audio al jugador.

Tabla 9.2: Comparación de *AudioMove* con nuestras obras de teatro.

Característica a comparar	<i>AudioMove</i>	Nuestras obras de teatro
Usa lectura de código de barra	Si	Si
Interactúan actores reales	No	Si
Provee audio	Si	No
Historia lineal	Si	No

Una comparación entre *HasleInteractive* y nuestros juegos educativos es presentada en la Tabla 9.3. Se puede observar que ambos usan códigos de barra para acceder a la información de la aplicación. En cuanto a la finalidad de las aplicaciones ambas difieren, *HasleInteractive* está orientado a la recolección y análisis de situaciones para luego ser analizadas en clase junto al docente. Mientras que nuestros juegos educativos están pensados para que el alumno sea evaluado mientras lo realiza, es decir, las preguntas que contiene el juego permiten saber el grado de conocimiento del alumno. Ambos (*HasleInteractive* como nuestros juegos educativos) necesitan de docentes calificados para la creación del material que contenga el juego acorde a los alumnos que lo jueguen. *HasleInteractive* está planteado como juegos grupales mientras nuestros juegos educativos se realizan de forma individual para poder evaluar al alumno.

Tabla 9.3: Comparación de *HasleInteractive* y nuestros juegos educativos.

Característica a comparar	<i>HasleInteractive</i>	Nuestros juegos educativos
Usa lectura de código de barra	Si	Si
Finalidad de la aplicación	Recolección y análisis de información de los diferentes lugares	Evaluar al alumno mientras contesta preguntas
Preparación del material	Docentes calificados	Docentes calificados
Forma de Juego	Grupal	Individual

En la Sección 9.2 se presentaron dos trabajos relacionados con nuestro enfoque de desarrollo dirigido por modelos. En la Tabla 9.4 se puede apreciar una comparación entre *CAUSE*, la extensión de *WebML* para aplicaciones sensibles al contexto y nuestro enfoque. Se puede observar que todas son desarrollos dirigidos por modelos, sin embargo, cada uno sirve para crear aplicaciones diferentes con características particulares. La extensión de *WebML* para aplicaciones Web sensibles al contexto, tiene en común con nuestro enfoque que ambos extienden de dos metodologías de Hipermedia como son *WebML* y *UWE*, por lo tanto ambas tienen representado el Modelo Navegacional como parte del modelado. Aunque la extensión presentada de *WebML* no tiene representado el concepto de nodos físico o links caminables, sino que modela los datos sólo a nivel digital y esta información a mostrar puede variar según los datos de contexto. *CAUSE* no cuenta con un Modelo Navegacional de la aplicación, no tiene el concepto de link (digitales o caminables).

CAUSE define un metamodelo propio para definir los elementos de su metodología, y no utiliza la definición de un perfil *UML* como lo hace nuestro enfoque. La extensión de *WebML* para aplicaciones Web sensibles al contexto no tiene un metamodelo definido. *CAUSE* utiliza un plugin de *Eclipse* creado usando los frameworks de *Eclipse EMF* y *GMF*. La extensión de *WebML* tiene como herramienta de modelado *WebRatio*. Nuestro enfoque, provee como herramienta de modelado el plugin *MagicMobileUWE* para *MagicDraw*.

CAUSE define transformaciones complejas que consideran varios modelos para

generar varios otros modelos, mientras que nuestras transformaciones son de un sólo modelo a otro. *CAUSE* y nuestro enfoque proveen además transformaciones de modelo a código. La extensión de *WebML* sólo cuenta con una transformación de modelos a código.

Tabla 9.4: Comparación de *CAUSE*, la extensión de *WebML* para aplicaciones sensibles al contexto y nuestro enfoque.

Característica a comparar	<i>CAUSE</i>	Extensión de <i>WebML</i>	Nuestro enfoque
Desarrollo dirigido por modelos	Si	Si	Si
Tipo de aplicación que modela	Aplicaciones móviles sensibles al contexto en ambientes ubicuos	Aplicaciones Web sensibles al contexto	Aplicaciones de HM
Metamodelo	---	---	Perfil de <i>UML</i>
Metodología que extiende	Ninguna	<i>WebML</i>	<i>UWE</i>
Herramienta que provee para el modelado	Plugin de <i>Eclipse</i> (basado en los frameworks <i>EMF</i> y <i>GMF</i>)	<i>WebRatio</i> ¹¹⁷	Plugin <i>MagicMobileUWE</i> para <i>MagicDraw</i>
Transformaciones de modelo a modelo	Si	No	Si
Transformaciones de modelo a código	Si	Si	Si

De esta manera, quedaron presentados distintos trabajos relacionados con las aplicaciones que se crean usando nuestro enfoque como así también distintos trabajos relacionados con nuestro enfoque.

¹¹⁷ Los autores [Daniel, 2010] indican que están trabajando en la extensión de *WebRatio* para soportar las características de la extensión de *WebML*.

10. CONCLUSIONES Y TRABAJOS FUTUROS

En esta tesis se presentó un enfoque de desarrollo dirigido por modelos que permite crear aplicaciones de HM, que cuenta con seis modelos:

- Modelo de Contenido (*Plataform Independent Model - PIM*)
- Modelo Unificado (*Plataform Independent Model - PIM*)
- Modelo Navegacional (*Plataform Independent Model - PIM*)
- Modelo de Instancias Navegacional (*Plataform Independent Model - PIM*)
- Modelo de Presentación (*Plataform Specif Model - PSM*)
- Modelo del Usuario (*Plataform Specif Model - PSM*)

Para representar cada uno de los elementos necesarios de nuestros modelos se definió un perfil de UML llamado *Mobile Hypermedia*. El perfil *Mobile Hypermedia* importa cuatro perfiles (*Mobile UWE, Location, Layout* y *User Model*) como se puede observar en la Figura 10.1.

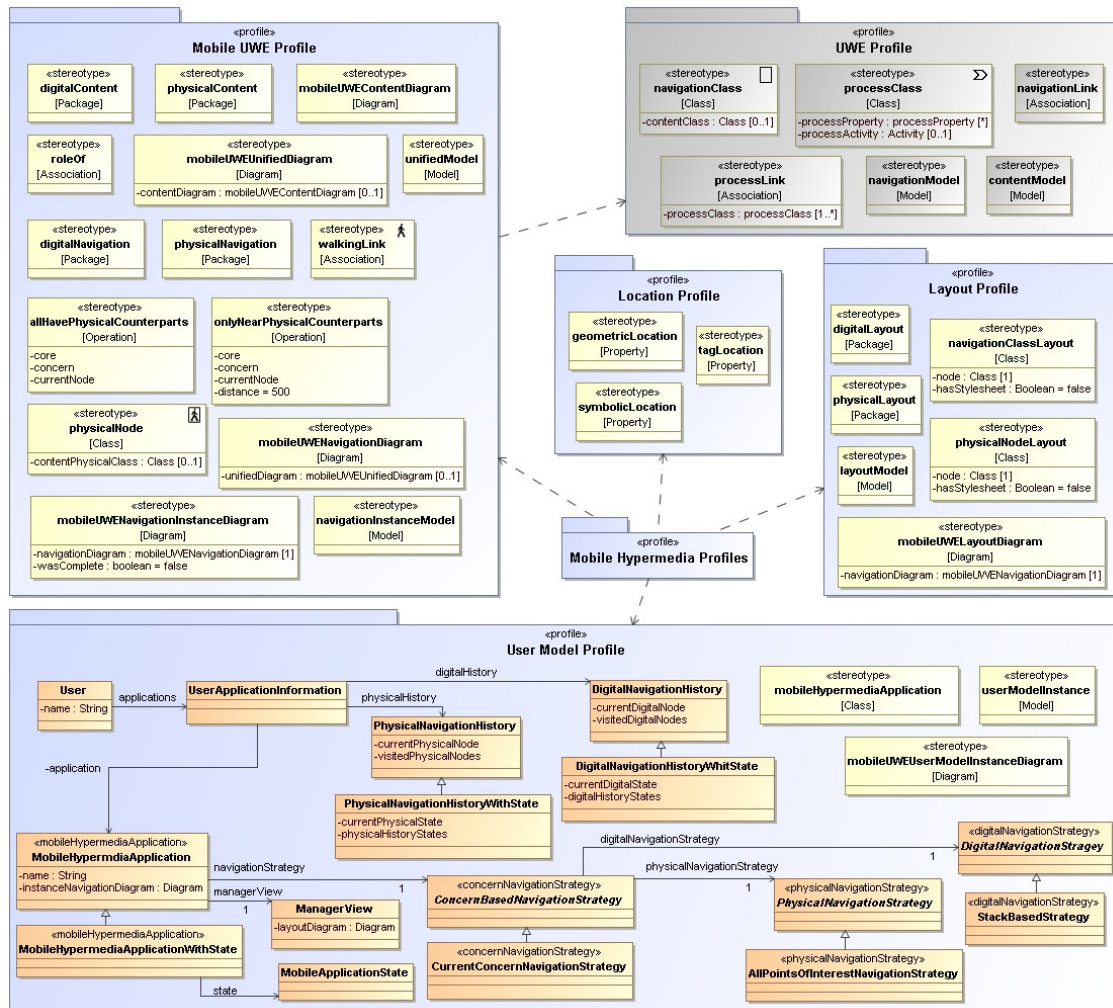


Figura 10.1: Elementos de nuestros perfiles UML.

Los perfiles *Mobile UWE, Location, Layout* y *User Model* contienen los estereotipos necesarios para representar los conceptos propios de HM. Esto permite que el enfoque presentado constituya una extensión conservativa de *UWE*, ya que no agrega

elementos nuevos a nivel del metamodelo de *UML*, sino que representa los conceptos mediante estereotipos.

En particular nuestro Modelo Navegacional usa, para el modelado de los concerns digitales, los elementos definidos por *UWE* (para su Modelo Navegacional). De esta manera se aprovechan las definiciones de los elementos Hipermediales ya definidos por el perfil de *UWE*.

El enfoque define tres tipos de transformaciones diferentes:

- Una transformación del Modelo de Contenido al Modelo Unificado considerando todos los concerns.
- Una transformación del Modelo de Contenido al Modelo Unificado considerando sólo los concerns de interés.
- Una transformación del Modelo Unificado al Modelo Navegacional.

Las tres transformaciones son entre modelos independientes de la plataforma (*Platform Independent Model - PIM*) y permiten al diseñador la generación semi-automática de los diferentes modelos. Los modelos resultantes (de cada transformación), pueden ser modificados para ajustarse a las necesidades de cada aplicación, esto se puede ver como un refinamiento que el diseñador aplica a cada modelo.

Un desarrollo dirigido por modelo, requiere de al menos una herramienta que permita llevarlo a cabo de manera automática (o semi-automática). Por esta razón, se creó una extensión para *MagicDraw* llamado *MagicMobileUWE* que permite crear cada uno de nuestros modelos, aplicar nuestras transformaciones y derivar una aplicación *J2ME* o Web Móvil (a partir de un Modelo del Usuario). Además, *MagicMobileUWE* provee diferentes menús contextuales que facilitan y agilizan la tarea del diseñador a la hora de crear aplicaciones de HM.

A lo largo de los Capítulos 6, 7 y 8 se describió cómo el diseñador, usando la herramienta, puede crear una aplicación de HM. A modo de resumen, se puede indicar que el diseñador puede:

- crear un Modelo de Contenido y definirle sus elementos (concerns, clases y relaciones). Cada clase del concern físico debe tener definida al menos una propiedad de ubicación.
- a partir de un Modelo de Contenido aplicar la transformación del Modelo de Contenido al Modelo Unificado considerando todos los concerns o la transformación del Modelo de Contenido al Modelo Unificado filtrándolos concerns de interés.
- teniendo un Modelo Unificado aplicar la transformación para generar un Modelo Navegacional. Este modelo resultante se puede refinar (agregar o sacar tanto sean nodos como links) según las necesidades de la aplicación que se este modelando. Además, puede definir, para cada uno de los nodos físicos, la función de links caminables derivados.
- crear un Modelo de Instancias Navegacional (a partir de un Modelo Navegacional de base) y definirle las instancias de los nodos digitales y físicos.
- crear un Modelo de Presentación (eligiendo cuál es el Modelo Navegacional de base), y a cada de las clase de presentación definirle un estilo de presentación.
- crear el Modelo del Usuario, eligiendo cuál es el Modelo de Instancias Navegacional y la presentación asociada.

- derivar una aplicación *J2ME* o Web Móvil a partir de un Modelo del Usuario.

En la Figura 10.2 se puede observar un esquema general de los ítems descritos anteriormente para crear una aplicación de HM usando la herramienta. Se especifican todos los modelos del enfoque, las transformaciones que se pueden aplicar entre ellos y las derivaciones (a aplicaciones) que se pueden realizar.

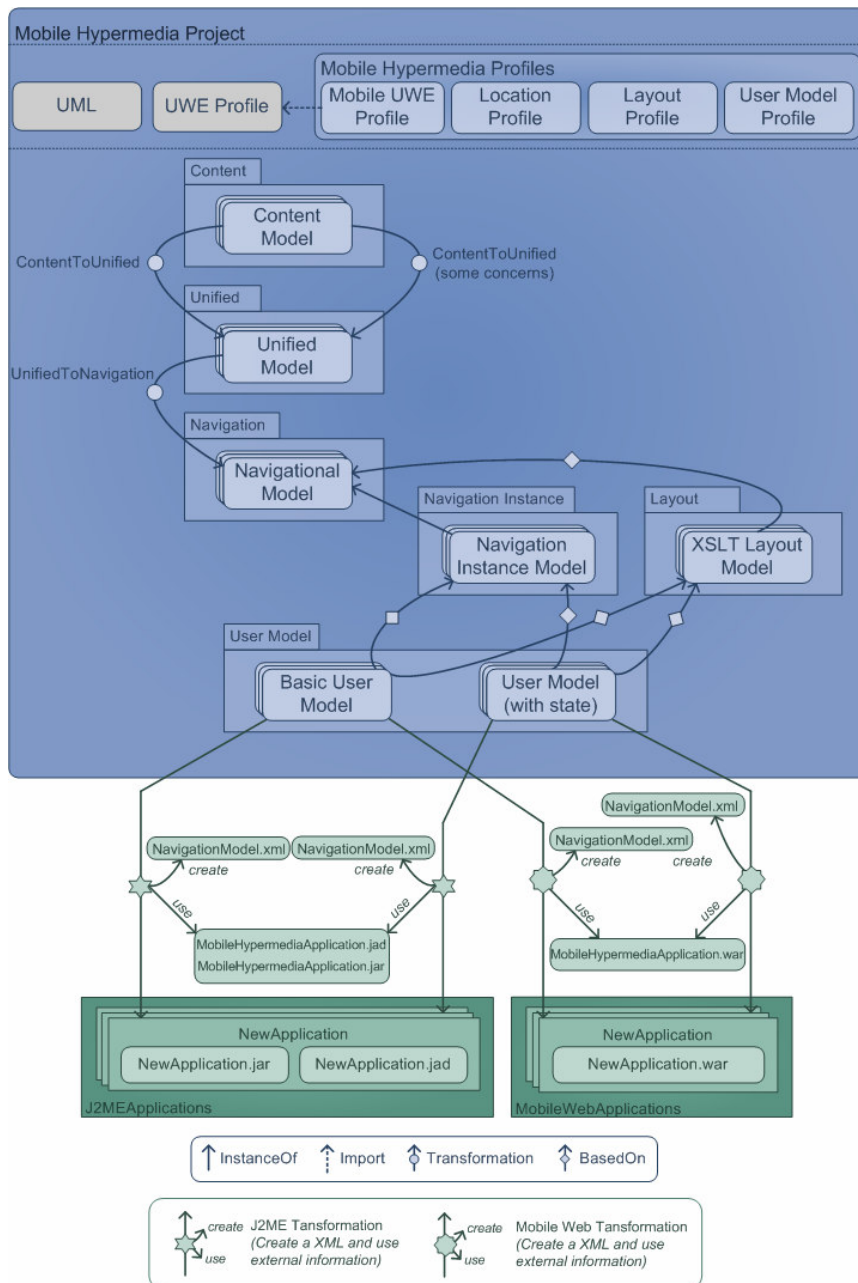


Figura 10.2: Desarrollo usando la herramienta *MagicMobileUWE*.

En esta tesis se pudo apreciar cómo nuestro desarrollo dirigido por modelo, cumple con los tres principios básicos:

- Representación directa. Usando estereotipos de *UML* para representar cada uno de los conceptos propios de las aplicaciones de HM.
- Automatización. La herramienta permite tanto la creación de nuestros modelos usando transformaciones, como así también derivar aplicaciones (*J2ME* y Web Móvil).

- Uso de estándares. Se creó una extensión conservativa de *UML*, es decir no se agrego ningún elemento nuevo a nivel de metamodelo. Por lo tanto, se puede afirmar que se usa para la representación de los conceptos de HM el estándar *UML* definido por la *OMG*. Además, *UML* se define en base *MOF* (metamodelo estándar definido por la *OMG*).

Aportes realizados por la Tesis

A continuación se retoman cada uno de los puntos presentados en el Capítulo 1, como aportes de la tesis, para ampliarlos acorde al desarrollo presentado en los Capítulos 2 al 9. El primer aporte presentado en el Capítulo 1 enunciaba:

- Determinar los conceptos específicos de aplicaciones de HM, que las diferencian de las aplicaciones de Hipermedia convencionales y las aplicaciones móviles.

En el Capítulo 3 se definieron las características propias que tienen las aplicaciones de HM, respecto de las aplicaciones de Hipermedia convencionales y de las aplicaciones móviles. Las aplicaciones de HM se pueden ver como una extensión de las aplicaciones de Hipermedia, agregando la complejidad de la movilidad del usuario en el mundo real y recibiendo información de los Puntos de Interés. Cada aplicación de HM debe tener registradas las posiciones de los Puntos de Interés, información que no es considerada en una aplicación de Hipermedia.

La diferencia entre las aplicaciones de HM y las aplicaciones móviles, es que estas últimas no cuentan necesariamente con una estructura Hipermedial como base de la aplicación. Si bien las aplicaciones móviles consideran la movilidad del usuario no tienen el concepto de nodo y link (tanto a nivel digital como físico). Por lo tanto, el usuario, que utiliza una aplicación móvil puede recibir información pero no necesariamente puede navegar. Las aplicaciones de HM, pueden considerarse aplicaciones móviles que permiten al usuario navegar digitalmente entre la información de los Puntos de Interés, como así también navegar en el mundo real. Es decir, las aplicaciones de HM son un subconjunto de aplicaciones móviles con características particulares relacionadas a la navegación.

Los conceptos propios incorporados por HM son los nodos físicos y links caminables, estos conceptos son modelados dentro del concerns físico (como se describió en el Capítulo 3). Además, estas aplicaciones (de HM) usan los conceptos propios de Hipermedia como son los concerns, nodos y links digitales. De las aplicaciones móviles, las aplicaciones de HM, usan los conceptos de movilidad del usuario y posicionamiento de los Puntos de Interés.

- Definir un enfoque que pueda ser utilizado para las aplicaciones de Hipermedia Móvil.

En el Capítulo 4 se mostró cómo extender una metodología de desarrollo dirigido por modelos para aplicaciones de Hipermedia de manera de soportar los conceptos propios de las aplicaciones de HM. La metodología extendida se denomina *UWE*, la cual se define como extensión conservativa de *UML*. Nuestra extensión se definió como una extensión conservativa de *UWE*, para lo cual se definió un perfil con los elementos propios de HM como se mostró en la Figura 10.1. En el Capítulo 5 se especificó cómo usar los elementos definidos por nuestro perfil en cada uno de nuestros seis modelos.

Los conceptos que no tiene modelado *UWE* y que fueron incorporados por nuestro enfoque (para modelar aplicaciones de HM) son:

- Los concerns digitales y físicos. En el modelo conceptual los concerns contienen clases, en el Modelo Navegacional los concerns

contienen nodos y los concerns del Modelo de Presentación contienen clases de presentación.

- La relación de *roleOf*, tanto entre clases como entre nodos.
- Los nodos físicos y los links caminables, propios del concern físico en el Modelo Navegacional. Para los nodos físicos la especificación de funciones para calcular links caminables derivados.
- Las propiedades de ubicación propias de las clases y nodos del concern físico.
- Clases de presentación (para los nodos digitales y físicos) propias de nuestro Modelo de Presentación.
- El Modelo del Usuario que modela al usuario y define las estrategias de navegación de la aplicación.

En particular, en nuestro Modelo Navegacional se usa, en los concerns digitales, los elementos de *UWE* definidos para su Modelo Navegacional. Aprovechando, de esta manera, la definición de *UWE* de los elementos propios de Hipermedia.

- Incorporar elementos de separación avanzada de concerns tanto verticales como horizontales (aplicativos y paradigmáticos).

En el enfoque contamos con separación de concerns en dos niveles diferentes, por un lado los distintos modelos que conforman el enfoque y, por otro, los concerns propios del dominio de la aplicación.

Los seis modelos del enfoque (enunciados al principio de este capítulo) pueden verse como una separación de concerns horizontales, cada modelo se encarga de representar alguna cuestión particular de la aplicación. Es decir, cada modelo puede ser visto como un concern, el cual cuenta con características propias.

En el Capítulo 5 se describió cada uno de nuestros modelos y cómo los mismos se relacionan, es decir se establecieron las relaciones entre estos concerns horizontales.

El otro nivel de concerns (concerns verticales) está dado acorde al dominio de la aplicación dentro de cada modelo. Estos concerns están representados en nuestros modelos por los concerns digitales y el concern físico. Los concerns digitales son creados acordes al dominio de la aplicación que se está modelando, esto se pudo apreciar en los ejemplos presentados en el Capítulo 7. Los concerns en la aplicación del turista permiten crear distintas vistas del mismo Punto de Interés, mientras que en los ejemplos del juego educativo y la obra de teatro, los concerns servían para representar distinta funcionalidad de la aplicación. Las aplicaciones de HM pueden contar con varios concerns digitales y un concern físico (obligatorio).

Para cada modelo, los concerns digitales y físico contienen distinta semántica, en el Modelo de Contenido (y Unificado) contienen clases de la aplicación, es decir son concerns de contenido. En el Modelo Navegacional los concerns contienen nodos tanto digitales como físicos, es decir son concerns navegacionales. Esto permite que los usuarios puedan navegar por la información de un concern determinado. Por otro lado, en el Modelo de Presentación los concerns contienen clases de presentación de la aplicación.

La separación de concerns, ofrece como ventaja, la posibilidad de hacer reingeniería y realizar un mejor mantenimiento de la aplicación, además de permitir la generación de aplicaciones para diferentes plataformas. Estos conceptos de separación de concerns también son compartidos por el desarrollo dirigido por modelo que además permite la derivación automática y la agilidad en la creación de aplicaciones.

- Disponer de un lenguaje genérico que pueda ser aplicado a dominios particulares.

En el Capítulo 7, se mostró cómo usando nuestro desarrollo se pueden modelar aplicaciones de dominios diferentes como por ejemplo: aplicaciones turísticas, juegos educativos y obras de teatros. Además, en el Capítulo 8 se ejemplificó cómo derivar a partir de nuestros modelos aplicaciones concretas.

Los elementos de nuestro perfil, permiten especificar aplicaciones de HM con diferentes dominios. Los estereotipos creados en nuestro perfil posibilitan representar los conceptos básicos de este tipo de aplicaciones (HM) permitiendo que el diseñador defina usando *UML* el Modelo de Contenido acorde al dominio que está modelando. Además, el diseñador puede usar estereotipos externos para representar información particular de la aplicación ya que nuestros modelos son compatibles con cualquier elemento de *UML*.

En los ejemplos presentados en el Capítulo 7, se pudo observar cómo para cada dominio se creó un Modelo de Contenido acorde a sus características. Para los modelos generados a partir del Modelo de Contenido, sólo fue necesario usar los elementos de nuestro perfil para representar cada concepto, llegando luego a derivar cada aplicación.

- Especificar e implementar herramientas de transformación de modelos, para derivar sucesivamente los modelos del enfoque hasta obtener aplicaciones ejecutables en el contexto del desarrollo dirigido por modelos.

En el Capítulo 6 se presentó la herramienta *MagicMobileUWE* que permite el desarrollo dirigido por modelos para aplicaciones de HM, mostrando cómo se puede crear cada uno de nuestros modelos como así también cómo realizar cada una de nuestras transformaciones. En el Capítulo 8, se mostró la derivación de aplicaciones ejecutables. En los Anexos A, B y C se especifican características básicas de implementación de la herramienta.

Trabajos Futuros

A partir del desarrollo dirigido por modelos presentado en esta tesis se desprenden distintos trabajos futuros relacionados, los cuales son enumerados a continuación:

- ❖ Proveer otras transformaciones entre modelos a parte de las ya presentadas en esta tesis. Por ejemplo, una transformación del Modelo Unificado al Modelo Navegacional que permita el filtrado de elementos, es decir que el diseñador pueda especificar qué concerns, clases, asociaciones y propiedades quiere pasar al Modelo Navegacional. Esto se podría realizar mostrando en una ventana los elementos del Modelo Unificado y, permitiendo que el diseñador pueda indicar (por ejemplo con un checkbox) aquellos elementos de interés, al realizar la transformación sólo se pasarían aquellos elementos seleccionados al Modelo Navegacional. Esta transformación evitaría que el diseñador genere un Modelo Navegacional y luego tenga que borrar información (concerns, clases, asociaciones y propiedades).
- ❖ Agregar en las transformaciones controles respecto del modelo que se quiere transformar. Por ejemplo, en la transformación del Modelo de Contenido al Unificado, controlar que las clases tengan nombre, que estén contenidas en un determinado concern, que las clases del concern físico tengan definida al menos una propiedad de ubicación. Estos controles permitirían que el diseñador detecte estas problemáticas lo antes posible, y no las arrastre hasta la derivación de la aplicación.
- ❖ Investigar otras opciones para llevar a cabo las transformaciones, como por

ejemplo, usar *QVT (Query - View - Transformation)*, *ATL (Atlas Transformation Language)*, etc. para luego evaluar cuál es la mejor solución a ser utilizada. Actualmente, las transformaciones están definidas a nivel de código *Java* por una cuestión de simplicidad.

- ❖ Definir más funciones para derivar links caminables, especificando tanto el comportamiento de las mismas, como agregando el estereotipo correspondiente en nuestro perfil. Además, agregar en la herramienta, la posibilidad de elegir estos nuevos estereotipos a la hora de crear la función de links caminables derivados (en los nodos físicos). Algunas posibles funciones podrían ser:
 - Una función que calcule links caminables derivados considerando el nodo digital que está visualizando el usuario, cuando éste no se corresponde con el nodo físico en el que está localizado el usuario.
 - Una función que lleve un historial de links caminables derivados sugeridos y a ese historial le sume los links caminables derivados a partir de la función mencionada anteriormente. De esta manera no solamente se le provee al usuario caminar al lugar visualizado digitalmente, sino además, que pueda caminar a todos aquellos lugares que fue navegando digitalmente.
 - Una función que tome los Puntos de Interés cercanos al usuario y derive, a partir de los mismos, links caminables. Donde la cercanía se define como un rango de distancia desde dónde está parado el usuario. A medida que el usuario va cambiando su posición estos links caminables se van actualizando. A diferencia de las anteriores funciones, ésta no tiene en cuenta cuestiones de navegación del usuario.
- ❖ Cuando está realizando un camino, producto de haber seleccionado un link caminable, enriquecerle el mapa acorde a la información digital que está visualizando. Es decir, la información de los concerns digitales también podría servir para enriquecer la información que se proporciona al usuario, cuando se realiza una navegación en el mundo real. Supongamos que el usuario está visualizando un concern digital y éste contiene links digitales, cuyos targets tienen contrapartes físicas, se podría usar la información de la posición de estos Puntos de Interés para indicarlos en el mapa que visualiza el usuario junto con el camino que tenía establecido. De esta manera, el usuario puede percibir estos lugares mientras realiza la navegación en el mundo real. En el caso que el camino que tiene que recorrer pase cerca de algún de estos Puntos de Interés, que está viendo digitalmente, podría aprovechar para también verlos físicamente. Esta funcionalidad, si bien no está implementada actualmente, se podría generar a partir de la información que maneja la aplicación.
- ❖ Agregar en el Modelo Navegacional la posibilidad de especificar links digitales y caminables que tengan condiciones asociadas. Esto es útil para aplicaciones como el juego educativo, donde los links caminables sólo se muestran cuando el alumno contestó una pregunta. Para que las condiciones sean útiles, a la hora de derivar aplicaciones, se debe definir la sintaxis específica para las mismas. Para lograr incorporar esta funcionalidad, no solamente hay que modificar la herramienta, sino que además se debe crear el estereotipo correspondiente para representar links con condición. La condición puede ser especificada como un valor etiquetado definido en el estereotipo.
- ❖ Analizar y evaluar (para incorporar a la herramienta) la necesidad o no de tener

más elementos representados en el concern físico del Modelo Navegacional, además de los nodos y links. Por ahora, se tiene los nodos digitales y físicos junto con sus links digitales y caminables respectivamente, pero a nivel digital, se tienen otros elementos como, por ejemplo, las clases de procesamiento. Un posible trabajo futuro sería tomar los elementos de los concerns digitales, y ver si estos tienen su correspondencia en el concern físico, por ejemplo si es necesario tener una clase de procesamiento en el concern físico del Modelo Navegacional.

- ❖ Actualmente los Modelos de Contenido, Unificado y Navegacional se crean desde el menú principal (es decir, que no provienen de una transformación), para ahorrarle tiempo al diseñador se podrían crear con al menos un concern digital y el concern físico. También se le podría permitir al diseñador indicar cuantos concerns digitales quiere considerar en el modelo a crear, para que el modelo creado defina tantos concerns digitales como indicó el diseñador. Esto evitaría que el diseñador deba crear uno por uno desde la barra de herramientas.
- ❖ Por ahora se definió un posible Modelo de Presentación basado en *XSLT*, un trabajo futuro sería proveer otros Modelos de Presentación. Sería interesante que diseñador pudiera trabajar con herramientas de diseño visuales específicas, para crear interfaces y las mismas estén integradas a la herramienta para poder generar una presentación adecuada para los nodos de la aplicación.
Otra alternativa, es brindarle al diseñador la posibilidad de crear modelos de presentación basados en plantillas de visualización, por ejemplo, *CSS*¹¹⁸ (*Cascading Style Sheet*).
- ❖ El Modelo del Usuario se está creando actualmente con estrategias de navegación por default, una mejora que puede realizarse a la herramienta es permitir que el diseñador especifique las estrategias de navegación. Además permitirle elegir si el Modelo del Usuario a crear tendrá una aplicación que considere estados o no. Estas modificaciones se deberían realizar en la herramienta, para que cuando el usuario selecciona la opción de crear un Modelo del Usuario, le aparezcan distintas ventanas para que pueda elegir cada una de las características descritas (estrategias de navegación, aplicación con estados o no).
- ❖ Por ahora, para poder tener aplicaciones para diferentes perfiles de usuarios se debe crear un Modelo de Instancias Navegacional acorde a la información de cada perfil. Luego, se deben crear los correspondientes Modelos del Usuario y derivar, a partir de cada uno de ellos una aplicación, como resultado, se obtiene una aplicación por cada perfil. Un trabajo futuro consiste en que la derivación genere una única aplicación que tenga embebida la información para todos los perfiles (acordes a la naturaleza de la aplicación).
- ❖ Las derivaciones de las aplicaciones (*J2ME* y *Web Móvil*) por ahora sólo consideran los datos del Modelo de Instancias Navegacional para crear un archivo *NavigationModel.xml*. Las derivaciones deberían contemplar además otras cuestiones como:
 - Agregar como parte de la derivación el Modelo de Contenido, permitiendo que la aplicación generada no solamente tenga información de los nodos y links sino que además tenga cuestiones de funcionalidad especificada en el Modelo de Contenido.

¹¹⁸ Página de la *W3C* sobre *CSS*: <http://www.w3.org/Style/CSS/>

- Agregar el Modelo de Presentación *XSLT* a la derivación de una aplicación Web Móvil, permitiendo de esta manera que la visualización de las páginas se hagan acordes a la especificación de cada clase de presentación. Se debe establecer cuál es la nomenclatura usada para definir cada uno de los parámetros de la clase de presentación, para determinar de dónde se sacan los datos para completar los parámetros del archivo *XSLT*. Cada parámetro del archivo *XSLT* se puede completar con un nombre que puede corresponderse a una propiedad, a una asociación, a una operación, etc. Es decir, que el valor definido en el parámetro tiene varias interpretaciones. Una posible solución es anteponerle al nombre una inicial para desambiguar y poder determinar el significado de ese valor, por ejemplo: *p_* (propiedades), *a_* (asociación) y *o_* (operación).
- Agregar las clases de procesamiento del Modelo Navegacional como parte de la derivación, permitiendo de esta manera, no solamente tener una aplicación derivada con los nodos y links, sino además tener links y clases de procesamiento. Además la derivación debería agregar, en el archivo *NavigationModel.xml*, información referente a otros elementos del los concerns digitales del Modelo Navegacional como son los menús, índices, búsquedas, etc.

Al considerar más información en la derivación, se debe modificar además las aplicaciones usadas de base (*J2ME* y *Mobile Web*) para que se pueda usar toda la información derivada. Por ahora las aplicaciones de base usadas en la derivación, sólo consideran el archivo *NavigationModel.xml* para tomar el contenido que debe mostrar la aplicación.

- ❖ Agregar en las aplicaciones (*J2ME* y *Web Móvil*) usadas de base para las derivaciones las siguientes cuestiones:
 - La funcionalidad para que el usuario pueda realizar el back y next tanto a nivel digital como caminable. Esta funcionalidad, debe estar acorde a las estrategias de navegación elegidas por el diseñador, cuando creó el Modelo del Usuario.
 - Más nivel de detalle en los mapas que se le muestran al usuario, por ejemplo, marcándole los lugares ya visitados en el mapa. Podría ayudar al usuario marcarle de manera diferente los links caminables a lugares ya visitados. Por ejemplo, un usuario está recorriendo una ciudad, va visitando lugares y en un determinado Punto de Interés tiene links caminables a lugares que ya visito, la aplicación podría marcárselos diferentes para que el usuario identifique esta situación (que ya visitó estos lugares).
 - Permitirle al usuario, cuando selecciona un link caminable, especificar el criterio para la búsqueda de camino, por ejemplo, el más rápido, el más pintoresco. Por ahora, se provee el camino acorde al resultado que brinda *Google Directions*. También podría ser parte del cálculo de camino, tener que realizar alguna tarea mientras se lleva a cabo el mismo, por ejemplo, un turista podría querer comprar regalos mientras realiza una navegación en el mundo real. Para esto, el cálculo de camino también debe incluir las tiendas de regalos que podrían estar en el camino pautado, logrando que el usuario llegue a destino elegido y además logre completar su tarea.
 - Brindar la posibilidad de pausar la navegación en el mundo real, es

decir, que el usuario pueda dejar pausado un camino. Por ejemplo, detenerse para almorzar, y una vez que termina, poder retomar el camino que tenía pendiente. Se debe analizar la dinámica de esta operación, y que problemáticas podrían llegar a surgir, por ejemplo, cuantos caminos se pueden dejar pausados o que pasa si cuando retoma el camino no está en la misma posición de cuando lo había pausado.

- ❖ Derivar aplicaciones a otras plataformas usando diferentes frameworks Web como, por ejemplo, *JSF (Java Server Faces)* como lo hace *UWE* (para la derivación de aplicaciones de Hipermedia). Actualmente, la herramienta define la derivación de aplicaciones Web Móvil usando en la aplicación base *Servlets* y *JSPs*.
- ❖ Las aplicaciones *J2ME* derivadas fueron probadas usando un emulador de *Samsung*, como trabajo futuro queda realizar pruebas en celulares, para detectar si hay que realizar algún tipo de ajuste en la derivación. Lo mismo sucede con las aplicaciones Web Móvil derivadas, fueron probadas desde la versión desktop del browser *Opera Mobile*, que permite probar aplicaciones Web. Como trabajo futuro resta probar estas aplicaciones desde un browser instalado en un celular, para detectar si la aplicación derivada necesita alguna otra especificación. Como trabajo futuro se realizarán pruebas en celulares de diferentes marcas para lograr que la derivación abarque la mayor cantidad posible variantes. Esto puede generar que se deban realizar ajustes en las aplicaciones de base.
- ❖ Crear con el enfoque aplicaciones más complejas. En la Sección 7.3 se describió cómo las aplicaciones turísticas, los juegos educaciones y las obras de teatros se pueden volver más complejas. Modelar estas características más complejas de cada una de las aplicaciones, para comprobar si se necesitan definir otros elementos a nivel de perfil (es decir, crear nuevos estereotipos) o se necesita mejorar la herramienta (en cuanto a los menús contextuales que se provee).
- ❖ Analizar qué características debería incorporar el enfoque si se consideran aplicaciones de HM que combinan posicionamiento indoor-outdoor, sobre todo a nivel del concern físico. Por ejemplo, caminos (generados al seleccionar un link caminable) que involucren salir de un edificio, caminar por la ciudad y entrar en otro edificio.
- ❖ Analizar cómo llevar a cabo reingeniería (o sincronización) entre los diferentes modelos. Es decir, que los cambios en un modelo se reflejen en el resto. Ya que, a partir de un modelo se pueden derivar varios, por ejemplo de un Modelo de Contenido se pueden derivar varios Modelos Unificados, hacer reingeniería trae como problemática la inconsistencia que se puede generar entre modelos. Todos los modelos unificados que se generaron a partir de este Modelo de Contenido, también deben ser modificados para que no se genere inconsistencia. A su vez, estos modelos fueron la base para crear otros modelos así que las modificaciones se deberían propagar, afectando a muchos modelos y generando posibles inconsistencias. Como base tomar los conceptos de sincronización presentados por *UWE* [Ruiz-González et al., 2009].
- ❖ Actualmente *UWE* está trabajando para incorporar conceptos de *RIA (Rich Internet Applications)* [Kolsch et al., 2009] tanto a nivel de la metodología como en su plugin *MagicUWE*. Un posible trabajo futuro, sería también considerar estas nuevas incorporaciones de *UWE* en el enfoque.

- ❖ *UWE* cuenta con diagramas de flujos para determinar el funcionamiento de sus aplicaciones Web. Un posible trabajo futuro es evaluar los beneficios o desventajas que nos puede traer usar diagramas de flujos en nuestro desarrollo. Esto traería como consecuencia, la dificultad de integrar las aplicaciones que usamos de base con los flujos de información definidos por un diseñador.

En este capítulo se detallo un resumen de la tesis presentada como así también se enunciaron distintos trabajos fututos que surgieron a partir de la misma.

ACRÓNIMOS

API – *Application Program Interface*
ATL – *Atlas Transformation Language*
CDC – *Connected Device Configuration*
CIM – *Computation-Independent Model*
CLDC – *Connected Limited Device Configuration*
DSL – *Domain Specific Language*
DTD – *Document Type Definition*
DTO – *Data Transfer Object*
EMF – *Eclipse Modelling Framework*
ER – *Entity Relationship*
FOHM – *Fundamental Open Hypermedia Model*
GMF – *Graphical Modelling Framework*
GPS – *Global Positioning System*
GUI – *Graphical User interface*
HM – *Hipermedia Móvil*
HTTP – *Hypertext Transfer Protocol*
J2EE – *Java Enterprice Edition*
J2ME – *Java Micro Edition*
JSP – *Java Server Pages*
JSF – *Java Server Faces*
J2SE – *Java Standard Edition*
MDA – *Model Driven Architecture*
MDD – *Model Driven Development*
MDWE – *Model Driven Web Engineering*
MID – *Mobile Information Device*
MOF – *Meta Object Facility*
OMG – *Object Management Group*
OOHDM – *Object Oriented Hypermedia Design Method*
PIM – *Plataform Independent Model*
POI – *Point of Interest*
PSM – *Plataform Specif Model*
QVT – *Query / View / Transformation*
RIA – *Rich Internet Applications*
RFID – *Radio Frequency Identification*
UML – *Unified Modeling Language*
URL – *Uniform Resource Locator*
UWE – *UML-based Web Enginnering*
WebML – *Web Modeling Language*

W3C – *World Wide Web Consortium*

XMI – *XML Metada Interchange*

XML – *Extensible Markup Language*

XSLT – *Extensible Stylesheet Language Transformation*

GLOSARIO DE TÉRMINOS Y DEFINICIONES

COMPUTACIÓN MÓVIL

Computación Móvil según [Talukder and Yavagal] se puede definir como en entorno de computación sobre la movilidad física. Un usuario puede acceder a los datos, información o cualquier otro objeto lógico desde cualquier dispositivo mediante cualquier red mientras se va moviendo. Los sistemas de computación móvil permiten al usuario desarrollar una tarea en cualquier lugar usando su dispositivo móvil.

Según [Roy et al., 2003] Computación Móvil se puede definir como la computación que se produce al mismo tiempo que un usuario está usando un dispositivo móvil mientras se mueve.

En [Lyytinen and Yoo, 2002] se define Computación Móvil como un servicio que se mueve con las personas, el cual permite que las mismas se puedan inscribir, recordar, comunicar y razonar independientemente de la posición de los dispositivos.

CONCERN

Un concern según [Kent, 2002] especifica un determinado tema o área de dominio de una aplicación. Según la clasificación especificada en [Nanard et al., 2008] puede haber dos niveles diferentes de separación de concerns, por un lado a nivel de la metodología mediante los distintos modelos que la conforman y, por otro, los concerns propios del dominio de la aplicación. Los primeros son denominados concerns horizontales mientras que los segundos son concerns verticales.

DESARROLLO DIRIGIDO POR MODELOS

En [Gherbi et al., 2009] se indica que la motivación del desarrollo dirigido por modelos es aprovechar los esfuerzos empleados en el análisis y concepción de las aplicaciones facilitando la migración de aplicaciones de una plataforma a otra. Se hace hincapié en la importancia de modelar una única vez y luego generar aplicaciones para distintas plataformas.

Según [Atkinson and Kuhne, 2003] el objetivo (del desarrollo dirigido por modelos) es automatizar las tareas complejas buscando reducir el tiempo de creación de las aplicaciones. En particular, hacen hincapié en la derivación automática de aplicaciones.

El objetivo que destacan los autores en [Hailpern and Tarr, 2006] es reducir las tareas del desarrollador y la complejidad de las aplicaciones para lo cual elevan el nivel de abstracción usando modelos para crear aplicaciones y hacer que las mismas evolucionen.

HIPERMEDIA

Hipermedia es un paradigma para organizar y acceder a información [Bieber et al., 1997]. Esta información está organizada mediante nodos que están conectados mediante links. Cada nodo contiene información multimedia. El usuario recorre el espacio de información navegando desde un nodo a otro utilizando los links, este tipo de navegación en el marco de esta tesis se considera navegación digital.

HIPERMEDIA MÓVIL

Hipermedia Móvil (también llamada en [Gronbaek et al., 2003] Hipermedia Física) extiende el concepto de Hipermedia incorporando la complejidad de la movilidad del usuario en el mundo real. La información está representada según [Challiol et al., 2006] mediante nodos digitales y físicos los cuales están relacionados

mediante links digitales y caminables respectivamente. Cuando un usuario está parado frente a un objeto del mundo real determinado recibe información del mismo (nodo digital y físico) pudiendo navegar digitalmente o en el mundo real (cuando selecciona un link caminable).

LINK

Un link conecta dos nodos permitiendo la navegación desde uno hacia el otro. En el marco de esta tesis se utilizan dos tipos de links, por un lado los links digitales entre nodos digitales [Bieber et al., 1997] y por otro los links caminables entre nodos físicos [Harper et. al., 2004].

MAGICDRAW

*MagicDraw*¹¹⁹ es una herramienta de modelado *UML* (soporta la versión de 2.0 de *UML*), está realizada íntegramente en *Java*. Provee una *API* abierta que permite la fácil extensión de funcionalidad mediante la creación de plugins para dicha herramienta. Además, de crear diferentes modelos *UML* permite la creación de perfiles de *UML*.

En esta tesis esta herramienta se utilizó como base para nuestro desarrollo dirigido por modelos a la cual le creamos una extensión para contar con todas las características de nuestro enfoque. También se utilizó para la definición de nuestros perfiles, los cuales cuenta con todos los conceptos propios de las aplicaciones de Hipermedia Móvil usados en nuestros modelos.

NODO

Un nodo representa contenido, el cual puede ser textual, audio, imágenes, etc. En esta tesis se distinguen dos tipos de nodos, los nodos digitales que representan información Hipermedial [Bieber et al., 1997] y los nodos físicos para representar los objetos del mundo real [Challioli et al., 2006].

NAVEGACION

Un link conecta dos nodos, cuando el usuario selecciona dicho link navega de un nodo hacia el otro. En el marco de esta tesis se utilizan dos tipos de navegaciones: la navegación digital [Bieber et al., 1997] y la navegación en el mundo real [Challioli et al., 2006]. La navegación digital se establece cuando un usuario selecciona un link digital, como respuesta recibe el nodo target del link. Esta navegación se denomina atómica porque inmediatamente el usuario recibe la información del destino del link. Por otro lado, la navegación en el mundo real se establece cuando un usuario selecciona un link caminable, como respuesta recibe un camino al target del link. Esta navegación implica que el usuario debe caminar hasta el destino elegido (target del link), la navegación en el mundo real finaliza cuando el usuario llega al destino (target).

GEOLOCATION

Geolocation¹²⁰ es la identificación de la ubicación geográfica (en el mundo real) de un objeto, como puede ser un celular. Geolocation puede referirse a la práctica de acceder a una ubicación o para la evaluar la ubicación actual.

UML

*UML*¹²¹ (*Unified Modeling Language*) es un lenguaje de modelado definido por la *OMG* (*Object Management Group*) para definir o especificar no solamente la

¹¹⁹ Página de *MagicDraw*: <http://www.magicdraw.com/>

¹²⁰ Definición en Wikipedia sobre Geolocation: <http://en.wikipedia.org/wiki/Geolocation>

¹²¹ Página de *UML*: <http://www.uml.org/>

estructura de una aplicación, el comportamiento, la arquitectura sino también el proceso de negocios y las estructuras de datos. La versión de *UML* 2.0 cuenta con trece diagramas divididos en tres categorías: diagramas de estructuras, diagramas de comportamiento y diagramas de interacción. En particular, en esta tesis nos focalizamos en un tipo específico de diagrama de estructura, como son los diagramas de clases.

UML – Perfil

Un Perfil de *UML* permite la extensión del metamodelo de *UML* a las necesidades concretas de una plataforma o dominio de una aplicación. Según [Fuentes and Vallecillo, 2004] un perfil se define en un paquete *UML*, estereotipado como «profile», puede servir para extender un metamodelo o a otro perfil de *UML*. Los mecanismos que se utilizan para definir perfiles son tres: estereotipos, restricciones y valores etiquetados.

UWE

*UWE*¹²² (*UML-Based Web Engineering*) es una extensión conservativa de *UML* para modelar aplicación Web para lo cual definieron un perfil llamado *UWE*. Este perfil cuenta con todos los conceptos para modelar los elementos de los modelos de *UWE*. Una de las herramientas que provee *UWE* es un plugin para *MagicDraw* llamado *MagicUWE*¹²³ este plugin agrega la funcionalidad para realizar desarrollo dirigido por modelos de aplicación Web siguiendo la metodología definida por *UWE*.

¹²² Página de *UWE*: <http://uwe.pst.ifi.lmu.de/>

¹²³ Página de *MagicUWE*: <http://uwe.pst.ifi.lmu.de/toolMagicUWE.html>

BIBLIOGRAFÍA

- [Ali et al., 2005] Ali, M., Ben-Abdallah, H. and Gargouri, F.: Towards a validation approach for UP conceptual models. In *Proceeding of European Conference on Model Driven Architecture - Foundations and Applications*, 2005, pp. 143-152.
- [Atkinson and Kuhne, 2003] Atkinson, C. and Kuhne, T.: Model-driven development: a metamodeling foundation. *Journal of IEEE Software*, 2003, Vol. 20, N° 5, pp. 36-41.
- [Azuma et al., 2001] Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S. and MacIntyre, B.: Recent Advances in Augmented Reality. In *Proceedings of IEEE Comput. Graph. Appl.* 2001, Vol. 21, N° 6, pp. 34-47.
- [Bang-Jensen and Gutin, 2007] Bang-Jensen, J. and Gutin, G.: *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, 2007.
- [Bézivin, 2006] Bézivin, J.: Model Driven Engineering: An Emerging Technical Space. In *Proceedings of Generative and Transformational Techniques in Software Engineering (GTTSE)*, Springer, 2006, LNCS 4143, pp. 34-64.
- [Bieber et al., 1997] Bieber, M., Vitali, F., Ashman, H. Balasubramanian, V. and Oinas-Kukkonen, H.: Fourth generation hypermedia: Some missing links for the World Wide Web. In *International Journal of Human-Computer Studies*, 1997, Vol. 47, N° 1, pp. 31-65.
- [Bouvin et al., 2003] Bouvin, N.O., Christensen, B.G., Gronbaek, K. and Hansen, F.A.: HyCon: a framework for context-aware mobile hypermedia. *Journal of The New Review of Hypermedia and Multimedia*, 2003, Vol. 9, N°1, pp. 59-88.
- [Bouvin, 2004] Bouvin, N.O.: Spatial hypermedia 1.0.
www.daimi.au.dk/~bouvin/hypermediar/2004/10/talk.html (12/08/2010).
- [Busch and Koch, 2009] Busch, M. and Koch, N.: MagicUWE - A CASE Tool Plugin for Modeling Web Applications. In *Proceedings of 9th International Conference Web Engineering (ICWE'09)*, Springer, 2009, LNCS 5648, pp. 505-508.
- [Brambilla et al., 2008] Brambilla, M., Comai, S., Fraternali, P. and Matera, M.: Designing Web Applications with WebML and Webratio. In *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, O. Pastor, D. Schwabe and L. Olsina (Eds.), Springer, 2008, Chapter 9, pp. 221-262.
- [Cabot and Raventós, 2006] Cabot, J. and Raventós R.: Conceptual Modeling Patterns for Roles. *Journal on Data Semantics V*, 2006, pp. 158-184.
- [Ceri et al., 2003] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S. and Matera, M.: *Designing Data-Intensive Web Applications*. Morgan Kaufmann Series in Data Management Systems, 2003.
- [Ceri et al., 2007] Ceri, S., Daniel, F., Facca, F. and Matera, M.: Model-driven Engineering of Active Context Awareness. In *World Wide Web Journal*, Springer Verlag, 2007, Vol. 10, N° 4, pp. 387-413.
- [Challiol et al., 2006] Challiol, C., Rossi, G., Gordillo, S. and De Cristófolo, V.: Systematic Development of Physical Hypermedia Applications. In *International Journal of Web Information Systems (IJWIS)*, 2006, pp. 232-246.
- [Challiol et al., 2007] Challiol, C., Muñoz, A., Rossi, G., Gordillo, S.E., Fortier, A. and Laurini, R.: Browsing Semantics in Context-Aware Mobile Hypermedia. In *Proceedings of OTM 2007 Workshops*. LNCS 4805, 2007, pp. 211-221.
- [Challiol et al., 2008] Challiol, C., Fortier, A., Gordillo, S.E. and Rossi, G.: Model-Based Concerns Mashups for Mobile Hypermedia. In *Proceedings of MoMM '08*, ACM, 2008, pp. 170-177.
- [Daniel, 2009] Daniel, F.: Context-Aware Applications for the Web: A Model-Driven Development Approach. In *Context-Aware Mobile and Ubiquitous Computing for*

- Enhanced Usability: Adaptive Technologies and Applications, D. Stojanovic (Ed.), IGI Global, 2009, Chapter 3, pp. 59-82.
- [Daniel, 2010] Daniel, F.: Context-Aware Web Applications - The Model-Driven Way. VDM Verlag Dr. Müller, 2010.
- [Estrin et al., 2002] Estrin, D., Culler, D., Pister, K. and Sukhatme, G.: Connecting the Physical World with Pervasive Networks. *Journal of IEEE Pervasive Computing*, 2002, Vol. 01, N° 1, pp. 59-69.
- [Firmenich et al., 2010] Firmenich, S., Rossi, G., Urbietta, M., Gordillo, S.E., Challiol, C., Nanard, J., Nanard, M. and Araújo, J.: Engineering Concern-Sensitive Navigation Structures, Concepts, Tools and Examples. In *Journal of Web Engineering*, 2010, Vol. 9, N° 2, pp. 157-185.
- [Fuentes and Vallecillo, 2004] Fuentes, L. and Vallecillo, A.: Una Introducción a los Perfiles UML. *Novática: Revista de la Asociación de Técnicos de Informática*, 2004, N° 168, pp. 6-11.
- [Gama et al., 1995] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns*. Addison-Wesley Professional, 1995.
- [Gaedke and Meinecke, 2008] Gaedke, M. and Meinecke, J.: The Web as an Application Platform. In *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, O. Pastor, D. Schwabe and L. Olsina (Eds.) Springer, 2008, Chapter 3, pp. 33-45.
- [Gasevic et al., 2006] Gasevic, D., Djuric, D. and Devedzic, V.: The Model Driven Architecture (MDA). In *Model Driven Architecture and Ontology Development*, Springer Berlin-Heidelberg, 2006, Chapter 4, pp. 109-126.
- [Gherbi et al., 2009] Gherbi, T., Meslati, D. and Borne, I.: MDE between Promises and Challenges. In *Proceedings of UKSim 2009: 11th International Conference on Computer Modelling and Simulation*, 2009, pp.152-155.
- [Gronbaek et al., 2003] Gronbaek, K., Kristensen, J. and Eriksen, M.: Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material. In *Proceedings of the Hypertext 2003*, ACM Press, Nottingham, UK, pp. 10-19.
- [Hafedh et al., 2004] Hafedh, M., Amel, E. and Hamid, M.: Understanding Separation of Concerns. In *Proceedings of Aspect-Oriented Software Development (AOSD) 2004*, pp. 76-85.
- [Hailpern and Tarr, 2006] Hailpern, B. and Tarr, P.L.: Model-driven development: The good, the bad, and the ugly. *Journal of IBM Systems Journal*, 2006, Vol. 45 N°3, pp. 451-462.
- [Hansen and Bouvin, 2009] Hansen, F.A. and Bouvin, N.O.: Mobile Learning in Context - Context-aware Hypermedia in the Wild. *International Journal of Interactive Mobile Technologies*, 2009, Vol. 3, N°1, pp. 6-21.
- [Hansen et al., 2004] Hansen, F., Bouvin, N., Christensen, B., Gronbaek, K., Pedersen T. and Gagach, J.: Integrating the Web and the World: Contextual Trails on the Move. In *Proceedings of the 15th. ACM International Conference of Hypertext and Hypermedia (Hypertext 2004)*, ACM Press, 2004, pp. 98-107.
- [Hansen et al., 2008] Hansen, F.A., Kortbek, K.J. and Gronbæk, K.: Mobile Urban Drama - Setting the Stage with Location Based Technologies. In *Proceedings of First Joint International Conference on Interactive Digital Storytelling (ICIDS 2008)*, Springer Berlin / Heidelberg, 2008, pp. 20-31.
- [Hansen et al., 2010] Hansen, F.A., Kortbek, K.J. and Gronbæk, K.: Mobile Urban Drama for Multimedia-Based Out-of-School Learning. In *Proceedings of MUM'10 (the 9th International Conference on Mobile and Ubiquitous Multimedia)*, 2010, Article N° 17.
- [Harper et al., 2004] Harper, S., Goble, C. and Pettitt, S.: proximity: Walking the Link. In *Journal of Digital Information (JODI)*, British Computer Society and Oxford University Press, UK, 2004, Vol. 5, N° 1.
- [Kalkusch et al., 2002] Kalkusch, M., Lidy, T., Knapp, N., Reitmayr, G., Kaufmann, H. and Schmalstieg, D.: Structured visual markers for indoor pathfinding. In *Proceedings of the First IEEE International Workshop Augmented Reality Toolkit*, IEEE Press, 2002.

- [Kent, 2002] Kent, S.: Model Driven Engineering. In IFM 2002, M. Butler, L. Petre, and K. Sere (Eds.), LNCS 2335, 2002, pp. 286–298.
- [Koch et al., 2008a] Koch, N., Knapp, A., Zhang, G. and Baumeister, H.: UML-based Web Engineering: An Approach based on Standards. In *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, O. Pastor, D. Schwabe and L. Olsina (Eds.) Springer, 2008, Chapter 7, pp. 157-191.
- [Koch et al., 2008b] Koch, N., Meliá, S., Moreno, N., Pelechano, V., Sánchez, F. and Vara, J.M.: Model-Driven Web Engineering. In *UPGRADE: the European online Magazine for the IT Professional*, 2008, Vol. 9, Nº 2, pp.40-45.
- [Kolsch et al., 2006] Kolsch, M., Bane, R., Hollerer, T. and Turk, M.: Múltimodal interaction with a wearable augmented reality system. *Journal of Computer Graphics and Applications*, IEEE, 2006, Vol. 26, Issue 3, pp. 62-71.
- [Kolsch et al., 2009] Koch, N., Pigerl, M., Zhang G. and Morozova, T.: Patterns for the Model-based Development of RIAs. In *Proceedings of 9th Int. Conf. Web Engineering (ICWE'09)*, Springer, 2009, LNCS 5648, pp. 283-291.
- [Kraus et al., 2007] Kraus, A., Knapp, A. and Koch, N.: Model-Driven Generation of Web Applications in UWE. In *Proceedings of 3rd International Workshop on Model-Driven Web Engineering (MDWE 2007)*, CEUR-WS, 2007, Vol. 261, pp. 23-38.
- [Kroib and Koch, 2008] Kroib, C. and Koch, N.: UWE Metamodel and Profile: User Guide and Reference. LMU Technical Report, 2008.
- [Kroiss and Koch, 2009] Kroiss, C. and Koch, N.: UWE4JSF - A Model-Driven Generation Approach for Web Applications. In *Proceedings of 9th International Conference Web Engineering (ICWE'09)*, Springer, 2009, LNCS 5648, pp. 493-496.
- [Kuske et al., 2009] Kuske, S., Gogolla, M., Kreowski, H.J. and Ziemann, P.: Towards an integrated graph-based semantics for UML. In *Software and Systems Modeling*, Springer Berlin, 2009, Vol. 8, Nº 3, pp. 403-422.
- [Lehman, 1996] Lehman, M.M: Laws of software evolution revisited. In *Proceedings of the 5th European Workshop on Software Process Technology (EWSPT '96)*, Springer-Verlag, 1996, LNCS 1149, pp.108-124.
- [Liarokapis and Mountain, 2007a] Liarokapis, F. and Mountain, D.: A Mobile Framework for Tourist Guides. In *Proceedings of 8th International Symposium on Virtual Reality, Archaeology and Cultural Heritage, VAST, 2007*.
- [Liarokapis et al., 2006a] Liarokapis, F., Mountain, D., Papakonstantinou, S., Brujic-Okretic, V. and Raper, J.: Mixed reality for exploring urban environments. In *Proceedings of 1st International Conference on Computer Graphics Theory and Applications*, 2006, pp. 208-215.
- [Liarokapis et al., 2006b] Liarokapis, F., Raper, J. and Brujic-Okretic, V.: Navigating within the urban environment using Location and Orientation-based Services. In *Proceedings of the European Navigation Conference & Exhibition 2006*.
- [Liddle, 2010] Liddle, S.W.: Model-Driven Software Development. Report June 2010 for Data Extraction Research Group. <http://www.deg.byu.edu/papers/LiddleMDD.pdf> (Accedido: 2-2-2011).
- [Leonhardt, 1998] Leonhardt, U.: Supporting location-awareness in open distributed systems. Ph.D. Thesis, Dept. of Computing, Imperial College London, 1998.
- [Lyytinen and Yoo, 2002] Lyytinen, K. and Yoo, Y.: Issues and challenges in ubiquitous computing: Introduction. In *Communications of ACM*, Vol. 45, Nº 12, 2002, pp. 62-65.
- [Mackay, 1998] Mackay, W.E.: Augmented Reality: Linking real and virtual worlds. A new paradigm for interacting with computers. In *Proceedings of the ACM Conference on Advanced Visual Interfaces (AVI 98)*, 1998, ACM Press, pp. 13-21.
- [Milgram and Kishino, 1994] Milgram, P. and Kishino, F.: A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*, 1994, Vol. E77-D, Nº 12, pp. 1321-1329.

- [Millard et al., 2002] Millard, D.E., Michaelides, D.T., De Roure, D.C and Weal, M.J.: Beyond the Traditional Domains of Hypermedia. In Proceedings of Workshop on Open Hypermedia Systems, 2002, pp. 26-32.
- [Millard et al., 2004] Millard, D.E., De Roure, D.C., Michaelides, D.T., Thompson, M.K. and Weal, M.J.: Navigational hypertext models For physical hypermedia environments. In Proceedings of the Fifteenth ACM Conference on Hypertext and Hypermedia (HYPERTEXT '04), ACM Press, pp. 110-111.
- [Moreira et al., 2005] Moreira, A., Araujo, J. and Rashid, A.: A Concern-oriented Requirement Engineering Model. In Proceedings of Conference on Advanced Information Systems Engineering (CAISE'05), Springer, 2005, LNCS 3520, pp. 293-308.
- [Moreno et al., 2008a] Moreno, N., Meliá, S., Koch, N. and Vallecillo, A.: Addressing New Concerns in Model-Driven Web Engineering Approaches. In Proceedings of 9th International Conference Web Information Systems Engineering (WISE 2008), Springer, 2008, LNCS 5175, pp. 420-434.
- [Moreno et al., 2008b] Moreno, N., Romero, J. R. and Vallecillo, A.: An overview of Model-Driven Web Engineering and the MDA. In Web Engineering: Modelling and Implementing Web Applications, G. Rossi, O. Pastor, D. Schwabe and L. Olsina (Eds.), Springer, 2008, Chapter 12, pp. 353-382.
- [Mountain and Liarokapis, 2007b] Mountain, D. and Liarokapis, F.: Mixed reality (MR) interfaces for mobile information systems. Journal of Aslib Proceedings, Special issue: UK library & information schools, Emerald Press, 2007, Vol. 59 Issue 4/5 pp. 422-436.
- [Nanard et al., 2008] Nanard, J., Rossi, G., Nanard M., Gordillo, S.E. and Perez, L.: Concern-Sensitive Navigation: Improving Navigation in Web Software through Separation of Concerns. In Proceedings of 20th International Conference Advanced Information Systems Engineering (CAiSE 2008), Springer, 2008, LNCS 5074, pp. 420-434.
- [Nazim et al., 2006] Nazim, H. Madhavji, N.H., Fernandez-Ramil, J. and Perry, D.: Software Evolution and Feedback: Theory and Practice, John Wiley & Sons, 2006.
- [Ossher and Tarr, 2000] Ossher, H. and Tarr, P.: Multi-Dimensional Separation of Concerns and the Hyperspace Approach. In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development, 2000, pp. 293-323.
- [Parnas, 1972] Parnas, D.L.: On the criteria to be used in decomposing systems into modules. In Communications of the ACM, 1972, Vol. 15, Issue 12, pp. 1053-1058.
- [Parunak, 1989] Parunak, H.V.D.: Hypermedia topologies and user navigation. In Proceedings of the second annual ACM conference on Hypertext (HYPERTEXT '89), ACM, 1989, pp. 43-50.
- [Reitmayr and Schmalstieg, 2003a] Reitmayr, G. and Schmalstieg, D.: Data Management Strategies for Mobile Augmented Reality. In Proceedings of Workshop Software Technology for Augmented Reality Systems, IEEE CS Press, 2003, pp. 47-52.
- [Reitmayr and Schmalstieg, 2003b] Reitmayr, G. and Schmalstieg, D: Location Based Applications for Mobile Augmented Reality. In Proceedings of Australasian User Interface Conf. (AUIC), IEEE CS Press, 2003, pp. 65-73.
- [Reitmayr and Schmalstieg, 2004] Reitmayr, G. and Schmalstieg, D.: Collaborative Augmented Reality for Outdoor Navigation and Information Browsing. In Proceedings of Symposium Location Based Services and TeleCartography 2004, pp. 31-41.
- [Romero and Correia, 2003] Romero, L. and Correia, N.: HyperReal: A Hypermedia model for Mixed Reality. In Proceedings of the 14th ACM International Conference of Hypertext and Hypermedia (Hypertext 2003), ACM Press, 2003, pp. 2-9.
- [Rossi and Schwabe, 2008] Rossi, G. and Schwabe D.: Modeling and Implementing Web Applications with OOHDM. In Web Engineering: Modelling and Implementing Web Applications, G. Rossi, O. Pastor, D. Schwabe and L. Olsina (Eds.), Springer, 2008, Chapter 6, pp. 109-155.

- [Roy et al., 2003] Roy, N., Scheepers, H. and Kendall, E.: Mapping the Road for Mobile Systems Development. In Proceedings of Pacific Asia Conference on Information Systems 2003 (PACIS 2003), paper 94, pp. 1358-1371.
- [Ruiz-González et al., 2009] Ruiz-González, D., Koch, N., Kroiss, C., Romero, J.R. and Vallecillo, A.: Viewpoint Synchronization of UWE Models. In Proceeding of 5th International Workshop on Model-Driven Web Engineering (MDWE 2009), 2009, Vol. 455, pp. 46-60.
- [Schmalstieg and Reitmayr, 2005] Schmalstieg, D. and Reitmayr, G.: The world as a user interface: augmented reality for ubiquitous computing. In Proceedings of Central European Multimedia and Virtual Reality Conf., 2005.
- [Schmalstieg et al., 2007] Schmalstieg, D., Schall, G., Wagner, D., Barakonyi, I., Reitmayr, G., Newman, J. and Ledermann, F.: Managing Complex Augmented Reality Models, Journal of IEEE Computer Graphics and Applications, 2007, Vol. 27, N° 4, pp. 48-57.
- [Schwinger et al., 2008] Schwinger, W., Retschitzegger, W., Schauerhuber, A., Kappel, G., Wimmer, M., Pröll, B., Cachero Castro, C., Casteleyn, S., De Troyer, O., Fraternali, P., Garrigos, I., Garzotto, F., Ginige, A., Houben, G., Koch, N., Moreno, N., Pastor, O., Paolini, P., Pelechano Ferragud, V., Rossi, G., Schwabe, D., Tisi, M., Vallecillo, A., van der Sluijs, K. and Zhang, G.: A survey on web modeling approaches for ubiquitous web applications, International Journal of Web Information Systems, 2008, Vol. 4, N° 3, pp. 234 – 305.
- [Singh and Sood, 2009] Singh, Y. and Sood, M.: Model Driven Architecture: A Perspective. In Proceedings of IEEE International Advance Computing Conference (IACC 2009), 2009, pp.1644-1652.
- [Sutton and Rouvellou, 2002] Sutton, S.M. and Rouvellou, I.: Modeling of Software Concerns in Cosmos. In Proceedings of the 1st international conference on Aspect-Oriented Software Development (AOSD '02), ACM Press, 2002, pp. 127-133.
- [Talukder and Yavagal, 2006] Talukder, A.K. and Yavagal, R.: Mobile Computing: Technology, Applications, and Service Creation. McGraw-Hill Professional. 2006.
- [Tarr et al., 1999] Tarr, P., Oshser, H., Harrison, W. and Sutton, S.M.: N Degrees of Separation Multi-Dimensional Separation of Concerns. In Proceedings of the 21st international conference on Software engineering, ACM Press, 1999, pp. 107-119.
- [Tesoriero, 2009] Tesoriero, R.: CAUCE: Model-driven Development of context-aware applications for ubiquitous computing environments. PhD thesis, University of Castilla-La Mancha, Spain, December 2009.
- [Tesoriero et al., 2010] Tesoriero, R., Gallud, J.A., Lozano, M.D. and Penichet, V.M.R.: CAUCE: Model-driven Development of Context-aware Applications for Ubiquitous Computing Environments. In Journal of Universal Computer Science, 2010, Vol. 16, N° 15, pp. 2111-2138.
- [Torlone et al., 2006] Torlone, R., Barbieri, T., Bertini, E., Bianchi, A., Billi, M., Bolchini, D., Bruna, S., Burzagli, L., Cal'i, A., Catarci, T., Ceri, S., Daniel, F., De Virgilio, R., Facca, F., Gabbanini, F., Gabrielli, S., Giunta, G., Graziani, P., Kimani, S., Legnani, M., Mainetti, L., Matera, M., Palchetti, E., Presenza, D., Santucci, G., Sbattella, L. and Simeoni, N.: Methods and Tools for the Development of Adaptive Applications. In Mobile Information Systems, B. Pernici (Ed.), Springer Berlin Heidelberg, 2006, Chapter 8, pp. 209-247.
- [UML Specification] OMG Unified Modeling Language™ (OMG UML), Specification. Version 1.5, March 2003. <http://www.omg.org/spec/UML/1.5/PDF>.
- [UML Superstructure] OMG Unified Modeling Language™ (OMG UML), Superstructure. Version 2.2, February 2009. <http://www.omg.org/spec/UML/2.2/Superstructure/PDF>.
- [Varró et al., 2002] Varró, D., Varró, G. and Pataricza, A.: Designing the automatic transformation of visual languages. Journal Science of Computer Programming - Special issue on applications of graph transformations, Elsevier, 2002, Vol.44, N° 2, pp. 205-227.

ANEXO A

CARACTERÍSTICAS BÁSICAS DE *MAGICMOBILEUWE*

MagicDraw es una herramienta de modelado *UML* que tiene disponible una *API* abierta que permite extender su funcionalidad. Esta *API* permite por ejemplo acceder a las clases del metamodelo de *UML*, crear nuevos diagramas a partir de los existentes y crear nuevos elementos tanto a nivel de menús como en la barra de herramienta.

La forma de extender la funcionalidad de *MagicDraw* es creando un plugin que defina la nueva funcionalidad a incorporar. Cada vez que se inicia *MagicDraw* se levantan todos los plugin definidos incorporando las nuevas funcionalidades de cada uno. Para indicar que hay un nuevo plugin se debe crear dentro del directorio "`<MagicDraw_Install_Dir>\plugin`" un subdirectorio, que en nuestro caso llamaremos *MagicMobileUWE*, que contenga un archivo descriptor y un archivo `.jar` (con las clases del plugin).

El archivo descriptor debe tener el nombre `plugin.xml` y debe especificar cuál es la clase que es punto de entrada del plugin entre otras características. En este descriptor se debe indicar el nombre del archivo `.jar` asociado al plugin. En el Código A1 se puede apreciar la especificación de nuestro archivo descriptor. Este archivo es leído cada vez que se inicia *MagicDraw* y según lo que se haya definido en el plugin incorpora esta funcionalidad a la herramienta *MagicDraw*.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="MobileUWE"
  name="MobileUWE"
  version="1.0"
  class="magicMobileUWE.core.MobileUWEPluginManager">
  <requires>
    <api version="1.0"/>
  </requires>
  <runtime>
    <library name="MagicMobileUWE.jar"/>
  </runtime>
</plugin>
```

Código A.1: Archivo `plugin.xml`.

Se puede apreciar que `MagicMobileUWE.jar` representa al archivo `.jar` asociado al plugin y `magicMobileUWE.core.MobileUWEPluginManager` es la clase que es punto de entrada del plugin.

El archivo `MagicMobileUWE.jar` contiene todas las clases compiladas de nuestro plugin, en particular tiene la clase que es punto de entrada `magicMobileUWE.core.MobileUWEPluginManager`. La clase que es punto de entrada tiene que extender de la clase `com.nomagic.magicdraw.plugins.Plugin` (clase definida en la *API* abierta de *MagicDraw*) e implementar el método `init()`.

Cuando se inicia *MagicDraw* se leen todos los archivos con el nombre `plugin.xml` que están dentro del directorio *plugin* y se ejecuta de cada archivo el método `init()` de la clase especificada como punto de entrada. Según lo que se haya especificado en este método es la nueva funcionalidad que se agrega a la herramienta.

MagicDraw está escrito en *Java* por lo tanto requiere que el plugin también sea desarrollado usando *Java*. Para crear el plugin se usó *Eclipse* como ambiente de desarrollo.

Veamos los elementos involucrados en la *API* abierta que permiten que cada plugin¹²⁴ agregue funcionalidad específica a *MagicDraw*. La *API* abierta cuenta con la clase *ActionsConfiguratioManager* que permite agregar funcionalidad al pluign. La *API* provee una forma de recuperar un objeto de esta clase y a partir del mismo se agrega las nuevas características que se quieren incorporar a *MagicDraw*.

La *API* provee el mecanismo de acciones para incorporar nueva funcionalidad. Las nuevas acciones que se agregan deben extender de la clase *MDAction* provista por la *API* (o de alguna de sus subclases como *DefaultBrowserAction*, *PropertyAction* o *DefaultDiagramAction*). Estas acciones son invocadas mediante alguna interacción del diseñador con la pantalla de *MagicDraw*. El diseñador visualiza en la pantalla de *MagicDraw* el nombre asignado a la acción al seleccionar el elemento asociado a una acción, dicha acción se ejecuta.

Cada una de estas acciones creadas debe ser agregada a un objeto de la clase *ActionsCategory*, este objeto permite agrupar acciones para luego ser usadas en conjunto o de forma agrupadas, por ejemplo, para un menú. De esta manera las acciones quedan agrupadas por categorías.

A su vez las categorías deben ser agregadas a una clase llamada *ActionsManager*, que permite manejar las diferentes categorías. Hay diferentes *ActionsManager* cada uno representa un elemento de la pantalla (de *MagicDraw*) como, por ejemplo, el menú principal, el menú contextual o la barra de herramientas. Estos manejadores son configurados para determinar bajo que circunstancias se ejecutan esas acciones o cuando las mismas están visualmente disponibles. Para realizar esta configuración la *API* provee tres interfaces específicas: la primera *AMConfigurator* que se utiliza para propósito general (por ejemplo, para menús o la barra de herramientas), la segunda *BrowserContextAMConfugurator* específica para los menús contextuales del navegador y por último *DiagramContextAMConfigurator* específica para los menús contextuales de los diagramas.

Estos son los elementos involucrados para realizar el plugin, se tienen acciones las cuales se configuran para que estén disponibles al diseñador en determinados momentos y cuando éstas son seleccionada se realiza la acción definida.

Como se mencionó anteriormente el método `init()` de la clase `magicMobileUWE.core.MobileUWEPluginManager` es el encargado de especificar todas la funcionalidad del plugin, por lo tanto su contenido debe definirse acorde a las acciones y configuraciones mencionadas. Analicemos por partes cómo está definido este método. En la Código A.2 se puede apreciar que el método realiza una inicialización, en la cual se levanta nuestro perfil llamado *Mobile Hypermedia* (éste a su vez importa nuestros perfiles *Mobile UWE*, *Location*, *Layout* y *User Model*)¹²⁵. Luego el método utiliza la *API* para recuperar el objeto de la clase *ActionsConfiguratioManager* al cual se le va agregando las distintas acciones. Además se crea un listener para escuchar los eventos que genera el usuario cuando interactúa con la pantalla de *MagicDraw*.

```
public void init() {
    MobileUWEPropertyLoader.initConstants();

    // Initialize project listener
    ProjectListener projectListener = new ProjectListener();
    Application.getInstance().addProjectEventListener(projectListener);
    ActionsConfiguratorsManager manager = ActionsConfiguratorsManager.getInstance();
}
```

Código A.2: Método `init()` - Realizar la carga de nuestro perfil.

¹²⁴ Los detalles referentes a los elementos definidos por *UWE* para su plugin *MagicUWE* están especificados en la tesis llamada “*MagicDraw-Plugin zur Modellierung und Generierung von Web-Anwendungen*” de Petar Blagoev, 2007.

http://tmi.pst.ifi.lmu.de/thesis_results/0000/0021/DA_PetarBlagoev.pdf

¹²⁵ Todos nuestros perfiles se deben copiar en la carpeta *profiles* de la herramienta *MagicDraw*.

Veamos ahora cómo (en el método `init()`) se realiza la creación de las acciones asociadas al menú principal y la configuración de las mismas. En la Código A.3 se puede apreciar que se crean las cinco opciones que se visualizan al desplegar el menú principal. Al objeto de la clase *ActionsConfiguratioManager* se le agregan cinco configuraciones. Estas configuraciones agregadas son de la clase *MagicMobileUWEMenuConfigurator*, la cual implementa la interfaz *AMConfigurator*.

```
// Adding Mobile UWE - Submenus
manager.addMainMenuConfigurator(new MagicMobileUWEMenuConfigurator(getNewMobileUWEDiagramsSubmenuActions()));
manager.addMainMenuConfigurator(new MagicMobileUWEMenuConfigurator(getMobileUWETransformatorSubmenuActions()));

// Adding Additional Mobile UWE - Submenus
manager.addMainMenuConfigurator(new MagicMobileUWEMenuConfigurator(getAdditionalMobileUWEDiagramsSubmenuActions()));
manager.addMainMenuConfigurator(new MagicMobileUWEMenuConfigurator(getAdditionalOperationMobileUWEActions()));

// Add About MagicMobileUWE menu
manager.addMainMenuConfigurator(new MagicMobileUWEMenuConfigurator(getAboutMobileUweAction()));
```

Código A.3: Método `init()` - Creación del menú principal.

Se puede ver (en el Código A.3) que cuando se crea cada configuración se pasa como parámetros del constructor las acciones asociadas a cada submenú. Veamos que representa cada uno de los mensajes invocados:

- *getNewMobileUWEDiagramsSubmenuActions* devuelve una categoría de acciones la cual contiene una colección de acciones para crear los Modelos de Contenido, Unificado y Navegacional. Estas acciones se crean a partir de la clase *MobileUWENewDiagramMenuAction* (que extiende de la acción *MDAction*). Cuando el diseñador elige una de las acciones de este menú se ejecuta la acción y se crea el diagrama correspondiente.
- *getMobileUWETransformatorSubmenuActions* devuelve una categoría de acciones la cual contiene una colección de acciones que permiten realizar cada una de nuestras transformaciones. Estas acciones se crean usando la clase *MobileUWETransformatorAction* (que extiende de la acción *MDAction*). Cuando el diseñador elige realizar una de las transformaciones de este menú, se ejecuta la acción y se deriva el modelo correspondiente acorde a la transformación elegida.
- *getAdditionalMobileUWEDiagramsSubmenuActions* devuelve una categoría de acciones, la cual contiene una colección de acciones para crear los Modelos de Instancias Navegacional, de Presentación y del Usuario. Estas acciones extienden de la acción *MDAction* y se crean a partir de instanciar las clases *MobileUWENewDiagramMenuAction*, *MobileUWECreateLayoutDiagramAction* y *MobileUWECreateUserModelDiagramAction* respectivamente. Cuando el diseñador elige alguna de estas acciones listadas en este submenú se crea el modelo correspondiente.
- *getAdditionalOperationMobileUWEActions* devuelve una categoría de acciones la cual contiene una colección de acciones relacionadas a operaciones que puede realizar el plugin como, por ejemplo, la generación de código.
- *getAboutMobileUweAction* devuelve una categoría que contiene una sola acción llamada *AboutMobileUweAction* (que extiende de la acción *MDAction*) y cuando selecciona esta opción el diseñador recibe una venta con la información de la herramienta.

En el método `init()` también se crean las acciones relacionadas a la barra de herramienta. En la Código A.4 se puede apreciar cómo se agrega al objeto *ActionsConfiguratioManager* las configuraciones relacionadas con cada elemento en la barra de herramienta. Se puede apreciar que cada configuración define para qué

diagrama se va a usar el elemento en cuestión. Todos los elementos que agregamos a la barra de herramienta son para los diagramas de clases de UML (`DiagramTypeConstants.UML_CLASS_DIAGRAM`). Además se pasa como parámetro de cada configuración una instancia del elemento *MobileUWEToolbarConfigurator* (extiende de la clase *BaseDiagramToolbarConfigurator* de la API), esta instancia se construye indicando las acciones que debe realizar el elemento de la barra y sobre que modelo es adecuado dibujar usar dicho elemento.

```
// Adding Mobile UWE diagram toolbar actions
// Content action
manager.addDiagramToolbarConfigurator(DiagramTypeConstants.UML_CLASS_DIAGRAM, new MobileUWEToolbarConfigurator(
    getDiagramToolbarMobileUWEContentPackageActions(), MobileUWEDiagramType.MOBILEUWECONTENT.toString());
// Unified action
manager.addDiagramToolbarConfigurator(DiagramTypeConstants.UML_CLASS_DIAGRAM, new MobileUWEToolbarConfigurator(
    getDiagramToolbarMobileUWEUnifiedAssociationActions(), MobileUWEDiagramType.MOBILEUWEUNIFIED.toString());
// Navigational actions
manager.addDiagramToolbarConfigurator(DiagramTypeConstants.UML_CLASS_DIAGRAM, new MobileUWEToolbarConfigurator(
    getDiagramToolbarMobileUWENavigationAssociationRoleOfActions(), MobileUWEDiagramType.MOBILEUWENAVIGATION.toString());
manager.addDiagramToolbarConfigurator(DiagramTypeConstants.UML_CLASS_DIAGRAM, new MobileUWEToolbarConfigurator(
    getDiagramToolbarMobileUWENavigationAssociationActions(), MobileUWEDiagramType.MOBILEUWENAVIGATION.toString());
manager.addDiagramToolbarConfigurator(DiagramTypeConstants.UML_CLASS_DIAGRAM, new MobileUWEToolbarConfigurator(
    getDiagramToolbarMobileUWENavigationClassActions(), MobileUWEDiagramType.MOBILEUWENAVIGATION.toString());
manager.addDiagramToolbarConfigurator(DiagramTypeConstants.UML_CLASS_DIAGRAM, new MobileUWEToolbarConfigurator(
    getDiagramToolbarMobileUWENavigationPackageActions(), MobileUWEDiagramType.MOBILEUWENAVIGATION.toString());
```

Código A.4: Método `init()` - Creación de los elementos de la barra de herramientas.

Veamos las acciones (que se le pasan como parámetro al constructor *MobileUWEToolbarConfigurator* en el Código A.4) con las que se configura cada elemento de la barra de herramientas.

- `getDiagramToolbarMobileUWEContentPackageActions` devuelve una categoría que contiene dos acciones para dibujar los dos concerns relacionados al Modelo de Contenido. Estas acciones se crean instanciando la clase llamada *MobileUWEDrawPackageAction* que extiende de la acción *DrawShapeDiagramAction* de la API, como parámetro se pasa el estereotipo con el cual se quiere crear el paquete en cuestión. Lo mismo ocurre con la invocación del método `getDiagramToolbarMobileUWENavigationPackageActions()` para dibujar los dos concerns relacionados con el Modelo Navegacional.
- `getDiagramToolbarMobileUWEUnifiedAssociationActions` devuelve una categoría que contiene una sola acción para dibujar la relación de rol. Esta acción se crea a partir de la clase *MobileUWEDrawAssociationAction* (extiende de la acción *DrawPathDiagramAction* de la API) como parámetro se pasa el estereotipo con el cual se quiere crear la asociación. Esta misma clase (*MobileUWEDrawAssociationAction*) se invoca desde `getDiagramToolbarMobileUWENavigationAssociationActions()` para crear las asociaciones que representan los links caminable, tanto sea el link caminable direccional como el bidireccional. Lo mismo ocurre con el método `getDiagramToolbarMobileUWENavigationAssociationRoleOfActions()` que termina llamando a la clase *MobileUWEDrawAssociationAction* para dibujar la asociación de rol en el Modelo Navegacional.
- `getDiagramToolbarMobileUWENavigationClassActions` devuelve una categoría que contiene una sola acción para dibujar el nodo físico. Usando la clase *MobileUWEDrawClassAction* y pasándole el estereotipo correspondiente se dibuja dicho nodo.

Para facilitar la explicación se menciona directamente la acción que se devuelve en algunos métodos sin mencionar detalles de código de cada uno. El Código A.5 muestra cómo agregar las configuraciones relacionadas con los menús contextuales de las clases del concerns físico para crear la propiedad de ubicación. Al objeto de la clase *ActionsConfiguratioManager* se le agregan dos configuraciones. La primera

configuración usa la clase *MobileUWEPhysicalContentPropertyContextConfigurator* (extiende de la clase de la API *ClassDiagramContextAMConfigurator*). La cual tiene una acción (*MobileUWESetPropertyAsLocationInContentClassAction*) que permite setear una propiedad como de ubicación. La segunda configuración usa la clase *MobileUWEPhysicalContentClassContextConfigurator* (extiende también de la clase de la API *ClassDiagramContextAMConfigurator*) y tiene asociada la acción *MobileUWECreateLocationPropertyInContentClassAction*, que permite crear una propiedad de ubicación.

```
// Add Set Location sub-context-menus (Only Mobile UWE Content Diagram)
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
    new MobileUWEPhysicalContentPropertyContextConfigurator (
        getContentDiagramSetLocationContextInsertMainMenuActions (),
        MobileUWEDiagramType.MOBILEUWECONTENT));
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
    new MobileUWEPhysicalContentClassContextConfigurator (
        getContentDiagramCreateLocationPropertyContextInsertMainMenuActions (),
        MobileUWEDiagramType.MOBILEUWECONTENT));
```

Código A.5: Método `init()` - Menús contextuales de las clases del concern físico.

Notar que se indica para qué tipo de diagrama es cada configuración, en el caso del Código A.5 es para los Modelos de Contenidos.

En la Código A.6 se puede apreciar la configuración relacionada con la operación para crear los links caminables derivados en el Modelo Navegacional. La configuración se crea usando la clase *MobileUWEPhysicalNavigationPhysicalNodeContextConfigurator* (extiende de la clase *ClassDiagramContextAMConfigurator* de la API) y tiene asociada la acción *MobileUWECreateDerivedWalkingLinkOperationInPhysicalNodeAction*.

```
// Add Create an Operation (Only Mobile UWE Navigation Diagram)
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
    new MobileUWEPhysicalNavigationPhysicalNodeContextConfigurator (
        getNavigationDiagramCreateDerivedWalkingLinkOperationPhysicalNodeContextInsertMainMenuActions (),
        MobileUWEDiagramType.MOBILEUWENAVIGATION));
```

Código A.6: Método `init()` - Operación de links caminables derivados.

En la Código A.7 se puede apreciar las configuraciones relacionada con la creación de las instancias de los nodos del Modelo de Instancias Navegacional. Las configuraciones que se crean son usando las clases *MobileUWEInstanceDiagramDigitalConcernContextConfigurator* y *MobileUWEInstanceDiagramPhysicalConcernContextConfigurator* (ambas extienden de la clase *DiagramContextAMConfigurator* de la API). Ambas configuraciones usan la acción *MobileUWEInstanceDiagramClassAction* (extiende de *DrawShapeDiagramAction* de la API) con el estereotipo adecuado según corresponda a una instancia de un nodo digital o físico.

```
// Add main instance navigation context-menu (Create instances navigationClass)
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
    new MobileUWEInstanceDiagramDigitalConcernContextConfigurator (
        getInstanceDiagramDigitalConcernContextInsertMainMenuActions (),
        MobileUWEInstanceDiagramType.MOBILEUWENAVIGATIONINSTANTIATION));
// Add main instance navigation context-menu (Create instances physicalNode)
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
    new MobileUWEInstanceDiagramPhysicalConcernContextConfigurator (
        getInstanceDiagramPhysicalConcernContextInsertMainMenuActions (),
        MobileUWEInstanceDiagramType.MOBILEUWENAVIGATIONINSTANTIATION));
```

Código A.7: Método `init()` - Creación instancias de los nodos.

Asociada a las instancias se tiene la posibilidad de definirles su nodo base. En el Código A.8 se puede ver cómo se realiza la configuración para proveer esta funcionalidad. La configuración que se define para el Modelo de Instancias Navegacional es mediante el uso de la clase *MobileUWEInstanceClassContextConfigurator* y la acción asociada es *MobileUWESetClassifierInInstanceClassAction*.

```
// Add Set Classifier sub-context-menus (Only Mobile UWE Instance Navigation Diagram)
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
new MobileUWEInstanceClassContextConfigurator (
getNavigationInstanceClassDiagramContextInsertMainMenuActions(),
MobileUWEInstanceDiagramType.MOBILEUWENAVIGATIONINSTANTLATION));
```

Código A.8: Método `init()` - Opción para definir el nodo base de la instancia.

Otra funcionalidad asociada al Modelo de Instancias Navegacional es completar dicho modelo. En el Código A.9 se puede visualizar la configuración de las dos operaciones de completar que provee la herramienta. Ambas operaciones utilizan la clase *MobileUWECompleteInstanceDiagramContextConfigurator* para especificar la configuración pero cada una tiene su acción asociada mediante las clases *MobileUWECompleteOnlyPropertiesAndRolesInstanceDiagramAction* y *MobileUWECompleteInstanceDiagramAction*. Estas acciones permiten completar el modelo en forma completa o sólo completar las propiedades de cada instancia junto con sus roles.

```
// Add main instance navigation context-menu (Complete Diagram)
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
new MobileUWECompleteInstanceDiagramContextConfigurator (
getNavigationInstanceCompleteOnlyPropertiesAndRolesInstanceDiagramaInsertMainMenuActions(),
MobileUWEInstanceDiagramType.MOBILEUWENAVIGATIONINSTANTLATION));
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
new MobileUWECompleteInstanceDiagramContextConfigurator (
getNavigationInstanceCompleteInstanceDiagramaInsertMainMenuActions(),
MobileUWEInstanceDiagramType.MOBILEUWENAVIGATIONINSTANTLATION));
```

Código A.9: Método `init()` - Completar el Modelo de Instancias Navegacional.

El Modelo de Presentación tiene como funcionalidad permitir la definición del estilo particular para cada clase. Esto se realiza con la configuración *MobileUWELayoutClassContextConfigurator* como se puede visualizar en la Código A.10. La acción asociada es *MobileUWESetStylesheetInLayoutClassAction*.

```
// Add Stylesheet sub-context-menus (Only Mobile UWE Layout Diagram)
manager.addDiagramContextConfigurator (DiagramTypeConstants.UML_CLASS_DIAGRAM,
new MobileUWELayoutClassContextConfigurator (
getLayoutDiagramSetStylesheetClassContextInsertMainMenuActions(),
MobileUWELayoutDiagramType.MOBILEUWELAYOUT));
```

Código A.10: Método `init()` - Definir el estilo de presentación.

La Código A.11 muestra las configuraciones relacionadas con los menús contextuales del navegador. Por un lado, se provee crear los modelos mediante la opción del navegador *New Diagram* y por otro lado las transformaciones posibles acordes a cada modelo. Se puede ver que la configuración usada es *MobileUWEBrowserContextConfigurator* la cual extiende de la clase *BrowserContextAMConfigurator* de la API. Las acciones usadas son las mismas que se definieron para el menú principal tanto para la creación de los diferentes modelos como para las transformaciones.

```
// Adding Mobile UWE Diagrams to containment browser (New Diagram - context menu)
manager.addContainmentBrowserContextConfigurator (new MobileUWEBrowserContextConfigurator (getMobileUWEDiagramsActions(),
ActionsID.NEW_DIAGRAM,null));

// Adding Mobile UWE transformation to containment browser of the specific models
manager.addContainmentBrowserContextConfigurator (new MobileUWEBrowserContextConfigurator (MobileUWEPluginManagerActions
.getTransformationMobileUWEContent2MobileUWEUnified(), null, MobileUWEInstanceDiagramType.MOBILEUWECONTENT));
manager.addContainmentBrowserContextConfigurator (new MobileUWEBrowserContextConfigurator (MobileUWEPluginManagerActions
.getTransformationMobileUWEContent2MobileUWEUnifiedSomeConcerns(), null, MobileUWEInstanceDiagramType.MOBILEUWECONTENT));
manager.addContainmentBrowserContextConfigurator (new MobileUWEBrowserContextConfigurator (MobileUWEPluginManagerActions
.getTransformationMobileUWEUnified2MobileUWENavigation(), null, MobileUWEInstanceDiagramType.MOBILEUWEUNIFIED));
```

Código A.11: Método `init()` - Opciones del menú de navegación.

Para poder invocar la funcionalidad de la API de *MagicDraw* el proyecto (Eclipse) con el código de nuestro plugin usa las librerías que contienen dicha API como son *md_api.jar* y *md_commond_api.jar*, para poder extender así la funcionalidad de la herramienta *MagicDraw*.

En la Figura A.1 se pueden observar todas las acciones creadas (las cuales se fueron detallando mientras se describía el método `init()`) y las clases de la API que extienden. En particular se extiende de tres clases de la API de *MagicDraw*:

- *MdAction*, para definir acciones asociadas, por ejemplo, a la creación de diagramas o realización de transformaciones.
- *DrawShapeDiagramAction*, para dibujar clases, paquetes o instancias.
- *DrawPathDiagramAction*, para dibujar asociaciones.

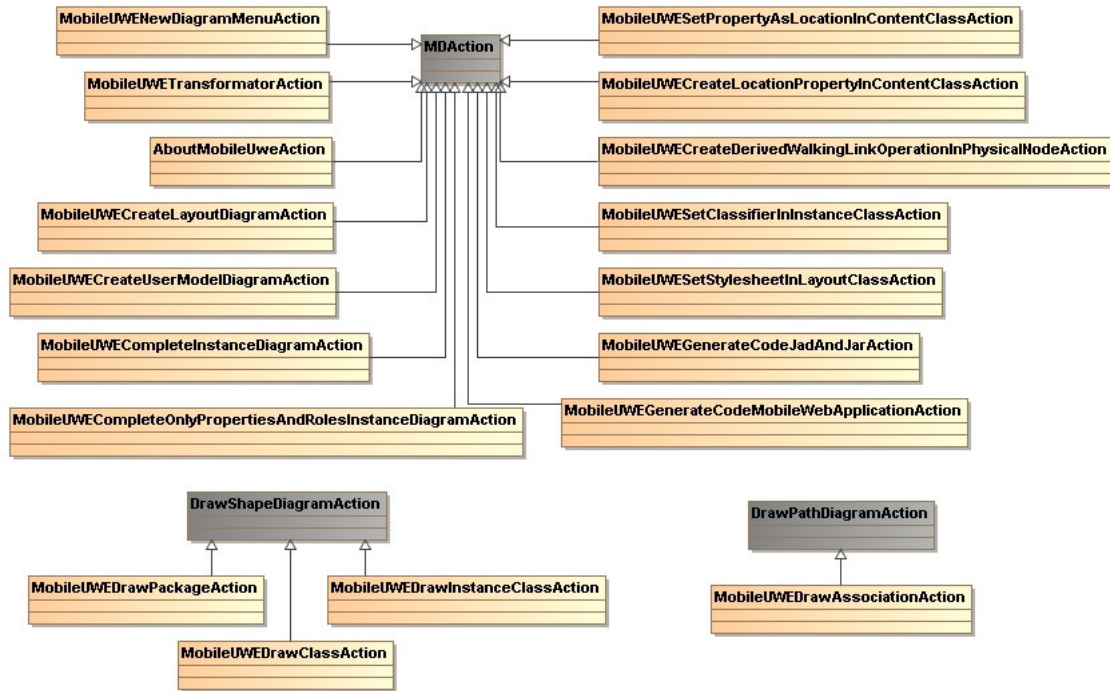


Figura A.1: Acciones creadas en la herramienta.

En la Figura A.2 quedan especificados todos los configuradores mencionados (mientras se describía el método `init()`). Se indica en la figura si implementan una interfaz de la API o si extienden alguna clase particular de la misma.

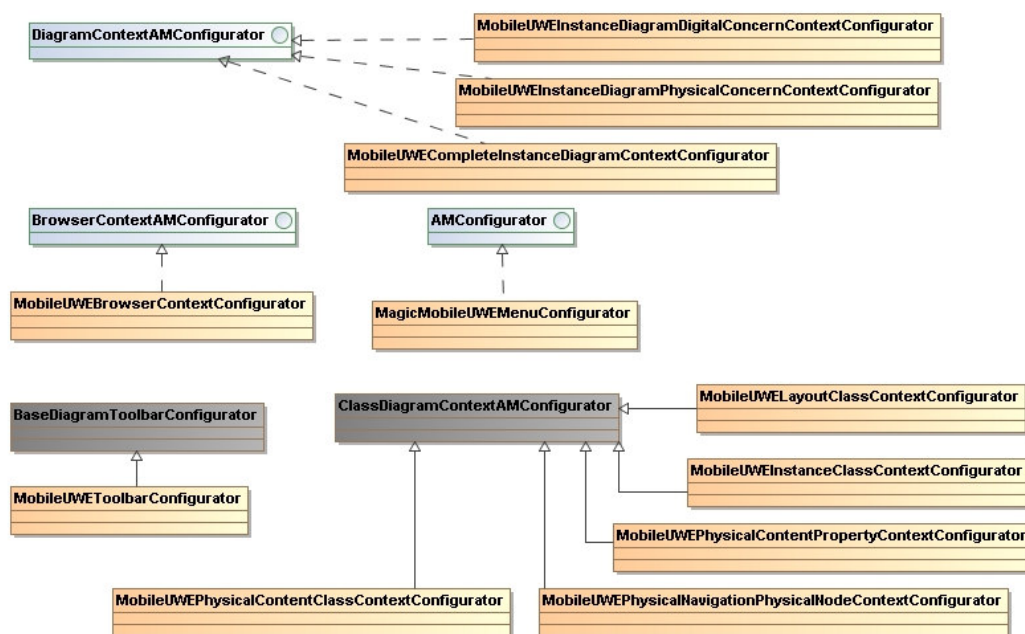


Figura A.2: Configuradores creados en la herramienta.

Para la representación de cada uno de nuestros modelos se crean cuatro nuevas clases como se puede observar en la Figura A.3. Se puede apreciar que todas las clases están definidas como enumerativos, esta decisión fue para mantener consistencia con la definición del plugin de *MagicUWE*. Para lograr que la implementación funcione de manera genérica se creó una interfaz general donde cada uno de los tipos de diagramas implementan dicha interfaz. Cada una de estas clases se encarga de realizar cuestiones específicas de cada modelo.

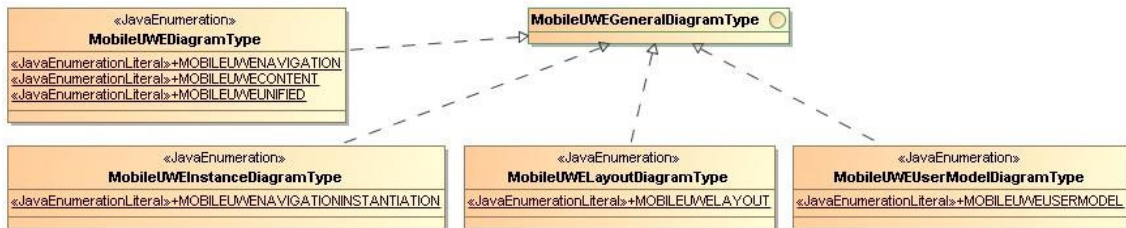


Figura A.3: Diagramas creados en la herramienta.

En la Figura A.4 se puede apreciar las clases creadas para representar nuestros estereotipos. Las clases están definidas como enumerativos y están agrupados depende a que elemento corresponde el estereotipo en cuestión. En particular tenemos los estereotipos de paquetes, clases y asociaciones. Se creó una interfaz para manejar los estereotipos dentro de las clases del plugin de manera general.

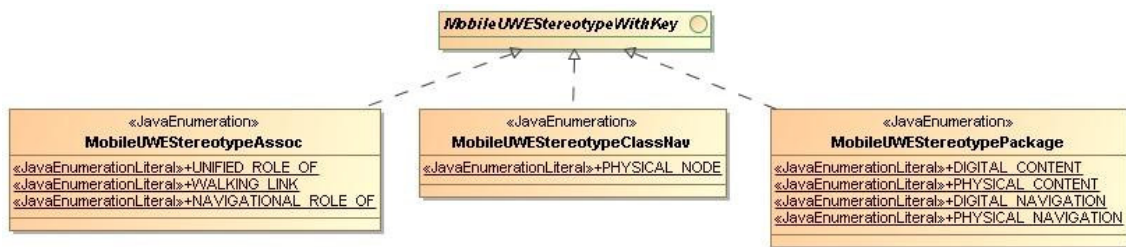


Figura A.4: Clases que representan los estereotipos usados la herramienta.

Para considerar las transformaciones que debe realizar nuestro plugin se definió la clase *MobileUWTransformationType* como un enumerativo con las tres posibles transformaciones como se puede visualizar en la Figura A.5. Se puede observar que estas transformaciones tienen como modelo origen y destino un elemento de la clase *MobileUWEDiagramType*. Es decir, un Modelo de Contenido, Unificado o Navegacional. La clase *MobileUWEDiagramTransformation* es una clase es un colaborador que desacopla funcionalidad común a todas las transformaciones.

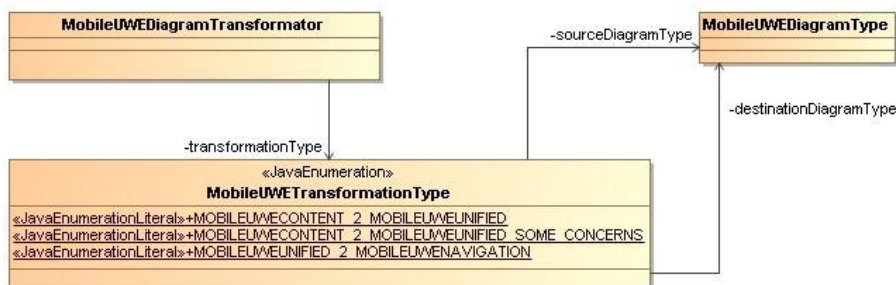


Figura A.5: Clases para representar las transformaciones en la herramienta.

ANEXO B

TRANSFORMACIONES DEFINIDAS EN *MAGICMOBILEUWE*

Las tres transformaciones que tiene definidas la herramienta (como se indicó en el Anexo A) están representadas con una clase llamada *MobileUWETransformationType*. Esta clase es un enumerativo donde cada uno de sus elementos representa cada una de las transformaciones. Los nombres de estos tres elementos son:

- MOBILEUWECONTENT_2_MOBILEUWEUNIFIED
- MOBILEUWECONTENT_2_MOBILEUWEUNIFIED_SOME_CONCERNS
- MOBILEUWEUNIFIED_2_MOBILEUWENAVIGATION

La clase *MobileUWETransformationType* define el constructor para estos elementos con tres parámetros. El primer parámetro indica el nombre que tendrá dicha transformación, el cual es usado para mostrar en la pantalla la transformación. El segundo parámetro especifica el tipo del modelo origen¹²⁶ (el cual se quiere transformar) y el tercer parámetro indica el tipo del modelo destino.

La clase *MobileUWETransformationType* define el método que comienza la transformación. Este método se denomina `transformation()` y recibe como parámetro el modelo a transformar y el paquete donde se debe crear el modelo producto de la transformación. Cada uno de los elementos de esta clase define este método según la característica de la transformación que representa. Para cada uno de los tres elementos definidos en la clase *MobileUWETransformationType* se invoca este método con los parámetros adecuados. Al invocar este método cada transformación crea el modelo adecuado con sus elementos y los estereotipos correspondientes a los mismos. En las siguientes subsecciones se muestra cómo se define este método para cada transformación de acuerdo a la naturaleza que tenga cada una.

Cabe mencionar que los modelos de *MagicDraw* representan sus elementos¹²⁷ en dos niveles. Un nivel de representación de los elementos propios de *UML* con las características definidas en su metamodelo. Y otra capa de visualización propia de *MagicDraw* donde se define cómo se muestra gráficamente cada elemento de *UML*. Asociados a un elemento *UML* puede haber varios elementos de visualización que sirven para mostrarlo. Por ejemplo para el elemento *Package* de *UML* se tienen los elementos *PackageView* (que define las dimensiones del paquete y conoce al elemento *Package* que se visualiza), *HeaderView* (que define las dimensiones del nombre del paquete y conoce el String que representa al nombre) entre otros elementos de visualización que pueden estar asociados. Sólo nos vamos a focalizar en el elemento de visualización que conocen al elemento *UML* que representa, para el ejemplo descrito sería *PackageView*.

Todos los elementos del metamodelo de *UML* son representados por *MagicDraw* mediante el uso de interfaces y además provee clases que implementan cada interfaz. Como nomenclatura las interfaces llevan el nombre del elemento *UML* que representan y las clases que implementan estas interfaces agregan al nombre el sufijo "Impl". A su vez todas las interfaces implementan la interfaz *Element*. Esta interfaz se usa en la definición de nuestras transformaciones como se muestra en este anexo según sea necesario para tratar los elementos de *UML* en forma genérica.

En la Tabla B.1 se puede visualizar cómo se representa la correspondencia de

¹²⁶ En el Anexo A se detallaron los tipos de modelos que tiene definida la herramienta.

¹²⁷ Para poder recuperar los elementos de un modelo este debe estar abierto en *MagicDraw*. Cuando se abre el modelo para visualizarlo, quedan disponibles sus elementos para ser manipulados por la herramienta.

algunos de los elementos que usamos en las transformaciones. Es decir, algunas de las principales interfaces de los elementos de *UML* con la correspondiente clase que la implementa. Además, la clase de visualización correspondiente que contiene (o conoce) el elemento *UML*. Todas las clases e interfaces presentadas en esta tabla son usadas en nuestras transformaciones.

Tabla B.1: Elementos de *UML* representados por *MagicDraw*.

Interfaz que representa a un elemento de <i>UML</i>	Clase que implementa la interfaz	Clases de visualización que contiene el elemento de <i>UML</i>
<i>Package</i>	<i>PackageImpl</i>	<i>PackageView</i>
<i>Class</i>	<i>ClassImpl</i>	<i>ClassView</i>
<i>Association</i>	<i>AssociationImpl</i>	<i>AssociationView</i>
<i>Generalization</i>	<i>GeneralizationImpl</i>	<i>GeneralizationView</i>
<i>InstanceSpecification</i> ¹²⁸	<i>InstanceSpecificationImpl</i>	<i>InstanceSpecificationView</i>

En las transformaciones que se muestran en las siguientes subsecciones se manejan cuestiones en ambos niveles (elementos de *UML* y de visualización), ya que hay que crear en el nuevo modelo tanto los elementos de *UML* como así también indicar cómo se visualizan.

B.1. Transformación del Modelo de Contenido al Modelo Unificado

En el Código B.1 se puede apreciar la definición de la transformación que realiza la creación de un Modelo Unificado a partir de un Modelo de Contenido. La transformación se crea pasando los tres parámetros correspondientes al constructor de la clase *MobileUWETransformationType*. El primer parámetro indica el nombre de la transformación, el segundo el tipo del modelo origen y el tercer parámetro indica el tipo del modelo destino de la transformación. Además se visualiza la definición de dos variables pertenecientes a esta transformación, una sirve para tener representada cual es el concern *Core* de la aplicación y la otra variable tiene todos los concerns digitales del Modelo de Contenido (que deben tener definidos un nombre para poderlos manipular).

```
MOBILEUWECONTENT_2_MOBILEUWEUNIFIED(MobileUWEGlobalConstants.getName_Content_To_Unified(),
    MobileUWEDiagramType.MOBILEUWECONTENT,
    MobileUWEDiagramType.MOBILEUWEUNIFIED) {
protected Package coreConcern;
protected List<Package> digitalConcerns = new ArrayList<Package>();
```

Código B.1: Definición de la transformación del Modelo de Contenido al Modelo Unificado.

Para usar esta transformación (Código B.1) basta con indicar lo siguiente: *MobileUWETransformationType.MOBILEUWECONTENT_2_MOBILEUWEUNIFIED*. Luego a este elemento se le invocan los métodos que tenga definido la clase *MobileUWETransformationType*. Como se mencionó anteriormente el método que comienza la transformación se denomina *transformation()*. A este método se le pasa como parámetro el Modelo de Contenido original y el paquete de destino donde se debe crear el nuevo Modelo Unificado que se creará con la transformación. En el Código B.2 se puede apreciar cómo esta transformación define este método¹²⁹.

¹²⁸ El elemento *InstanceSpecification* se utiliza para representar las instancias de los elementos de *UML*. Por ejemplo, se utiliza para representar las instancias particulares de los nodos en nuestro Modelo de Instancias Navegacional.

¹²⁹ Este método invoca a otros métodos internos de la transformación. Sólo se mostrará el código de aquellos métodos que sean de interés, del resto sólo se describirá la funcionalidad.

```

protected void transformation(DiagramPresentationElement sourceDiagram, Package destPackage) {
    HashMap<Element, List<PresentationElement>> elementsInDiagram =
        new HashMap<Element, List<PresentationElement>>();
    sourceDiagram.open();
    MobileUWEDiagramType.collectElementsAndPresentationElements(sourceDiagram, elementsInDiagram);
    resetDigitalConcerns();
    collectDigitalConcerns(elementsInDiagram);
    if (hasConcernsWithNames()){
        int index =
            (new MobileUWEGUISelector(getDigitalConcernsNames(),
                MobileUWEGUIGlobalConstants.getSELECT_CORE()).getIndex());
        if (index != -1) {
            setCoreConcern(index);
            Diagram destDiagram = MobileUWEDiagramType.MOBILEUWEUNIFIED.createAndAddDiagram(destPackage);
            setContentDiagram(sourceDiagram, destDiagram);
            setDiagramTransformator(
                new MobileUWEDiagramTransformator(sourceDiagram, destDiagram, destPackage, this));
            launchPackageTransformation(elementsInDiagram, destPackage);
            showDoneMessage(sourceDiagram.getName());
        }
    }else{
        MobileUWEMessageWriter.showMessage(MobileUWEGUIGlobalConstants.getNO_CONCERN(), getLogger());
    }
}
}

```

Código B.2: Método inicial invocado para realizar la transformación.

En el Código B.2 se puede apreciar que primero se crea la variable `elementsInDiagram` que servirá para contener todos los elementos del Modelo de Contenido origen. Luego se invoca el método `collectElementsAndPresentationElements()`¹³⁰ que recolecta todos los elementos de Modelo de Contenido origen y los guarda en la variable `elementsInDiagram`. La estructura guardada en `elementsInDiagram` contiene cada elemento *UML* del Modelo de Contenido junto a todos los elementos de visualización asociados al mismo. Previamente se había abierto (`sourceDiagram.open()`) el modelo para poder recuperar todos los elementos del mismo.

Siguiendo con el Código B.2, se puede ver que se resetea (vacía) la colección de concerns (`digitalConcerns`) que contienen el modelo para que no queden registros de concerns de alguna transformación anterior.

Luego se obtienen todos los nombres de los concerns digitales que tiene el Modelo de Contenido usando el método `collectDigitalConcerns()`¹³⁰ y se guardan en la variable de la transformación `digitalConcerns`.

Si hay concerns digitales se abre una ventana usando la clase *MobileUWEGUISelector*, la cual le muestra al diseñador todos nombres de los concerns digitales (mediante el método `getDigitalConcernsNames()`) para que éste seleccione uno como *Core*. Usando el índice del elemento seleccionado se guarda dicho elemento en la variable `coreConcern`.

En la variable `destDiagram` queda almacenado el Modelo Unificado creado (vacío sin elementos internos) dentro del paquete recibido como parámetro del método `transformation()`.

El método `setContentDiagram()`¹³⁰ le agrega al modelo creado el estereotipo correspondiente (en este caso `<<mobileUWEUnifedDiagram>>`) y se le setea el valor etiquetado `contentDiagram` que se corresponde con el Modelo de Contenido del cual se derivó dicho modelo.

Luego se crea un objeto de la clase *MobileUWEDiagramTransformator*¹³¹ y éste se guarda en la variable `diagramTransformator` de la clase

¹³⁰ La definición de este método se encuentra en la Sección B.4.

¹³¹ Esta clase contiene la funcionalidad para poder realizar transformaciones de elementos en forma genérica. Luego la transformación le agrega a estos elementos creados los estereotipos que se corresponden con las características de la transformación. Cuando se avance en la especificación de esta transformación se mostrará como la transformación interactúa con esta clase.

MobileUWETransformationType. Esta variable es accedida por la transformación cuando lo necesite para completar la transformación como se mostrará más adelante. El método que realiza la transformación de cada uno de los elementos de un modelo a otro es `launchPackageTransformation()`. Este método se encarga de crear por cada elemento del Modelo de Contenido su correspondencia en el Modelo Unificado. Además, agrega las relaciones de rol según el concern elegido como *Core*.

A continuación se irán mostrando todos los detalles de códigos que se disparan a partir de del método `launchPackageTransformation()` que contiene la transformación en cuestión. En el Código B.3 se puede apreciar cómo está definido este método. Se puede ver que se recorren todos los elementos que contiene el Modelo de Contenido los cuáles están almacenados en la variable `elementsInDiagram`. Se van recorriendo los concerns, se realiza la transformación de los mismos y luego se procede a recorrer todas las clases que contiene cada concern para realizar la transformación de las mismas. Luego se recorren todas las asociaciones internas al concerns y las almacenan en la variable `associations`. No se puede realizar la transformación de las asociaciones a está altura ya que alguna de las clases involucradas (en la asociación) pueden no estar creadas aún (por ejemplo, relaciones entre clases de diferentes concerns). Una vez recorridos todos los concerns y creadas todas las clases se procede a recorrer las asociaciones que contiene el Modelo de Contenido para realizar la transformación de las mismas. Luego se realiza la transformación de las generalizaciones. Por último, se invoca al método que crea las relaciones de rol según el concern elegido como *Core*.

```

public void launchPackageTransformation(HashMap<Element, List<PresentationElement>> elementsInDiagram,
    Package destPackage) {
    if (getCoreConcern() != null) {
        SessionManager.getInstance().createSession(MobileUWEStringsGlobalConstants.getTRANSFORMATION());
        HashMap<AssociationImpl, Package> associations = new HashMap<AssociationImpl, Package>();
        for (Element element : elementsInDiagram.keySet()) {
            if (MagicDrawAndMobileUWEOperations.instanceOfPackage(element)) {
                Package transformedPackage = packageTransformation(elementsInDiagram,
                    destPackage, element);

                for (PackageableElement packageElement : ((Package)element).getPackagedElement())
                    if (MagicDrawAndMobileUWEOperations.instanceOfClassImpl(packageElement))
                        classTransformation(elementsInDiagram, transformedPackage, packageElement);
                for (PackageableElement packageElement : ((Package)element).getPackagedElement())
                    if (MagicDrawAndMobileUWEOperations.instanceOfAssociationImpl(packageElement))
                        if (! associationTransformation(elementsInDiagram, transformedPackage, packageElement))
                            associations.put((AssociationImpl) packageElement, transformedPackage);
            }
        }
        for (Element element : associations.keySet())
            associationTransformation(elementsInDiagram, associations.get(element), element);
        for (Element element : elementsInDiagram.keySet())
            if (MagicDrawAndMobileUWEOperations.instanceOfGeneralizationImpl(element))
                generalizationTransformation(elementsInDiagram, destPackage, element);
        createRoleRelationships(elementsInDiagram);
        SessionManager.getInstance().closeSession();
    }
}

```

Código B.3: Método que recorre todos los elementos del modelo origen para transformarlos.

Se puede apreciar en el Código B.3 que los métodos que realizan las transformaciones acordes a cada uno de los elementos del modelo son los siguientes:

- a. `packageTransformation()`
- b. `classTransformation()`
- c. `associationTransformation()`
- d. `generalizationTransformation()`
- e. `createRoleRelationships()`

A continuación se mostrará en detalle cómo están definidos cada uno de estos

métodos, mostrando cómo se crean los elementos en el Modelo Unificado.

a. **packageTransformation()**. En el Código B.4 se puede apreciar el método que realiza la transformación de cada uno de los paquetes del Modelo de Contenido.

```
private Package packageTransformation(HashMap<Element, List<PresentationElement>> elementsInDiagram,
    Package destPackage, Element element) {
    Package transformedPackage = null;
    for (PresentationElement pe : elementsInDiagram.get(element)) {
        if (MagicDrawAndMobileUWEOperations.instanceOfPackageView(pe)) {
            setNumberOfTransformedPackage( getNumberOfTransformedPackage() + 1);
            transformedPackage = getDiagramTransformator().transformPackage(pe, destPackage);
        }
    }
    return transformedPackage;
}
```

Código B.4: Método para transformar paquetes.

El método de transformación de paquetes (`packageTransformation()`) recorre todos los elementos de visualización asociados al paquete (parámetro `element`) y cuando se detecta que es un *PackageView*, se invoca al método `transformPackage()` de la clase *MobileUWEDiagramTransformator*. Este método crea el paquete en el Modelo Unificado y envía a la transformación el mensaje `addConvertedStereotypeToPackage()` para que la transformación agregue el estereotipo correspondiente.

En la Figura B.1 se puede apreciar un diagrama de secuencia reducido que se genera a partir del método `packageTransformation()`, sólo se destacan los principales métodos.

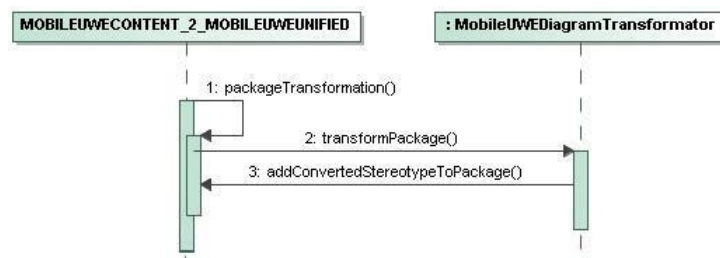


Figura B.1: Diagrama de secuencia reducido del método que transforma los paquetes.

Veamos a continuación los dos métodos involucrados en la Figura B.1, `transformPackage()` y `addConvertedStereotypeToPackage()`.

Se puede visualizar en el Código B.5 la definición del método `transformPackage()` de la clase *MobileUWEDiagramTransformator*. Se crea un elemento que representa el paquete usando el método `createPackageInstance()`¹³². Se setea cual es el owner del elemento y el nombre del mismo. El nombre se obtiene del nombre que tiene el paquete en el Modelo de Contenido. Luego se le envía a la transformación el mensaje `addConvertedStereotypeToPackage()`¹³³ para que sea la transformación la que decida que estereotipo agregar según su naturaleza.

Al definir el método `transformPackage()` de esta manera se logra que el mismo sea usado por todas nuestras transformaciones pero cada una le agrega al paquete el

¹³² Este método (`createPackageInstance()`) está definido en la clase *ElementsFactory* que provee la API de *MagicDraw*. Esta clase se encarga de la creación de todos los elementos bases de UML.

¹³³ Este método (`addConvertedStereotypeToPackage()`) es definido vacío en la clase *MobileUWETransformationType* y cada elemento de esta clase lo redefine acorde a sus características.

estereotipo que corresponda.

El método `setPackageLayout()` le define al paquete las características de visualización dentro del modelo. Es decir, indica en qué posición del modelo se debe ubicar y cuáles son sus medidas (alto y ancho). Estos datos se toman de los datos que tenía el paquete origen (en el Modelo de Contenido) por esta razón cuando se hace la transformación se visualiza que los elementos del Modelo Unificado quedan ubicados en la misma posición que los elementos del Modelo de Contenido.

```
public Package transformPackage(PresentationElement origPackage, Package destPackage) {
    Package newPackage = Application.getInstance().getProject().getElementsFactory().createPackageInstance();
    newPackage.setOwner(destPackage);
    newPackage.setName(origPackage.getName());
    getTransformationType().addConvertedStereotypeToPackage(origPackage, newPackage);
    try {
        setPackageLayout(origPackage, newPackage);
    } catch (Exception e) {
        MobileUWEMessageWriter.showError(
            MobileUWEStringsGlobalConstants.getSOMETHING_WRONG_TRANSFORMING_PACKAGE(), getLogger());
    }
    return newPackage;
}
```

Código B.5: Creación del nuevo paquete en el Modelo Unificado.

La definición del método `addConvertedStereotypeToPackage()` se puede apreciar en el Código B. 6. Este método agrega todos los estereotipos que tenía el paquete origen, permitiendo la flexibilidad de que no sólo se pasen nuestros estereotipos. La clase *StereotypesHelper* está definida en la API de *MagicDraw* y contiene operaciones que facilitan el uso de los estereotipos.

```
public void addConvertedStereotypeToPackage(PresentationElement origPackage, Package newPackage) {
    Package packageElement = (Package) origPackage.getElement();
    StereotypesHelper.addStereotypes(newPackage, StereotypesHelper.getStereotypes(packageElement));
}
```

Código B.6: Agregar el estereotipo correspondiente al paquete.

b. `classTransformation()`. Este método permite la creación de cada una de las clases que tenía el Modelo de Contenido. En el Código B.7 se puede apreciar la definición de este método. Es muy parecida a la definición del método que transforma cada paquete. El método de transformación de clase recorre todos los elementos visuales asociados a la clase y cuando detecta un *ClassView* se envía el mensaje `transformClass()` a la clase *MobileUWEDiagramTransformer*

```
private void classTransformation(HashMap<Element, List<PresentationElement>> elementsInDiagram,
    Package transformedPackage, PackageableElement packageElement) {
    for (PresentationElement classElement : elementsInDiagram.get(packageElement)) {
        if (MagicDrawAndMobileUWEOperations.instanceOfClassView(classElement)) {
            setNumberOfTransformedClasses(getNumberOfTransformedClasses() + 1);
            getDiagramTransformer().transformClass(classElement, transformedPackage);
        }
    }
}
```

Código B.7: Método para transformar las clases.

En la Figura B.2 se puede apreciar un diagrama de secuencia reducido que se focaliza en los principales métodos que se invocan para realizar la transformación de cada clase. El método `transformClass()` de la clase *MobileUWEDiagramTransformer* crea la clase y envía el mensaje `addConvertedStereotypeToClass()` para que la transformación agregue el estereotipo correspondiente.

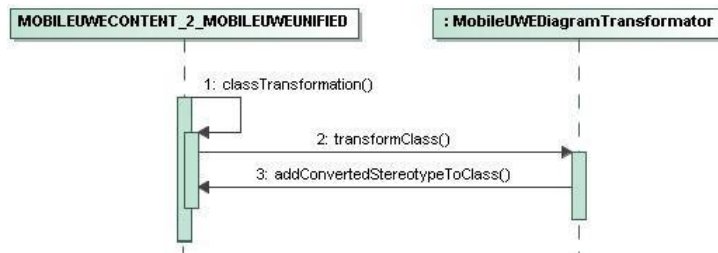


Figura B.2: Diagrama de secuencia reducido del método que transforma las clases.

Veamos a continuación los dos métodos involucrados en la Figura B.2, `transformClass()` y `addConvertedStereotypeToClass()`.

El Código B.8 muestra la definición del método `transformClass()` que realiza la creación de la nueva clase y le agrega las mismas características que tenía la clase origen (del Modelo de Contenido). Se define¹³⁴ la nueva clase con el mismo nombre y con el método `setProperties()` se le crean las mismas propiedades que tenía la clase origen, con los mismos nombres y tipos. Los estereotipos son agregados a la clase por la transformación cuando se invoca el método `addConvertedStereotypeToClass()`.

Luego se invocan métodos relacionados con la visualización de la clase dentro del modelo. El método `savePath()` permite guardar las relaciones que tiene esta clase con otras, esta información luego es usada cuando las clases son abstractas.

```

Class transformClass(PresentationElement origClass, Package destPackage) {
    Class newClass = Application.getInstance().getProject().getElementsFactory().createClassInstance();
    newClass.setOwner(destPackage);
    newClass.setName(origClass.getName());
    Class orig = (Class) origClass.getElement();
    newClass.setAbstract(orig.isAbstract());
    setProperties(orig, newClass);
    getTransformationType().addConvertedStereotypeToClass(origClass, newClass);
    try {
        ShapeElement shape = setClassLayout(origClass, newClass);
        updateHashMapsValues(origClass, newClass, orig, shape);
        savePath(origClass, destPackage);
    } catch (Exception e) {
        e.printStackTrace();
        MobileUWMessageWriter.showError(
            MobileUWStringsGlobalConstants.getSOMETHING_WRONG_TRANSFORMING_CLASSES(), getLogger());
    }
    return newClass;
}
  
```

Código B.8: Creación de la nueva clase.

La definición del método `addConvertedStereotypeToClass()` se puede visualizar en el Código B.9. Se puede apreciar que no se agregan estereotipos nuevos sino que si la clase tenía algún estereotipo, éste es agregado en la clase del Modelo Unificado. De esta manera, se permite que se pasen al Modelo Unificado otros estereotipos aparte de los definidos por nuestros perfiles.

```

public void addConvertedStereotypeToClass(PresentationElement origClass, Class newClass) {
    Class classElement = (Class) origClass.getElement();
    StereotypesHelper.addStereotypes(newClass, StereotypesHelper.getStereotypes(classElement));
}
  
```

Código B.9: Agregar el estereotipo correspondiente a la clase.

c. **associationTransformation()**. Este método crea las asociaciones entre las clases que tenía el Modelo de Contenido. En el Código B.10 se puede apreciar cómo

¹³⁴ La nueva clase usando el método `createClassInstance()` de la clase `ElementsFactory` (de `MagicDraw`). Esta clase se encarga de la creación de todos los elementos bases de UML.

es la definición del método. Se recorren todos los elementos visuales relacionados a la asociación y cuando se detecta un *AssociationView* se envía el mensaje *transformAssociation()* a la clase *MobileUWEDiagramTransformator*.

```
private boolean associationTransformation(HashMap<Element, List<PresentationElement>> elementsInDiagram,
    Package transformedPackage, Element packageElement) {
    for (PresentationElement assoc : elementsInDiagram.get(packageElement)) {
        if (MagicDrawAndMobileUWEOperations instanceof AssociationView(assoc)) {
            if (getDiagramTransformator().transformAssociation(assoc, transformedPackage)){
                setNumberOfTransformedAssociations(getNumberOfTransformedAssociations() + 1) ;
                return true;
            }
        }
    }
    return false;
}
```

Código B.10: Método para transformar las asociaciones.

En la Figura B.3 se puede apreciar los principales métodos que se invocan para realizar la transformación de las asociaciones mediante un diagrama de secuencia reducido. La clase *MobileUWEDiagramTransformator* crea la asociación y luego envía el mensaje *addConvertedStereotypeToAssociation()* a la transformación que la misma agregue el estereotipo correspondiente.

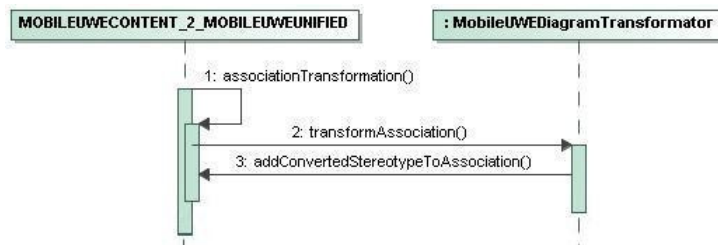


Figura B.3: Diagrama de secuencia reducido del método que transforma las asociaciones.

Veamos a continuación los dos métodos involucrados en la Figura B.3, *transformAssociation()* y *addConvertedStereotypeToAssociation()*.

En el Código B.11 se puede apreciar la definición del método *transformAssociation()* que se encarga de la creación de la asociación entre las clases correspondientes. Se obtienen las clases que tenían la asociación origen (*origClient* y *origSupplier*) y las nuevas clases que se deben relacionar (*newClient* y *newSupplier*). Luego se invoca el método *createAndSetValuesAssociation()* que se encarga de la creación¹³⁵ de la asociación (agregando las nuevas clases como origen y destino) y se le define la visualización dentro del modelo.

Además en el método *createAndSetValuesAssociation()* se le asigna a la asociación creada, el nombre y la cardinalidad según la asociación origen. Dentro de este método además se invoca al método *addConvertedStereotypeToAssociation()* de la transformación.

Se puede apreciar el método *addConvertedStereotypeToAssociation()* en el Código B.12. Se agregan a la nueva asociación los estereotipos que tenía la asociación origen. Esto permite que en el modelo se utilicen otros estereotipos que no sean sólo los definidos por nuestros perfiles.

¹³⁵ Para la creación de cada asociación se usa de la clase *ElementsFactory* (de la API de *MagicDraw*) el método *createAssociationInstance()*.

```

boolean transformAssociation(PresentationElement origAssoc, Package destPackage) {
    PathElement pathEl = (PathElement) origAssoc;
    PresentationElement origClient = getPath2origClient().get(pathEl);
    PresentationElement newClient = getTransformedClassElements().get(origClient);
    PresentationElement origSupplier = getPath2origSupplier().get(pathEl);
    PresentationElement newSupplier = getTransformedClassElements().get(origSupplier);
    if (origClient != null && newClient != null && origSupplier != null && newSupplier != null) {
        return createAndSetValuesAssociation(origAssoc, destPackage, pathEl, newClient, newSupplier);
    }
    return false;
}

```

Código B.11: Creación de la nueva asociación.

```

public void addConvertedStereotypeToAssociation(PresentationElement origAssociation,
    Association newAssociation) {
    Association assocElement = (Association) origAssociation.getElement();
    StereotypesHelper.addStereotypes(newAssociation, StereotypesHelper.getStereotypes(assocElement));
}

```

Código B.12: Agregar el estereotipo correspondiente a la asociación.

d. `generalizationTransformation()`. Este método crea las relaciones que representan la generalización entre clases. El Código B.13 muestra como se recorren todos los elementos de visualización asociados a la generalización y cuando se detecta un *GeneralizationView* se envía el mensaje `transformGeneralization()`. Se puede observar que es similar al tratamiento de las asociaciones.

```

private void generalizationTransformation(HashMap<Element, List<PresentationElement>> elementsInDiagram,
    Package transformedPackage, Element packageElement) {
    for (PresentationElement generalization : elementsInDiagram.get(packageElement)) {
        if (MagicDrawAndMobileUWEOperations instanceof GeneralizationView(generalization))
            getDiagramTransformator().transformGeneralization(generalization, transformedPackage);
    }
}

```

Código B.13: Método para transformar las generalizaciones.

En la Figura B.4 se puede apreciar cómo es la secuencia para crear la generalización. En este caso la clase *MobileUWEDiagramTransformation* no invoca ningún método de la transformación. En el caso que la generalización tuviera algún estereotipo definido del enfoque no lo está pasando en la creación del Modelo Unificado¹³⁶.

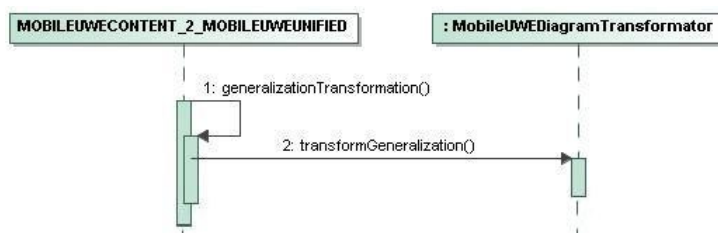


Figura B.4: Diagrama de secuencia reducido del método que transforma las generalizaciones.

El método `transformGeneralization()` se puede observar en el Código B.14. Se puede apreciar que se trata de manera muy similar a la asociación, se obtienen las clases origen (`origClient`) y destino (`origSupplier`) de la generalización y las nuevas clases que se deben relacionar (`newClient` y `newSupplier`). Se crea¹³⁷ la generalización mediante el método `createGeneralization()`. Luego se le asigna la misma las clases correspondientes y su visualización dentro del Modelo Unificado.

¹³⁶ En futuras evoluciones de nuestra herramienta podría definir el pasaje de los estereotipos de las generalizaciones al Modelo Unificado.

¹³⁷ Para la creación de cada asociación se usa de la clase *ElementsFactory* (de la API de *MagicDraw*) el método `createGeneralizationInstance()`.

```

boolean transformGeneralization(PresentationElement origGeneralization, Package destPackage) {
    boolean thisAssociationWasTransformed = false;
    PathElement pathEl = (PathElement) origGeneralization;
    PresentationElement origClient = getPath2origClient().get(pathEl);
    PresentationElement newClient = getTransformedClassElements().get(origClient);
    PresentationElement origSupplier = getPath2origSupplier().get(pathEl);
    PresentationElement newSupplier = getTransformedClassElements().get(origSupplier);
    if (origClient != null && newClient != null && origSupplier != null && newSupplier != null) {
        Generalization newGeneralization = createGeneralization(origGeneralization, destPackage);
        try {
            setGeneralizationValues(origGeneralization, newClient, newSupplier, newGeneralization);
            generalizationLayout(pathEl, newClient, newSupplier, newGeneralization);
            thisAssociationWasTransformed = true;
        } catch (Exception e) {
            e.printStackTrace();
            MobileUWEMessageWriter.showError(
                MobileUWEMessageGlobalConstants.getSOMETHING_WRONG_TRANSFORMING_ASSOCIATION(),
                getLogger());
        }
    }
    return thisAssociationWasTransformed;
}

```

Código B.14: Creación de la nueva generalización.

e. `createRoleRelationships()`. La creación de los roles no es realmente una transformación porque en el Modelo de Contenido no existe este elemento. Este método crear la asociación y le agrega el estereotipo. En el Código B.15 se ve la definición de este método. Primero se buscan todas las clases del modelo y se guardan en `allCoreClasses` todas aquellas que están en el concern *Core*. El resto de las clases se guardan en `allClasses` (inclusive las clases del concern físico). Esto se realiza cuando se invoca el método `collectClasses()`.

Luego invocando el método `associateClassWithRole()` se guarda en la variable `roleClasses` cada una de las clases con sus correspondientes roles. La agrupación de las clases se realiza por los nombres de las mismas. Si una clase no tiene representación en el concern *Core*, este método no la agregar en `roleClasses`. Al invocar el método `createRoleCoreClass()` se crean las asociaciones que representa el rol entre todas las clases definidas en `roleClasses`.

Hasta esta parte el Código B.15 genera las asociaciones de rol entre las clases que tienen su representación en el concern *Core*. Falta agregar la relación de rol para aquellas clases que no tienen representación en el concern *Core*. Para esto primero se utiliza el método `associateClassByNames()`, el cual agrupa por nombre aquellas clases que están en los concerns digitales y no tienen su representación en el concern *Core* (`otherCore`) y además agrupa por nombre las clases del concern físico (`physicalClass`). Luego el método `associateClassWithRole()` recorre todos los nombres detallados en `otherCore` y por cada nombre toma la primera de las clases agrupadas bajo ese nombre y esa se define como *Core*. El resto de las clases agrupadas bajo ese mismo nombre serán roles.

También se busca en `physicalClass` si hay alguna clase del concern físico con el nombre que se está analizando, en caso afirmativo se agrega como un rol más de la clase. Esta agrupación de clase *Core* y sus roles queda representadas en `otherRoleClasses`¹³⁸.

Para la creación de las asociaciones que representan los roles se invoca el método `createRoleCoreClass()` de la clase `MobileUWETransformationType`. Se puede apreciar que dicho método se invoca dos veces dentro del Código B.15 y lo que se envía como parámetro es el valor correspondiente según las asociaciones de rol que se quieran crear en cada caso.

¹³⁸ Esta variable se define como `HashMap<Class, List<Class>>`, agrupa la clase y la lista de clases que son roles.

```

private void createRoleRelationships(HashMap<Element, List<PresentationElement>> elementsInDiagram) {
    List<Class> allClasses = new ArrayList<Class>();
    List<Class> allCoreClasses = new ArrayList<Class>();
    collectClasses(elementsInDiagram, allClasses, allCoreClasses);

    HashMap<Class, List<Class>> roleClasses = new HashMap<Class, List<Class>>();
    List<Class> aux = associateClassWithRole(allClasses, allCoreClasses, roleClasses);
    getDiagramTransformator().createRoleCoreClass(roleClasses);

    HashMap<String, List<Class>> otherCore = new HashMap<String, List<Class>>();
    HashMap<String, Class> physicalClass = new HashMap<String, Class>();
    List<String> names = associateClassByNames(aux, otherCore, physicalClass);
    HashMap<Class, List<Class>> otherRoleClasses = associateClassWithRole(otherCore,
                                                                           physicalClass, names);
    getDiagramTransformator().createRoleCoreClass(otherRoleClasses);
}

```

Código B.15: Método para crear los roles.

En el Código B.16 se puede apreciar la definición del método `createRoleCoreClass()`. Se puede observar que se recorre todas las clases elegidas como *Core* (client) y de cada una las clases que son roles (supplier). Por cada iteración se crea una asociación a la cual se le agrega el estereotipo `<<roleOf>>`. Además se le agregan a la asociación características como navegabilidad y multiplicidad para que en el modelo quede claro cual es la clase *Core*. Luego se crea la visualización¹³⁹ correspondiente para que sea visualizada en el modelo. Como el Modelo de Contenido no tenía esta asociación la visualización creada se genera usando los valores por default que establece *MagicDraw*.

```

public void createRoleCoreClass(HashMap<Class, List<Class>> roleClasses) {
    for (Class core : roleClasses.keySet()) {
        for (Class role : roleClasses.get(core)) {
            Association newAss =
                Application.getInstance().getProject().getElementsFactory().createAssociationInstance();
            newAss.setOwner(getTransformedClasses().get(role).getOwner());
            StereotypesHelper.addStereotypeByString(newAss,
                MobileUWEStereotypeAssoc.UNIFIED_ROLE_OF.toString());
            ModelHelper.setClientElement(newAss, getTransformedClasses().get(core));
            ModelHelper.setSupplierElement(newAss, getTransformedClasses().get(role));
            List<Property> newMemberEnds = newAss.getMemberEnd();
            Property newProperty = newMemberEnds.get(1);
            ModelHelper.setNavigable(newProperty, true);
            ModelHelper.setMultiplicity(1, 1, newProperty);
            PresentationElement presentationClient = getTransformedPresentationElement().get(core);
            PresentationElement presentationSupplier = getTransformedPresentationElement().get(role);
            PathElement newAssPath;
            try {
                newAssPath = PresentationElementsManager.getInstance().createPathElement(newAss,
                    presentationClient, presentationSupplier);
                PresentationElementsManager.getInstance().resetLabelPositions(newAssPath);
            } catch (ReadOnlyElementException e) { e.printStackTrace(); }
        }
    }
}

```

Código B.16: Creación de la asociación que representa al rol.

B.2. Transformación del Modelo de Contenido al Modelo Unificado seleccionando algunos concerns

Esta transformación tiene la particularidad, con respecto de la presentada en la Sección B.1, de filtrar los concerns de interés. En el Código B.17 se puede apreciar cómo se crea la transformación, el primer parámetro indica el nombre de la misma, el segundo parámetro indica el tipo del modelo de origen (en este caso un Modelo de Contenido) y el último parámetro indica el tipo del modelo a crear (en este caso un Modelo Unificado). Esta transformación define cuatro variables. Dos son iguales a la

¹³⁹ La clase *PresentationElementsManager* es la clase de la API de *MagicDraw* encargada de crear los elementos que se corresponden a la visualización de los elementos UML.

transformación definida en la Sección B.1 (`coreConcern` y `digitalConcerns`) y dos son variables nuevas (`physicalConcern` y `selectedDigitalConcerns`). Estas dos variables nuevas permiten especificar cuáles son los concerns digitales de interés y además tener separado el concern físico.

```
MOBILEUWECONTENT_2_MOBILEUWEUNIFIED_SOME_CONCERNS(
    MobileUWEGlobalConstants.getNAME_CONTENT_TO_UNIFIED_SOME_CONCERNS(),
    MobileUWEDiagramType.MOBILEUWECONTENT,
    MobileUWEDiagramType.MOBILEUWEUNIFIED) {

protected Package coreConcern;
protected Package physicalConcern;
protected List<Package> digitalConcerns = new ArrayList<Package>();
protected List<Package> selectedDigitalConcerns = new ArrayList<Package>();
```

Código B.17: Definición de la transformación del Modelo de Contenido al Modelo Unificado que filtra concerns.

Como se mencionó anteriormente en la introducción de este anexo, el método que comienza la transformación es `transformation()`. En el Código B.18 se puede apreciar cómo se define dicho método para esta transformación.

```
protected void transformation(DiagramPresentationElement sourceDiagram, Package destPackage) {
    HashMap<Element, List<PresentationElement>> elementsInDiagram =
        new HashMap<Element, List<PresentationElement>>();

    sourceDiagram.open();
    MobileUWEDiagramType.collectElementsAndPresentationElements(sourceDiagram, elementsInDiagram);
    resetDigitalConcerns();
    resetSelectedDigitalConcerns();
    collectDigitalConcerns(elementsInDiagram);
    setPhysicalConcern(elementsInDiagram);
    if (hasConcernsWithNames()) {
        int index = (new MobileUWEGUISelector(getDigitalConcernsNames(),
            MobileUWEGUIGlobalConstants.getSELECT_CORE()).getIndex());

        if (index != -1) {
            setCoreConcern(index);
            if (selectOtherConcerns(index)){
                Diagram destDiagram = MobileUWEDiagramType.MOBILEUWEUNIFIED.createAndAddDiagram(destPackage);
                setContentDiagram(sourceDiagram, destDiagram);
                setDiagramTransformator(new MobileUWEDiagramTransformator(sourceDiagram, destDiagram,
                    destPackage, this));

                HashMap<Element, List<PresentationElement>> elementsInSelectedDigitalConcerns =
                    getElementsInSelectedDigitalConcerns(elementsInDiagram);
                launchPackageTransformation(elementsInSelectedDigitalConcerns, destPackage);
                showDoneMessage(sourceDiagram.getName());
            }
        }
    } else {
        MobileUWEMessageWriter.showMessage(MobileUWEGUIGlobalConstants.getNO_CONCERN(), getLogger());
    }
}
```

Código B.18: Método inicial invocado para realizar la transformación.

Se puede observar que el método `transformation()` está definido de forma similar al método presentado en el Código B.1, pero se agrega la posibilidad de seleccionar aquellos concerns digitales que son de interés y realizar la transformación sólo considerando los concerns digitales elegidos.

Se puede apreciar que primero se crea la variable `elementsInDiagram` que servirá para contener todos los elementos del Modelo de Contenido origen. Luego se invoca el método `collectElementsAndPresentationElements()`¹⁴⁰ que recolecta todos los elementos de Modelo de Contenido origen y se guardan en la variable `elementsInDiagram`. Se puede apreciar que primero se abrió (`sourceDiagram.open()`) el modelo porque así están disponibles todos los elementos del mismo.

Siguiendo con el análisis del Código B.18 se observa que se resetea la colección de

¹⁴⁰ La definición de este método se encuentra en la Sección B.4.

concerns digitales (`digitalConcerns`) y la colección que contiene los concerns digitales seleccionados (`selectedDigitalConcerns`). De esta manera no quedan registros de los datos anteriores de las transformaciones. Luego se obtienen todos los concerns digitales que tiene el Modelo de Contenido mediante el método `collectDiagramsConcerns()`¹⁴⁰. Todos los concerns digitales que tienen nombre son usados para ser desplegados por la ventana *MobileUWEGUISelector*, la cual le muestra al diseñador todos nombres (obtenidos mediante el método `getDiagramsConcernsNames()`) para que éste seleccione uno como *Core*. El índice es usado para guardar el concern *Core* en `coreConcern`.

Para elegir los otros concern de interés se invoca el método `selectOtherConcerns()`¹⁴⁰ que despliega una ventana con los otros concerns digitales disponibles para que el diseñador seleccione aquellos que le son de interés y estos son guardados en `selectedDigitalConcerns`.

En la variable `destDiagram` queda almacenado el Modelo Unificado creado (vacío sin elementos internos) dentro del paquete recibido como parámetro del método `transformation()`. El método `setContentDiagram()` le agrega al modelo creado el estereotipo correspondiente (`<<mobileUWEUnifedDiagram>>`) y se le setea el valor etiquetado `contentDiagram` que se corresponde con el Modelo de Contenido del cual se derivó dicho modelo.

Luego se crea un objeto de la clase *MobileUWEDiagramTransformator* y éste se guarda en la variable `diagramTransformator` de la clase *MobileUWETransformationType*. Esta variable es accedida por la transformación cuando lo necesite para completar la transformación como se mostrará más adelante.

Se invoca al método `getElementsInSelectedDigitalConcerns()` para almacenar en `elementsInSelectedDigitalConcerns` todos los elementos de los concerns de interés. Estos son los elementos que se usan para realizar la transformación.

El método que realiza la transformación de cada uno de los elementos de un modelo a otro es `launchPackageTransformation()` que recibe como parámetro sólo los elementos de interés para la transformación (`elementsInSelectedDigitalConcerns`). Este método se encarga de crear por cada elemento del Modelo de Contenido su correspondencia en el Modelo Unificado. Además agrega las relaciones de rol según el concern elegido como *Core*. Este método se ejecutará sobre los elementos de los concerns de interés y se define de la misma manera que para la transformación presentada en la Sección B.1 en el Código B.3.

Se puede apreciar que esta transformación agrega un filtrado en los elementos que trata pero tiene las mismas características que la transformación presentada en la Sección B.1.

B.3. Transformación del Modelo Unificado al Modelo Navegacional

La transformación del Modelo Unificado al Modelo Navegacional presenta más variantes respecto de las dos transformaciones anteriores, ya que varían los estereotipos que se deben definir en el nuevo modelo. En el Código B.19 se puede apreciar la definición de la transformación la cual no define ninguna variable en particular.

```
MOBILEUWEUNIFIED_2_MOBILEUWENAVIGATION(MobileUWEGlobalConstants.getName_Unified_To_Navigation(),
    MobileUWEDiagramType.MOBILEUWEUNIFIED,
    MobileUWEDiagramType.MOBILEUWENAVIGATION) {
```

Código B.19: Definición de la transformación del Modelo Unificado al Modelo Navegacional.

Como se mencionó anteriormente la transformación comienza mediante la invocación del método `transformation()`. En el Código B.20 se puede apreciar cómo se definió

este método para la transformación definida en el Código B.19. Se observa que primero se crea la variable `elementsInDiagram` que servirá para contener todos los elementos del Modelo de Contenido origen. Se abre el modelo en cuestión (`sourceDiagram.open()`) para poder acceder a sus elementos. Luego se invoca el método `collectElementsAndPresentationElements()`¹⁴¹ que recolecta todos los elementos del Modelo de Contenido origen y los guarda en la variable `elementsInDiagram`.

Luego se crea el Modelo Navegacional. El método `setUnifiedDiagram()` busca el estereotipo (`<<mobileUWENavigationDiagram>>`) que se le debe asignar al nuevo modelo y se le define el valor etiquetado `unifiedDiagram` que almacena el Modelo Unificado del cual se deriva.

Se crea un objeto de la clase `MobileUWEDiagramTransformer` y éste se guarda en la variable `diagramTransformer` de la clase `MobileUWETransformationType` y ésta será accedida por la transformación cuando lo necesite.

```
protected void transformation(DiagramPresentationElement sourceDiagram, Package destPackage) {
    HashMap<Element, List<PresentationElement>> elementsInDiagram =
        new HashMap<Element, List<PresentationElement>>();
    sourceDiagram.open();
    MobileUWEDiagramType.collectElementsAndPresentationElements(sourceDiagram, elementsInDiagram);
    Diagram destDiagram = MobileUWEDiagramType.MOBILEUWENAVIGATION.createAndAddDiagram(destPackage);
    setUnifiedDiagram(sourceDiagram, destDiagram);
    setDiagramTransformer(new MobileUWEDiagramTransformer(sourceDiagram, destDiagram, destPackage, this));
    launchPackageTransformation(elementsInDiagram, destPackage);
    showDoneMessage(sourceDiagram.getName());
}
```

Código B.20: Método inicial invocado para realizar la transformación.

El método que realiza la transformación de cada uno de los elementos es `launchPackageTransformation()`. Este método se encarga de crear por cada elemento del Modelo Unificado su correspondencia en el Modelo Navegacional. En esta sección se irán mostrando todos los detalles de códigos que se disparan a partir de este método (que contiene la transformación en cuestión). La definición de este método se puede observar en el Código B.21.

```
public void launchPackageTransformation(HashMap<Element, List<PresentationElement>> elementsInDiagram,
    Package destPackage) {
    SessionManager.getInstance().createSession(MobileUWEStringsGlobalConstants.getTRANSFORMATION());
    HashMap<AssociationImpl, Package> associations = new HashMap<AssociationImpl, Package>();
    for (Element element : elementsInDiagram.keySet()) {
        if (MagicDrawAndMobileUWEOperations.getInstanceOfPackage(element)) {
            Package transformedPackage = packageTransformation(elementsInDiagram, destPackage, element);
            for (PackageableElement packageElement : ((Package) element).getPackagedElement())
                if (MagicDrawAndMobileUWEOperations.getInstanceOfClassImpl(packageElement))
                    classTransformation(elementsInDiagram, transformedPackage, packageElement);
            for (PackageableElement packageElement : ((Package) element).getPackagedElement())
                if (MagicDrawAndMobileUWEOperations.getInstanceOfAssociationImpl(packageElement))
                    associations.put((AssociationImpl) packageElement, transformedPackage);
        }
    }
    for (Element element : associations.keySet()) {
        if (StereotypesHelper.hasStereotype(element, MobileUWEStereotypeAssoc.UNIFIED_ROLE_OF.toString()))
            transformRoleOf(elementsInDiagram, associations.get(element), element);
        else associationTransformation(elementsInDiagram, associations.get(element), element);
    }
    SessionManager.getInstance().closeSession();
}
```

Código B.21: Método que recorre todos los elementos del modelo origen para transformarlos.

Se puede apreciar en el Código B.21 que se recorren todos los elementos que contiene el Modelo Unificado los cuales están almacenados en la variable `elementsInDiagram`. Por cada paquete que se detecta luego se buscan las clases

¹⁴¹ La definición de este método se encuentra en la Sección B.4.

internas al mismo para transformarlas y además se almacenan las asociaciones. Luego se recorren todas las asociaciones almacenadas para transformarlas, ya sean relaciones de rol o no.

En el Código B.21 se puede apreciar que se tienen cuatro métodos que son invocados para realizar el proceso de la transformación, estos son:

- a. `packageTransformation()`
- b. `classTransformation()`
- c. `associationTransformation()`
- d. `transformRoleOf()`

A continuación se mostrará en detalle cada uno de ellos, mostrando solamente aquellos métodos que varían respecto de los presentados en la Sección B.1.

a. **`packageTransformation()`**. En la Figura B.5 se puede apreciar el diagrama de secuencia (reducido) asociado a la definición de este método, es igual al presentado en la Figura B.1 sólo cambia el nombre de la transformación.

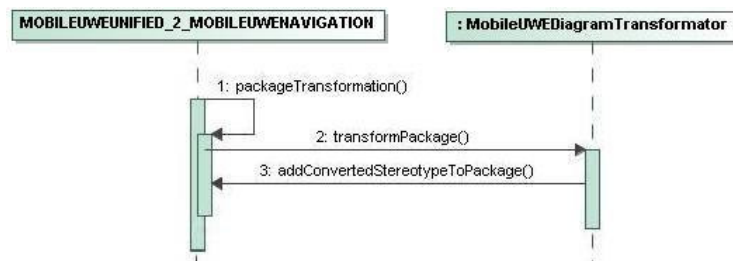


Figura B.5: Diagrama de secuencia reducido del método que crear los paquetes.

Tanto el método `packageTransformation()` como `transformPackage()` se definen de la misma manera que los Códigos B.4 y B.5 respectivamente. Es decir, usan la misma lógica ya especificada por esta razón no se volverán a mostrar. En esta transformación sólo varían la definición del método `addConvertedStereotypeToPackage()`. Esto se debe a que los estereotipos asociados a los concerns navegacionales no son los mismos que los definidos en el Modelo de Contenido.

El Código B.22 muestra la definición del método `addConvertedStereotypeToPackage()`. Se puede apreciar que según el estereotipo que tenía el concern origen (en el Modelo Unificado) es el que se agrega al concern del Modelo Navegacional creado. Si los concerns en el Modelo Unificado poseen más estereotipos estos no son pasados al Modelo Navegacional.

```
public void addConvertedStereotypeToPackage(PresentationElement origPackage, Package newPackage) {
    Package packageElement = (Package) origPackage.getElement();
    if (StereotypesHelper.hasStereotype(packageElement,
        MobileUWEStereotypePackage.DIGITAL_CONTENT.getName()) {
        StereotypesHelper.addStereotypeByString(newPackage,
            MobileUWEStereotypePackage.DIGITAL_NAVIGATION.getName());
    } else if (StereotypesHelper.hasStereotype(packageElement,
        MobileUWEStereotypePackage.PHYSICAL_CONTENT.toString())
        StereotypesHelper.addStereotypeByString(newPackage,
            MobileUWEStereotypePackage.PHYSICAL_NAVIGATION.getName());
}
```

Código B.22: Agregar el estereotipo correspondiente al paquete.

b. **`classTransformation()`**. En la Figura B.6 se puede apreciar el diagrama de

secuencia (reducido) asociado a la definición del método. Es igual al presentado en la Figura B.2, sólo cambia el nombre de la transformación.

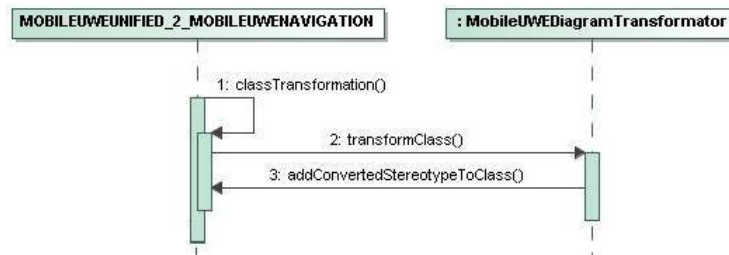


Figura B.6: Diagrama de secuencia reducido del método que crea las clases.

El método `transformClass()` se definen de la misma manera que el Código B.8. En esta transformación sólo varía las definiciones de los métodos `classTransformation()` y `addConvertedStereotypeToClass()`.

En el Código 23 se puede apreciar la definición del método `classTransformation()`, el cual debe controlar si es una clase abstracta o no. Sólo se sigue con la transformación si no es una clase. En el caso de ser una clase abstracta no se crea ninguna clase en el Modelo Navegacional.

```

private void classTransformation(HashMap<Element, List<PresentationElement>> elementsInDiagram,
    Package transformedPackage, PackageableElement packageElement) {
    Class transformedClass = null;
    List<PresentationElement> classList = elementsInDiagram.get(packageElement);
    PresentationElement classElement = null;
    boolean pass2 = false;
    for (int j = 0; j < classList.size(); j++) {
        classElement = classList.get(j);
        if (classElement instanceof ClassView) {
            if (!(ClassImpl) classElement.getElement().isAbstract()) {
                setNumberOfTransformedClasses(getNumberOfTransformedClasses() + 1);
                if (!pass2) {
                    transformedClass = getDiagramTransformator().transformClass(classElement,
                                                                transformedPackage);
                    pass2 = true;
                } else getDiagramTransformator().setClassLayout(classElement, transformedClass);
            } else getDiagramTransformator().savePath(classElement, transformedPackage);
        }
    }
}

```

Código B.23: Método para transformar las clase.

El método `addConvertedStereotypeToClass()` se muestra en el Código B.24. Se puede apreciar que se controla cuál es el estereotipo que se le debe agregar a la clase para indicar si es un nodo digital o físico.

El estereotipo dependerá si la clase está contenida en un concern digital o físico. Se puede observar que se le agrega el valor etiquetado del estereotipo (`<<navigationClass>>` o `<<physicalNode>>`) para indicar la clase origen de la cual se genero el nodo en cuestión. Para esto se debe recuperar el *Slot*¹⁴² que se corresponde con la propiedad y luego a este *Slot* se le agrega el valor que se desea asignar.

¹⁴² La interfase *Slot* representa el valor particular que puede tomar una propiedad (ya sea de una clase o de un estereotipo). La API provee la clase *StereotypeHelper* que permite la recuperación de un *Slot* a partir de por ejemplo indicar una clase particular y una propiedad de la misma.

```

public void addConvertedStereotypeToClass(PresentationElement origClass, Class newClass) {
    Class classElement = (Class) origClass.getElement();
    Stereotype stereotype = null;
    Property property = null;
    if (StereotypesHelper.hasStereotype(newClass.getOwner(),
        MobileUWEStereotypePackage.DIGITAL_NAVIGATION.toString()){
        StereotypesHelper.addStereotypeByString(newClass,
            MobileUWEStereotypesGlobalConstants.getNAVIGATION_CLASS_STEREOTYPE());
        stereotype = StereotypesHelper.getStereotype(Application.getInstance().getProject(),
            MobileUWEStereotypesGlobalConstants.getNAVIGATION_CLASS_STEREOTYPE());
        property = StereotypesHelper.getPropertyByName(stereotype,
            MobileUWEStereotypesGlobalConstants.getCONTENT_CLASS_STEREOTYPE());
    } else if (StereotypesHelper.hasStereotype(newClass.getOwner(),
        MobileUWEStereotypePackage.PHYSICAL_NAVIGATION.toString()){
        StereotypesHelper.addStereotypeByString(newClass,
            MobileUWEStereotypeClassNav.PHYSICAL_NODE.toString());
        stereotype = StereotypesHelper.getStereotype(Application.getInstance().getProject(),
            MobileUWEStereotypeClassNav.PHYSICAL_NODE.toString());
        property = StereotypesHelper.getPropertyByName(stereotype,
            MobileUWEStereotypesGlobalConstants.getCONTENT_PHYSICAL_CLASS_STEREOTYPE());
    }
    if (stereotype != null && property != null) {
        Slot slot = StereotypesHelper.getSlot(newClass, property, true, false);
        if (slot != null) {
            ElementValue value =
                Application.getInstance().getProject().getElementsFactory().createElementValueInstance();
            value.setElement(classElement);
            slot.getValue().add(value);
        }
    }
}
}

```

Código B.24: Agregar el estereotipo correspondiente a la clase.

c. `associationTransformation()`. En la Figura B.7 se muestra el diagrama de secuencia (reducido) asociado a la definición del método, es muy parecido al presentado en la Figura B.3.

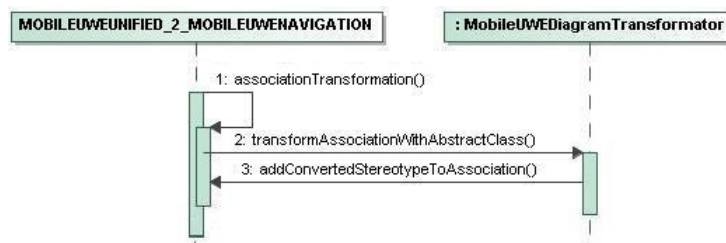


Figura B.7: Diagrama de secuencia reducido del método que crear las asociaciones.

Si bien el método `associationTransformation()` tiene la misma estructura definida en el Código B.10 cambia el método que invoca. Esto se debe a que en la transformación se debe considerar que las clases abstractas no se pasan como nodos, por lo tanto no se puede usar el mismo método. Se invoca al método `transformAssociationWithAbstractClass()`, en vez de llamar al método `transformAssociation()`, como se ve en la figura. Además se mostrará que el método `addConvertedStereotypeToAssociation()` también varía respecto del presentado en el Código B.12, ya que se deben agregar los estereotipos correspondientes para indicar que la asociación representa un link.

En el Código B.25 se define el método `transformAssociationWithAbstractClass()` que se encarga de la creación de la asociación entre las clases correspondientes, considerando que las clases abstractas no son consideradas, por lo tanto estas relaciones pasan a sus subclases. Se puede observar que se obtienen las clases que tenían la relación origen (`origClient` y `origSupplier`) y las nuevas clases que se deben relacionar (`newClient` y `newSupplier`). Luego se buscan todas las subclases de `newClient` y `newSupplier` sólo en el caso de que estas sean abstractas (las

subclases quedan almacenadas en `firstConnector` y `secondConnector`). En el caso de que las clases `newClient` y `newSupplier` no sean abstractas, en `firstConnector` y `secondConnector` se asignan estas clases. De esta manera se recorren `firstConnector` y `secondConnector`, sin importar si las clases relacionadas son o no clases abstractas y se ejecuta el método `createAndSetValuesAssociation()`¹⁴³. Este último método es el encargado de invocar al método `addConvertedStereotypeToAssociation()`.

```
int transformAssociationWithAbstractClass(PresentationElement origAssoc, Package destPackage) {
    PathElement pathEl = (PathElement) origAssoc;
    PresentationElement origClient = getPath2origClient().get(pathEl);
    PresentationElement newClient = getTransformedClassElements().get(origClient);
    PresentationElement origSupplier = getPath2origSupplier().get(pathEl);
    PresentationElement newSupplier = getTransformedClassElements().get(origSupplier);
    List<PresentationElement> firstConnectors = new ArrayList<PresentationElement>();
    List<PresentationElement> secondConnectors = new ArrayList<PresentationElement>();
    calculateSubclasses(origClient, newClient, origSupplier, newSupplier, firstConnectors, secondConnectors);
    int i = 0;
    for (PresentationElement firstConnector: firstConnectors){
        for (PresentationElement secondConnector: secondConnectors){
            if (createAndSetValuesAssociation(origAssoc, destPackage, pathEl, firstConnector, secondConnector))
                i++;
        }
    }
    return i;
}
```

Código B.25: Creación de la nueva asociación considerando que alguna de las clases puede ser abstracta.

La definición del método `addConvertedStereotypeToAssociation()` se puede apreciar en el Código B.26.

```
public void addConvertedStereotypeToAssociation(PresentationElement origAssociation,
    Association newAssociation) {
    Association associationElement = (Association) origAssociation.getElement();
    if (StereotypesHelper.hasStereotype(associationElement,
        MobileUWEStereotypeAssoc.UNIFIED_ROLE_OF.toString()))
        StereotypesHelper.addStereotypeByString(newAssociation,
            MobileUWEStereotypeAssoc.NAVIGATIONAL_ROLE_OF.toString());
    else if (StereotypesHelper.hasStereotype(associationElement.getOwner(),
        MobileUWEStereotypePackage.DIGITAL_CONTENT.toString()){
        StereotypesHelper.addStereotypeByString(newAssociation,
            MobileUWEStereotypesGlobalConstants.getNAVIGATION_LINK_STEREOTYPE());
    } else if (StereotypesHelper.hasStereotype(associationElement.getOwner(),
        MobileUWEStereotypePackage.PHYSICAL_CONTENT.toString()){
        StereotypesHelper.addStereotypeByString(newAssociation,
            MobileUWEStereotypeAssoc.WALKING_LINK.toString());
    }
}
```

Código B.26: Agregar el estereotipo correspondiente a la asociación.

Se puede observar en el Código B.26 que este método es usado para asignar el estereotipo a cualquier asociación que se crea en la transformación del Modelo Unificado al Navegacional. Por esta razón se verifica si la asociación tenía el estereotipo de rol en el Modelo Unificado; caso afirmativo se le asigna este estereotipo a la nueva asociación del Modelo Navegacional; caso negativo, se verifica en qué concern está contenida la asociación origen y se agrega el estereotipo correspondiente (<<navigationLink>> o <<walkingLink>>).

d. `transformRoleOf()`. Este método es invocado para transformar las asociaciones de rol que tiene el Modelo Unificado. Se puede apreciar en el Código

¹⁴³ Crea la asociación asignándole la cardinalidad y los nombres adecuados según la asociación origen. Además, se indica cuál es la visualización dentro del Modelo Navegacional.

B.27 la definición de este método (es muy similar a cómo se tratan las asociaciones simples como se mostró en el Código B.10). Se busca en todos los elementos de visualización de la asociación uno que sea *AssociationView* y se invoca al método `transformAssociationRole()`. No se puede usar el mismo método que para las asociaciones comunes ya que las asociaciones de rol deben visualizarse diferente respecto del Modelo Unificado. Esto se debe a que en el Modelo Navegacional las clases constan de estereotipos que dan la semántica de nodos y por lo tanto se genera una variación en el tamaño de las mismas. Cuando se dibuja la asociación de rol entre nodos esto debe ser considerado para que queden bien conectados.

```
private void transformRoleOf(HashMap<Element,
    List<PresentationElement>> elementsInDiagram, Package destPackage, Element element) {
    for (PresentationElement assoc : elementsInDiagram.get(element)) {
        if (MagicDrawAndMobileUWEOperations.instanceOfAssociationView(assoc)) {
            int i = getDiagramTransformator().transformAssociationRole(assoc, destPackage);
            setNumberOfTransformedAssociations(getNumberOfTransformedAssociations()+ i);
        }
    }
}
```

Código B.27: Método para transformar la relación de rol.

En la Figura B.8 se puede observar un diagrama de secuencia (reducido) que se corresponde con la transformación de los roles. Se puede observar que se invoca al método `transformAssociationRole()` y éste es el encargado de invocar al método `addConvertedStereotypeToAssociation()`. Este último método es el mismo que se describió en el Código B.26, que dependerá de si se invoca desde el método que crea la asociación de rol o desde el método que crea una asociación común cuál será el estereotipo que agregue a la asociación creada en el Modelo Navegacional.

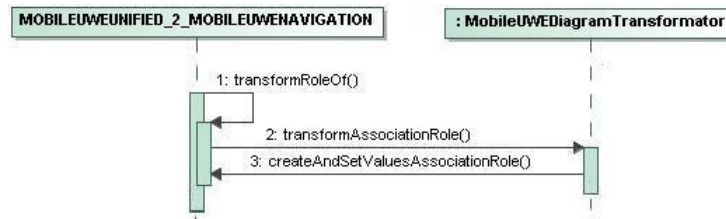


Figura B.8: Diagrama de secuencia reducido del método para crear el rol.

El método `transformAssociationRole()` se especifica en el Código B.28. Se puede observar que es similar al presentado en el Código B.25 pero varía el método que continua la transformación.

```
int transformAssociationRole(PresentationElement origAssoc, Package destPackage) {
    PathElement pathEl = (PathElement) origAssoc;
    PresentationElement origClient = getPath2origClient().get(pathEl);
    PresentationElement newClient = getTransformedClassElements().get(origClient);
    PresentationElement origSupplier = getPath2origSupplier().get(pathEl);
    PresentationElement newSupplier = getTransformedClassElements().get(origSupplier);
    List<PresentationElement> firstConnectors = new ArrayList<PresentationElement>();
    List<PresentationElement> secondConnectors = new ArrayList<PresentationElement>();
    calculateSubclasses(origClient, newClient, origSupplier, newSupplier, firstConnectors, secondConnectors);
    int i = 0;
    for (PresentationElement firstConnector: firstConnectors){
        for (PresentationElement secondConnector: secondConnectors){
            if (createAndSetValuesAssociationRole(origAssoc, destPackage, pathEl, firstConnector,
                secondConnector))
                i++;
        }
    }
    return i;
}
```

Código B.28: Creación del nuevo rol.

Se puede apreciar en el Código B.28 que, para este caso particular, se invoca al método `createAndSetValuesAssociationRole()`¹⁴⁴, desde el cual se llama al método `addConvertedStereotypeToAssociation()`.

B.4. Métodos invocados por las transformaciones

En esta sección se detallarán algunos métodos usados por las transformaciones descritas en las Secciones B.1, B.2 y B.3. En el Código B.29 se puede ver la definición del método `collectElementsAndPresentationElements()` de la clase *MobileUWEDiagramType*. Es un método recursivo que se mete en profundidad en la estructura que tiene el elemento de visualización pasado como parámetro (`presentationElement`), el cual puede ser un elemento simple o más complejo como, por ejemplo, un modelo. Este método guarda en la variable `elements` cada elemento de *UML* contenido en `presentationElement` junto a todos sus elementos de visualización. Como ya se ha mencionado cada elemento de *UML* tiene varios elementos de visualización asociados.

```
public static void collectElementsAndPresentationElements(PresentationElement presentationElement,
    HashMap<Element, List<PresentationElement>> elements) {
    List<PresentationElement> ownedPresentationElements = presentationElement.getPresentationElements();
    for (PresentationElement pe : ownedPresentationElements) {
        Element element = pe.getElement();
        if (element != null) {
            List<PresentationElement> oldList = elements.get(element);
            if (oldList != null) {
                oldList.add(pe);
                elements.put(element, oldList);
            } else {
                LinkedList<PresentationElement> newList = new LinkedList<PresentationElement>();
                newList.add(pe);
                elements.put(element, newList);
            }
        }
        collectElementsAndPresentationElements(pe, elements);
    }
}
```

Código B.29: Obtener todos los elementos de un diagrama.

Para obtener los concerns digitales que se listan para seleccionar cuál es el *Core* de la aplicación (para aquellas transformaciones que generan un Modelo Unificado) se usa el método `collectDigitalConcerns()`. Este método está definido en las transformaciones presentadas en las Secciones B.1 y B.2 y su definición se puede visualizar en el Código B.30.

```
private void collectDigitalConcerns(HashMap<Element, List<PresentationElement>> elementsInDiagram) {
    for (Element element : elementsInDiagram.keySet()) {
        if (MagicDrawAndMobileUWEOperations.instanceOfPackage(element))
            if (StereotypesHelper.hasStereotype(element,
                MobileUWEStereotypePackage.DIGITAL_CONTENT.getName())
                && ((Package)element).getName().length() > 0) {
                addDigitalConcern((Package)element);
            }
    }
}
```

Código B.30: Obtener los nombres de los diagramas para listarlos.

¹⁴⁴ Crea la asociación asignándole la cardinalidad y los nombres adecuados. Además se indica cuál es la visualización dentro del modelo teniendo en cuenta que las clases agregaron el estereotipo adecuado para dar la semántica de nodo variando su tamaño respecto del Modelo Unificado. Por lo tanto, la visualización de las asociaciones que representan al rol deben ajustar su visualización. El método `createAndSetValuesAssociationRole()` varía del `createAndSetValuesAssociation()` (invocado en el Código B.25) respecto a como trata la visualización de la asociación.

Se puede observar (Código B.30) que se hace el control que el concern tenga nombre definido un nombre, así en la lista que se le muestra al usuario sólo aparecen nombres que puede seleccionar.

En particular la transformación presentada en la Sección B.2 debe además listar los concerns digitales que no son el *Core* para que el diseñador elija cuál de estos son de interés. El método que define esta transformación es `selectOtherConcerns()`, este se puede visualizar en el Código B.31. Se puede apreciar que se usa la ventana *CkeckListGUISelector*, que contiene los concerns seleccionados. Todos los concerns seleccionados son agregados a la variable `selectedDigitalConcerns` mediante el método `setSelectedDigitalConcerns()`.

Para poder obtener los nombres de los concerns almacenados en la variable `digitalConcerns` se usa el método `getDigitalConcernsNames()`. De esta manera se pueden listar los nombres de los concerns digitales para que el diseñador pueda elegir uno como *Core*.

```
private boolean selectOtherConcerns(int index) {
    Vector<String> listElementName = getDigitalConcernsNames();
    listElementName.remove(index);
    getDigitalConcerns().remove(index);
    if (listElementName.size() > 0) {
        CheckListGUISelector gui = new CheckListGUISelector(listElementName,
            MobileUWEGlobalConstants.getSELECT_OTHER_CONCERNS());
        if (gui.isNotCancel()) {
            setSelectedDigitalConcerns(gui);
        } return false;
    }
    return true;
}
```

Código B.31: Seleccionar los concerns que son de interés (sólo para la transformación del Modelo de Contenido al Modelo Unificado que selecciona concerns de interés).

El método usado por las transformaciones presentadas en las Secciones B.1 y B.2 para asignar el Modelo de Contenido al estereotipo (`<<mobileUWEUnifedDiagram>>`) del Modelo Unificado es denominado `setContentDiagram()`. Este método se puede visualizar en el Código B.32.

Se puede observar que se recupera el *Slot* que se corresponde con el valor etiquetado del estereotipo y luego a este *Slot* se le asigna como valor el Modelo de Contenido que se transformo.

```
private void setContentDiagram(DiagramPresentationElement sourceDiagram, Diagram destDiagram) {
    Stereotype stereotype = StereotypesHelper.getStereotype(Application.getInstance().getProject(),
        MobileUWEDiagramType.MOBILEUWEUNIFIED.getDiagramStereotype());
    Property property = StereotypesHelper.getPropertyByName(stereotype,
        MobileUWEStereotypesGlobalConstants.getUNIFIED_TAG_CONTENT_DIAGRAM());
    if (stereotype != null && property != null) {
        Slot slot = StereotypesHelper.getSlot(destDiagram, property, true, false);
        if (slot != null) {
            ElementValue value = Application.getInstance().getProject().
                getElementFactory().createElementValueInstance();
            value.setElement(sourceDiagram.getElement());
            slot.getValue().add(value);
        }
    }
}
```

Código B.32: Setear un Modelo de Contenido en el estereotipo del Modelo Unificado.

La transformación Sección B.3 usa para asignar el Modelo Unificado al estereotipo del Modelo Navegacional (`<<mobileUWENavigationDiagram>>`) el método `setUnifiedDiagram()`. Este método se puede visualizar en el Código B.33. Se puede observar que se recupera el *Slot* que se corresponde con el valor etiquetado del estereotipo y luego a este *Slot* se le asigna como valor el Modelo Unificado que se transformo.

```

private void setUnifiedDiagram(DiagramPresentationElement sourceDiagram, Diagram destDiagram) {
    Stereotype stereotype = StereotypesHelper.getStereotype(Application.getInstance().getProject(),
        MobileUWEDiagramType.MOBILEUWENAVIGATION.getDiagramStereotype());
    Property property = StereotypesHelper.getPropertyByName(stereotype,
        MobileUWEStereotypesGlobalConstants.getNAVIGATION_TAG_UNIFIED_DIAGRAM());
    if (stereotype != null && property != null) {
        Slot slot = StereotypesHelper.getSlot(destDiagram, property, true, false);
        if (slot != null) {
            ElementValue value = Application.getInstance().getProject().getElementsFactory().
                createElementInstance();
            value.setElement(sourceDiagram.getElement());
            slot.getValue().add(value);
        }
    }
}

```

Código B.33: Setear un Modelo Unificado en el estereotipo del Modelo Navegacional.

ANEXO C

CREACION DE MODELOS EN *MAGICMOBILEUWE*

En este anexo se detalla cómo se realiza la creación de cada uno de los modelos del enfoque. Se muestra que el Modelo de Contenido, Unificado y Navegacional se crean si elementos. El Modelo de Instancias Navegacional se crea con los concerns correspondientes al Modelo Navegacional elegido de base. En cuanto a la implementación particular del Modelo de Presentación definida en esta tesis se muestra que en la creación del mismo se definen los concerns correspondientes al Modelo Navegacional elegido de base. Además, se muestra que este modelo es creado con las clases presentación de los nodos (digitales o físicos). El Modelo del Usuario se crea con las estrategias navegacionales definidas por default y además se le agrega los Modelos de Instancias Navegacional y de presentación elegidos por el diseñador.

C.1. Modelo de Contenido, Unificado y Navegacional

Como se mencionó en el Anexo A, los Modelos de Contenido, Unificado y Navegacional se representan mediante la clase *MobileUWEDiagramType*. Esta clase se define como un tipo enumerativo con tres elementos:

- MOBILEUWECONTENT
- MOBILEUWEUNIFIED
- MOBILEUWENAVIGATION

La clase *MobileUWEDiagramType* define el siguiente constructor:

```
MobileUWEDiagramType(String name, String originalPackName,  
                    String packageNamePart, String umlDiagramType,  
                    String modelStereotype, String diagramStereotype)  
name                // Nombre del modelo  
originalPackName   // Nombre del paquete que lo contiene  
packageNamePart    // Abreviatura del paquete que lo contiene  
umlDiagramType     // Nombre del tipo del modelo  
modelStereotype    // Nombre del estereotipo del paquete contenedor  
diagramStereotype  // Nombre del estereotipo del modelo
```

La clase *MobileUWEDiagramType* define el método `createAndAddDiagram()` como se visualiza en el Código C.1. Según cuál de los elementos (Modelos de Contenido, Unificado o Navegacional) recibe este mensaje es el modelo que se crea. El modelo creado no contiene ningún elemento.

Primero se controla (Código C.1) que si no llega un nombre que el usuario haya elegido se le asigna el nombre establecido por default. Luego hay que controlar si ese no existe. En caso de existir se le agrega un valor incremental al final del nombre por default.

La clase *ModelElementsManager* (de la API de *MagicDraw*) permite crear¹⁴⁵ un modelo en el paquete que llega como parámetro mediante el método `createDiagram()`. El modelo creado queda almacenado en la variable `diag`. Luego se le agrega al modelo el estereotipo adecuado¹⁴⁶. Finalmente se abre el modelo creado en la pantalla de *MagicDraw*. Los últimos controles son para setear en el caso de que el diseñador haya indicado un nombre particular, que este sea asignado al

¹⁴⁵ Usando el método `getUmlDiagramType()` se determina cuál es el modelo en particular que se debe crear.

¹⁴⁶ El método `getDiagramSterotypeType()` se determina cuál es el estereotipo asociado a cada modelo, cada elemento enumerativo define el estereotipo que tiene asociado.

modelo creado. Además se controla que el modelo haya quedado contenido en algún paquete evitando inconsistencias en la creación.

```

public synchronized Diagram createAndAddDiagram(Package destinationPackage, String diagramName) {
    // A diagram should have a name
    if (diagramName == null || diagramName.trim().equals(MobileUWEStringsGlobalConstants.getEMPTY()) {
        diagramName = getNameWithDiagramTail();
    }
    String diagName = getNameWithoutDiagramTail();
    // If there would be duplicates, add a number
    diagramName = diagramName + getDiagramNameExtension(destinationPackage, diagramName,
        MobileUWEStringsGlobalConstants.getEMPTY());
    Project project = Application.getInstance().getProject();
    SessionManager.getInstance().createSession(MobileUWEGUIGlobalConstants.getCREATE_AND_OPEN() + diagramName);
    Diagram diag = null;
    try {
        diag = ModelElementsManager.getInstance().createDiagram(getUmlDiagramType(), destinationPackage);
        StereotypesHelper.addStereotypeByString(diag, getDiagramStereotype());
        // Open diagram (automatically makes it to the active one)
        project.getDiagram(diag).open();
    } catch (ReadOnlyElementException e) { e.printStackTrace();}
    finally { SessionManager.getInstance().closeSession();}
    // Set the name at the first time does not work with ProcessFlow diagram
    SessionManager.getInstance().createSession(MobileUWEStringsGlobalConstants.getSET_NAME_TO() + diagramName);
    if (diag != null) {
        diag.setName(diagramName);
        // This if is because of the undo-case when MagicDraw has detected a model inconsistency
        if (diag.getOwner() != null) {
            MobileUWEMessageWriter.log(MobileUWEStringsGlobalConstants.getDIAGRAM() + diagName +
                MobileUWEStringsGlobalConstants.getWAS_CREATE_STORED_IN()
                + diag.getOwner().getHumanName() + MobileUWEStringsGlobalConstants.getAND_OPEN());
        }
    }
    SessionManager.getInstance().closeSession();
    return diag;
}

```

Código C.1: Creación del Modelo de Contiendo, Unificado o Navegacional.

C.2. Modelo de Instancias Navegacional

La clase *MobileUWEInstanceDiagramType* representa a los Modelos de Instancias Navegacional. También se define como un enumerativo¹⁴⁷ que cuenta con un sólo elemento llamado `MOBILEUWENAVIGATIONINSTANTIATION`. El constructor de la clase se define de la misma manera que la clase presentada en la Sección C.1 con los mismos parámetros. La definición del método `createAndAddDiagram()` es similar al Código C.1, en el Código C.2¹⁴⁸ se muestra solamente la parte que varía.

```

public synchronized Diagram createAndAddInstanceDiagram(Package destinationPackage, String diagramName) {
    Diagram diag = null;
    Vector<String> names = getDiagramsName();
    if (names.size() > 0) {
        int index = (new MobileUWEGUISelector(names,
            MobileUWEGUIGlobalConstants.getSELECT_NAVIGATION_DIAGRAM())) .getIndex();
        if (index != -1) {
            ...
            try {
                diag = ModelElementsManager.getInstance().createDiagram(getUmlDiagramType(), destinationPackage);
                setNewDiagramValuesAndPackage(destinationPackage, diag, names, index);
                // Open diagram (automatically makes it to the active one)
                project.getDiagram(diag).open();
            } catch (ReadOnlyElementException e) {e.printStackTrace();}
            finally {SessionManager.getInstance().closeSession();}
            ...
        }
    }
    else {MobileUWEMessageWriter.showMessage(MobileUWEGUIGlobalConstants.getNO_NAVIGATION_DIAGRAM(),
        getLogger());}
    return diag;
}

```

Código C.2: Creación del Modelo de Instancias Navegacional.

¹⁴⁷ La herramienta está definida con un sólo Modelo de Instancias Navegacional pero podrían generarse más elementos en futuras evoluciones.

¹⁴⁸ Los puntos suspensivos son para aquellas partes de código que son iguales al método presentado en la Sección C.1.

Se puede apreciar en el Código C.2 que se despliega una ventana (*MobileUWEGUISelector*) que permite seleccionar el Modelo Navegacional de base. El método `setNewDiagramValuesAndPackage()` se encarga de realizar el seteo del estereotipo correspondiente del modelo agregando el Modelo Navegacional de base y además crea los concerns acordes al modelo de base.

En el Código C.3 se puede apreciar la definición del método `setNewDiagramValuesAndPackage()`. Primero se agrega al modelo el estereotipo correspondiente (`<<mobileUWENavigationInstanceDiagram>>`). Luego se busca el Modelo Navegacional que se corresponde con el nombre que selecciono el diseñador. Una vez encontrado, éste es almacenado en el valor etiquetado (`navigationDiagram`) del estereotipo `<<mobileUWENavigationInstanceDiagram>>` usando el método `setNavigationDiagram()`. Luego se indica en el valor etiquetado `wasComplete` del estereotipo que el modelo no fue completado, esto se realiza usando el método `setWasComplete()`. Finalmente se crean los concerns acordes al Modelo Navegacional de base mediante el método `createPackage()`.

```
private void setNewDiagramValuesAndPackage(Package destinationPackage,
    Diagram diag, Vector<String> names, int index) {
    StereotypesHelper.addStereotypeByString(diag, getDiagramStereotype());
    Diagram navigationDiagram = null;
    DiagramPresentationElement navigationDiagramPresentationElement = null ;
    for (DiagramPresentationElement element : Application.getInstance().getProject().getDiagrams()){
        Diagram auxDiagram = (Diagram) element.getElement();
        if (auxDiagram.getName() == names.get(index)){
            navigationDiagram = auxDiagram;
            navigationDiagramPresentationElement = element;
        }
    }
    setNavigationDiagram(diag, navigationDiagram);
    setWasComplete(diag);
    createPackage(destinationPackage, diag, navigationDiagramPresentationElement);
}
```

Código C.3: Setear los valores correspondientes al Modelo de Instancias Navegacional.

En el Código C.4 se puede apreciar la definición del método `createPackage()`. Se obtienen primero todos los elementos del modelo y cada paquete encontrado se manda a transformar mediante el método `packageTransformation()`. Este método se define de manera muy similar a cómo se definían los métodos de las transformaciones cuando se tenían que crear desde un modelo origen. Creando así los concerns acordes al Modelo Navegacional de base.

```
private void createPackage(Package destinationPackage, Diagram diag,
    DiagramPresentationElement navigationDiagramPresentationElement) {
    HashMap<Element, List<PresentationElement>> elementsInDiagram =
        new HashMap<Element, List<PresentationElement>>();
    navigationDiagramPresentationElement.open();
    MobileUWEDiagramType.collectElementsAndPresentationElements(navigationDiagramPresentationElement,
        elementsInDiagram);
    for (Element element : elementsInDiagram.keySet()) {
        if (MagicDrawAndMobileUWEOperations.instanceOfPackage(element)) {
            packageTransformation(elementsInDiagram, destinationPackage, element, diag);
        }
    }
}
```

Código C.4: Crear los concerns en el Modelo de Instancias Navegacional.

C.3. Modelo de Presentación

La clase que representa los Modelos de Presentación se denomina *MobileUWELayoutDiagramType*, se define como un enumerativo¹⁴⁹ que cuenta con un

¹⁴⁹ La herramienta está definida con un sólo Modelo de Instancias Navegacional pero se podrían generar más elementos en futuras evoluciones.

sólo elemento llamado `MOBILEUWELAYOUT`. El constructor de la clase se define de la misma manera que la clase presentada en la Sección C.1 con los mismos parámetros. La definición del método `createAndAddDiagram()` se define igual al presentado en el Código C.2 pero el método `setNewDiagramValuesAndPackage()` cambia su definición.

En el Código C.5 se puede observar cómo se define el método `setNewDiagramValuesAndPackage()`. Primero se agrega al modelo el estereotipo adecuado (`<<mobileUWELayoutDiagram>>`).

Luego se busca el Modelo Navegacional que se corresponde con el nombre que selecciono el diseñador, una vez encontrado, éste es almacenado en el valor etiquetado `navigationDiagram` del estereotipo (`<<mobileUWELayoutDiagram>>`) usando el método `setNavigationDiagram()`. Finalmente se crean las clase de presentación y los concerns acordes al Modelo Navegacional de base mediante el método `createPackage()`.

```
private void setNewDiagramValuesAndPackage(Package destinationPackage, Diagram diag, Vector<String> names,
int index) {
    StereotypesHelper.addStereotypeByString(diag, getDiagramStereotype());
    Diagram navigationDiagram = null;
    DiagramPresentationElement navigationDiagramPresentationElement = null ;
    for (DiagramPresentationElement element : Application.getInstance().getProject().getDiagrams()){
        Diagram auxDiagram = (Diagram) element.getElement();
        if (auxDiagram.getName() == names.get(index)){
            navigationDiagram = auxDiagram;
            navigationDiagramPresentationElement = element;
        }
    }
    setNavigationDiagram(diag, navigationDiagram);
    createElements(destinationPackage, diag, navigationDiagramPresentationElement);
}
```

Código C.5: Setear los valores correspondientes en el Modelo de Presentación.

La definición del método `createPackage()` se puede apreciar en el Código C.6. En este método se deben crear los concerns y las clases de presentación adecuadas al Modelo Navegacional elegido de base. Primero se encuentran todos los elementos del Modelo Navegacional de base. Luego se recorren y por cada concern que se detecta se crea uno en el Modelo de Presentación. De cada concern se recorre sus elementos y por cada nodo que se detecta se crea su correspondiente clase de presentación. Sólo se crean clases de presentación para los nodos digitales o físicos, el resto de las clases que puede contener un concern no tiene clase de presentación asociada. Las creaciones de los nuevos elementos se realizan de la misma manera que en las transformaciones.

```
private void createElements(Package destinationPackage, Diagram diag,
DiagramPresentationElement navigationDiagramPresentationElement) {
    HashMap<Element, List<PresentationElement>> elementsInDiagram =
        new HashMap<Element, List<PresentationElement>>();
    navigationDiagramPresentationElement.open();
    MobileUWEDiagramType.collectElementsAndPresentationElements(navigationDiagramPresentationElement,
        elementsInDiagram);
    for (Element element : elementsInDiagram.keySet()) {
        if (MagicDrawAndMobileUWEOperations instanceof Package(element)) {
            Package transformedPackage = packageTransformation(elementsInDiagram, destinationPackage,
                element, diag);
            for (PackageableElement packageElement : ((Package)element).getPackagedElement())
                if (MagicDrawAndMobileUWEOperations instanceof ClassImpl(packageElement) &&
                    (StereotypesHelper.hasStereotype(packageElement,
                        MobileUWEStereotypesGlobalConstants.getNAVIGATION_CLASS_STEREOTYPE())
                    ||StereotypesHelper.hasStereotype(packageElement,
                        MobileUWEStereotypeClassNav.PHYSICAL_NODE.toString()))
                classTransformation(elementsInDiagram, transformedPackage, packageElement, diag);
        }
    }
}
```

Código C6: Crear los elementos adecuados en el Modelo de Presentación.

C.4. Modelo del Usuario

La clase *MobileUWEUserModelDiagramType* representa a los Modelos del Usuario. También se define como un enumerativo¹⁵⁰ que cuenta con un sólo elemento llamado *MOBILEUWEUSERMODEL*. El constructor de la clase se define de la misma manera que la clase presentada en la Sección C.1 con los mismos parámetros. La definición del método *createAndAddDiagram()* es similar al Código C.2 pero se deben considerar algunos detalles particulares. La definición de este método se puede apreciar en el Código C.7, se despliegan dos ventanas (*MobileUWEGUISelector*) que permiten al diseñador seleccionar el Modelo de Instancias Navegacional primero y luego el Modelo de Presentación. Los puntos suspensivos son para aquellas partes de código que son iguales al método presentado en C.2. Se puede ver que se crea el modelo correspondiente usando el método *createDiagram()* y luego se le agrega el estereotipo `<<mobileUWEUserModelInstanceDiagram>>`. El método *createInstances()* se encarga de crear las instancias particulares que contiene este modelo.

```
public synchronized Diagram createAndAddInstanceDiagram(Package destinationPackage, String diagramName) {
    Diagram diag = null;
    Vector<String> instanceDiagramsNames = getInstanceDiagramsName();
    if (instanceDiagramsNames.size() > 0){
        int index = (new MobileUWEGUISelector(instanceDiagramsNames,
            MobileUWEGUIGlobalConstants.getSELECT_INSTANCE_NAVIGATION_DIAGRAM()).getIndex());
        if (index != -1){
            Vector<String> layoutDiagramsNames = getLayoutDiagramsName(instanceDiagramsNames, index);
            if (layoutDiagramsNames.size() > 0){
                int index2 = (new MobileUWEGUISelector(layoutDiagramsNames,
                    MobileUWEGUIGlobalConstants.getSELECT_LAYOUT_DIAGRAM()).getIndex());
                if (index2 != -1){
                    ...
                    try {
                        diag = ModelElementsManager.getInstance().createDiagram(getUmlDiagramType(), destinationPackage);
                        StereotypesHelper.addStereotypeByString(diag,
                            MobileUWEDiagramsGlobalConstants.getSTEREOTYPE_USER_MODEL_INSTANCE_DIAGRAM());
                        createInstances(destinationPackage, diag, instanceDiagramsNames, index, layoutDiagramsNames, index2);
                        // Open diagram (automatically makes it to the active one)
                        project.getDiagram(diag).open();
                    } catch (ReadOnlyElementException e) {e.printStackTrace();}
                    finally {SessionManager.getInstance().closeSession();}
                    ...
                }
            } else{
                MobileUWEMessageWriter.showMessage(MobileUWEGUIGlobalConstants.getNO_LAYOUT_DIAGRAM(), getLogger());
            }
        } else{
            MobileUWEMessageWriter.showMessage(MobileUWEGUIGlobalConstants.getNO_INSTANCE_NAVIGATION_DIAGRAM(), getLogger());
        }
    }
    return diag;
}
```

Código C.7: Creación del Modelo del Usuario.

¹⁵⁰ La herramienta está definida con un sólo Modelo del Usuario pero se podrían generar más elementos en futuras evoluciones.