UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Representación de Espacios Indoor utilizando la Herramienta de Desarrollo de Metamodelos ADOxx
Autores: Poch, Karen - Rispoli, Paula
Director: Dra. Mostaccio, Catalina
Codirector: Dra. Pons, Claudia
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

Esta tesina ofrece un caso de estudio de un proceso de construcción de metamodelos, haciendo uso de una técnica iterativa e incremental y aplicando el lenguaje formal OCL. A partir de un análisis detallado sobre los espacios indoor, se diseñó el metamodelo abstracto "Espacio Indoor" que atravesó diferentes etapas en las que se muestra cómo el metamodelo puede ir mejorándose progresivamente a través de la aplicación de patrones de diseño y otras buenas prácticas de modelado, hasta obtener el metamodelo final. Siguiendo con un estudio semejante se definió el metamodelo concreto "Facultad" que hereda el comportamiento de "Espacio Indoor" y agrega los conceptos propios de las facultades. Para plasmar dicho metamodelo, se analizó la plataforma ADOxx con la que se desarrolló la herramienta denominada "Modelador de facultades" que permite modelar cualquier facultad que se desee. Como complemento se presenta un prototipo en el que, a partir de la importación de un archivo XML generado con la herramienta desarrollada, permite realizar ciertas consultas para mostrar que es posible el recorrido del espacio. Finalmente, con el objetivo de demostrar la validez del metamodelo "Facultad", se detalla la instanciación de una sección de la Facultad de Informática.

Palabras Claves

Espacio Indoor - Metamodelo - Lenguajes de modelado - UML - OCL - Extensión de metamodelos - Herencia conservativa - Herramientas de metamodelado - ADOxx - OMI - Representación visual - Algoritmos de búsqueda - Posicionamiento - Recorrido de espacios

Conclusiones

Esta tesis ofrece un aporte empírico a la disciplina denominada "Ingeniería de metamodelos". Además de mostrar cómo es posible mejorar la construcción de metamodelos a través de una técnica iterativa e incremental, también presenta la forma en que un metamodelo puede estructurarse de manera que los conceptos abstractos puedan reutilizarse en varios dominios concretos. Adicionalmente se muestra que es posible definir reglas de buena formación de modelos e incorporar conceptos semánticos propios del dominio a través de la incorporación de restricciones OCL.

Trabajos Realizados

Diseño de metamodelo para representación de Espacios Indoor.
Extensión del metamodelo de Espacios Indoor para un dominio particular, las facultades.
Desarrollo en plataforma ADOxx de herramienta "Modelador de facultades".
Desarrollo de prototipo "Buscador de Caminos" para recorrer un espacio modelado.
Utilización de la herramienta instanciando una sección de la Facultad de Informática de la UNLP.

Trabajos Futuros

Estudio para obtener coordenadas geográficas reales de los ambientes dentro de un Espacio Indoor.
Estudio de sensores de posicionamiento para retornar posición exacta del usuario dentro del Espacio Indoor.
Incorporar algoritmos de búsqueda dentro de la herramienta "Modelador de Facultades"
Desarrollo móvil de la herramienta "Modelador de Facultades"
Incluir audio en los resultados obtenidos de la búsqueda de caminos para mejorar accesibilidad.

Fecha de la presentación: Octubre de 2015

A mi mamá.
A mi hermana.
A mis abuelos Palmira y Tito.
Karen Poch

A Guillermo, mi papá.
A Iván, mi hermano.
Ambos estarían orgullosos de mí.
Paula Rispoli

Índice de contenidos

Capítulo 5 - Herramientas de metamodelado y ADOxx

Capítulo 6 - Implementación

Índice de figuras

Capítulo 1
Introducción

Capítulo 1 - Introducción

En este capítulo se presenta, en primer lugar, la motivación que nos ha llevado al desarrollo de esta tesina junto con los objetivos que se persiguen. En segundo lugar se presenta la estructura, describiendo de manera concisa el contenido de la misma.

1.1 Objetivo

Uno de los objetivos principales de esta tesina es analizar todos los conceptos generales involucrados en la representación de espacios indoor, identificando los conceptos comunes, para finalmente poder lograr la definición de un metamodelo abstracto que permita realizar la instanciación de los mismos.

Por otro lado, se realiza una investigación sobre la herramienta de desarrollo de metamodelos denominada ADOxx**,** implementada en la Universidad de Viena, Austria; con el objetivo de definir el metamodelo mencionado anteriormente.

1.2 Motivación

1.2.1 Carencia en representaciones existentes de Espacios Indoor

La mayor parte de las personas pasan en torno a un 80%-90% de su tiempo en ambientes internos: viven en casas, trabajan en edificios, visitan clínicas u hospitales, estudian en universidades, etc. [1,2]. Sin embargo, mientras el posicionamiento en espacios outdoor se considera ya una tecnología relativamente madura, su contrapartida en espacios indoor está aún en una fase de estudio y poco desarrollado.

Podríamos definir ciertas situaciones en las que el uso del posicionamiento en espacios interiores sería enriquecedor:

- en un museo de arte, ofrecer información de la ubicación de una imagen determinada en función de la posición del visitante;
- en una universidad, donde hay múltiples pisos y pasillos, mostrar dónde se encuentra cada aula y cómo llegar hasta cada una de ellas;
- ante una emergencia en un edificio, detectar la posición de las salidas de emergencia e indicar el camino hacia la más cercana;
- en un hospital, cuando un paciente tiene una emergencia, detectar la ubicación de la sala de guardia [3].

Hoy en día consultando, por ejemplo, los sitios web de algunos espacios indoor, podemos obtener una imagen que nos muestre el mapa del mismo. Si logramos interpretar esa imagen vamos a poder localizar por nuestros propios medios, los lugares que nos interesen visitar dentro del espacio. Actualmente, con la tecnología existente en el mercado, se podría brindar al usuario una asistencia más completa acerca de cómo movilizarse dentro del espacio de su interés. Por ejemplo, ofrecerle

información sobre los puntos de interés existentes e indicarle el camino más corto para llegar a ellos, en base a su posición actual [4]. Para poder lograr una aplicación que provea este tipo de funcionalidades, es necesario obtener información del espacio que la imagen mencionada anteriormente no provee. Es necesario definir un modelo que logre representar de alguna manera el espacio para poder obtener esta información. La finalidad de dicho modelo es permitir seleccionar la ubicación de puntos de acceso en el interior del espacio, los cuales pueden tener varios niveles y varios ambientes en cada nivel.

1.2.2 Carencia en la definición de un modelo único: El Metamodelo

Un metamodelo define la sintaxis abstracta de un lenguaje que establece los conceptos y relaciones entre ellos, e incluye las reglas que determinan qué es un modelo bien formado.

En otras palabras, el metamodelo de un dominio específico debe realizar una abstracción de los elementos estáticos y dinámicos de los modelos que se pueden instanciar en la realidad para, de esta forma, permitir crear nuevos modelos que retomen las características genéricas y las personalicen al ámbito según los requerimientos específicos [5].

De acuerdo al estudio que hemos realizado acerca de modelos de representación de espacios indoor se puede decir que no existe una definición genérica de los mismos, es decir, un *metamodelo* que reúna los conceptos y características generales de ellos.

1.2.3 Herramienta de desarrollo de metamodelos: ADOxx

Se puede definir a las herramientas de desarrollo de metamodelos como las herramientas básicas para diseñar el metamodelo de acuerdo al lenguaje de metamodelado [6].

Las herramientas que permiten la definición de lenguajes de modelado, incorporan mecanismos para:

- ✓ crear metamodelos;
- ✓ asociar una notación a los elementos de un metamodelo;
- ✓ expresar transformaciones para la generación de modelos a partir de un modelo que conforme a cierto metamodelo;
- ✓ realizar consultas sobre los metamodelos creados.

El componente de estas herramientas que permite crear un metamodelo y asociarle una sintaxis concreta se denomina *editor de metamodelos.*

La aplicación de herramientas de metamodelado en un proyecto puede traer consigo varias ventajas, entre las que se encuentran:

- **Reducción en tiempo y recursos para el mantenimiento de las aplicaciones existentes**

  Todo desarrollador de software sabe que los productos que realizan, por representar modelos de lo que ocurre en el mundo real, son cambiantes y dinámicos, por lo que deben ser adaptables a las nuevas exigencias de los clientes o beneficiarios institucionales, esto conlleva a realizar un proceso que involucra grandes cambios. La aplicación del metamodelado, implica que los pasos a seguir para realizar la misma modificación sean menores. Este cambio en el paradigma del mantenimiento de las aplicaciones genera sustanciales beneficios a la organización que toma la decisión de adoptar esta tecnología para la construcción y mantenimiento de sus sistemas de información.

- **Evita la introducción de errores en los programas**

  La capacidad de introducir una nueva funcionalidad en un sistema de información sin escribir líneas de código adicional, elimina la posibilidad de introducir errores de programación cuyo costo tanto para la información registrada (errores en la base de datos a causa de un programa erróneo) o el costo de ubicar, corregir y probar el código fuente causante del error, son eliminados.

- **Reducción en el tiempo de entrenamiento a los usuarios**

  Luego de entrenar a un usuario en la utilización de estas herramientas, el entrenamiento para que aprenda a utilizar la aplicación del metamodelo para otras aplicaciones, se reduce a trabajar con éste sobre los formatos, grupos y campos de información con los cuales se ha estructurado la información en el nuevo contexto. Esto es gracias a la unicidad de las interfaces en el software de registro y consulta de información.

- **Generación de consultas a la medida**

  El sistema generador de consultas para un metamodelo, se convierte en una herramienta muy poderosa en manos de las personas que dominan el contexto temático de la información registrada en este metamodelo, ya que puede consultar, ordenar, agrupar, graficar o producir información alfanumérica para la información registrada.

Esta tesina se realiza en el marco del "Proyecto de Colaboración" de la Facultad de Informática de la UNLP, con la Universidad de Viena en Austria, denominado "ADOxx Metamodel Compiler" que tiene por objetivo contribuir con la iniciativa internacional OMI (Open Model Initiative) [7].

ADOxx es una herramienta de metamodelado que permite la definición de los diferentes metamodelos en un entorno gráfico con las mismas características que emplea el usuario en la construcción de los diferentes modelos. De esta manera, se puede definir cualquier tipo de metamodelo en términos de las entidades que forman parte del mismo y sus posibles interconexiones o relaciones. Una vez definido el

metamodelo, se puede emplear su definición para construir los modelos pertinentes a un problema específico del mundo [8].

1.3 Organización de la tesina

Esta tesina se ha dividido en un total de 7 (siete) capítulos. Además, la misma consta con un anexo sobre el patrón de diseño "Composite", una sección de definición de acrónimos, y por último otra sección en la que se mencionan las citas bibliográficas.

Capítulo 1: Introducción

El primer capítulo sirve como introducción y brinda una visión general acerca del contenido de la tesina. Se definen los objetivos, la motivación que nos lleva a realizar este estudio, los resultados que esperamos obtener al concluir el trabajo de grado, y finalmente se presenta la estructura organizacional de la tesina.

Capítulo 2: Espacios Indoor-Outdoor

Este capítulo presenta un gran aporte introductorio para el posterior análisis que se realiza en el capítulo 4. Aquí se presenta una definición acerca de los espacios en general, junto con una descripción más detallada sobre los espacios indoor, que son la base de nuestro estudio. De éstos últimos se enuncian sus características y diferentes clasificaciones existentes.

A su vez se presenta una visión general sobre los espacios outdoor, indicando sus características principales. Finalmente se realiza una comparación entre ambos tipos de espacios en donde se analizan similitudes y diferencias entre ellos.

Además, se analiza el concepto de posicionamiento y se definen las tecnologías que se utilizan en la actualidad para lograr el posicionamiento de objetos o personas, tanto en espacios outdoor como en espacios indoor. Este último se encuentra en la etapa de desarrollo.

Capítulo 3: Técnicas para la definición de lenguajes

Aquí se presenta otro de los conceptos principales relacionados con el desarrollo de esta tesina. Se define el propósito y las características del metamodelado, junto con la definición de metamodelo; así como también las diferentes metodologías existentes para definir lenguajes.

Se describen los conceptos y propósitos principales de los diferentes mecanismos para la definición de lenguajes, en particular se presenta una investigación más exhaustiva acerca de los lenguajes gráficos, los cuales se utilizan para definir los metamodelos, que son la base de nuestro estudio.

Capítulo 4: Representación de Espacios Indoor

Este capítulo aborda uno de los temas principales de la tesina. En principio, y como manera introductoria para que el lector pueda tener un óptimo entendimiento del

metamodelo propuesto, se describen los dos lenguajes que utilizamos para la generación del mismo: UML y OCL. Se brinda una breve explicación sobre ellos y se presentan características y motivos que llevaron a la utilización de cada uno.

Como sección principal, se detalla todo el análisis realizado para la definición y obtención del metamodelo que proponemos como final. Se enumeran los conceptos comunes en todo espacio indoor y se especifica el comportamiento de cada uno de ellos junto con las características que los definen.

A su vez, se presenta cada etapa por las que pasó la definición del metamodelo hasta llegar al metamodelo final, mostrando un mecanismo de construcción iterativo e incremental. En cada una de las etapas se explican las decisiones tomadas, los refinamientos realizados y las justificaciones pertinentes que abalan los cambios plasmados, mostrando que el metamodelo puede ir mejorándose a través de la aplicación de patrones de diseño y otras buenas prácticas de modelado.

En este capítulo también se estudia la manera de estructurar un metamodelo de forma tal que los conceptos abstractos comunes puedan ser reutilizados en cualquier dominio concreto diferente. Para ello, a partir del refinamiento del metamodelo "Espacios Indoor" (metamodelo abstracto), se creó el metamodelo "Facultad" (metamodelo concreto) que reutiliza todos los conceptos comunes.

Por último se describen en el lenguaje OCL las restricciones que deben cumplir los metamodelos finales, enriqueciendo aún más la definición propuesta.

Capítulo 5: Herramientas de metamodelado y ADOxx

Existen varias herramientas para realizar metamodelado, es decir, herramientas para diseñar el metamodelo de acuerdo al lenguaje de metamodelado. En este capítulo se explica el uso de las mismas, características y ventajas de su uso; como así también se enuncian ciertas herramientas que fueron analizadas.

Particularmente se realiza un análisis sobre la plataforma ADOxx, herramienta utilizada para la implementación del metamodelo obtenido, explicando los diferentes entornos que posee para la definición de herramientas de modelado.

Capítulo 6: Implementación

En este capítulo se expone la implementación del metamodelo obtenido en el capítulo 4, realizada sobre la herramienta explicada en el capítulo anterior, ADOxx. Aquí se muestra la herramienta resultante desarrollada, junto con una explicación de la utilización de la misma.

Como complemento se presenta un prototipo de herramienta desarrollado en PHP, que tiene como objetivo realizar las validaciones de restricciones que no pudieron ser resueltas en la herramienta ADOxx, y mostrar la validez del metamodelo propuesto, debido a que permite aplicar algoritmos de búsqueda de caminos con el objetivo de mostrar que es posible recorrer el espacio instanciado.

Finalmente, se utilizan las herramientas analizando un caso de estudio y, por consiguiente, se realiza la instanciación de un espacio indoor en particular. Para nuestro estudio hemos elegido una sección de la Facultad de Informática; de esta manera se podrán analizar características cada herramienta.

Capítulo 7: Conclusiones

En este capítulo se plasman las conclusiones finales arribadas. Se busca compartir tanto enfoques objetivos como subjetivos.

La intención general del apartado es integrar los conceptos aprendidos en los capítulos iniciales, con lo expuesto en la definición del metamodelo para espacios indoor. A su vez, se presentan trabajos relacionados que impulsaron al desarrollo de esta tesina, los cuales fueron estudiados y analizados para la composición de la misma.

Finalmente, se describen los aportes que esta tesina ofrece al mundo de la informática y plantean y describen posibles trabajos futuros que puedan realizarse basándose en el aporte de este trabajo de grado.

Capítulo 2

Espacios Indoor - Outdoor

Capítulo 2 - Espacios Indoor - Outdoor

2.1 Introducción

En este capítulo se discuten los conceptos de los Espacios Indoor y los Espacio Outdoor, junto con las características de cada uno de ellos.

Luego, se realiza una comparación de ambos espacios, analizando similitudes y diferencias entre ellos y finalmente, se presenta una clasificación de los espacios en general, con la intención de completar las definiciones anteriormente mencionadas.

2.2 Espacios Indoor

2.2.1 Definición

Se pueden definir los Espacios Indoor como cualquier espacio cerrado construido por el hombre, tal como edificios, casas, hospitales, universidades, etc., excluyendo los espacios naturales, tales como cuevas, parques, etc., que son lo que comúnmente denominamos Espacios Outdoor [9].

Hay que notar que en distintas partes del mundo existen espacios abiertos que pueden ser construidos por el hombre, tales como parques artificiales, reservas naturales, etc.; en tal caso estos espacios serían Espacios Outdoor, ya que pertenecen al espacio exterior.

2.2.2 Características

Un Espacio Indoor cumple con las siguientes características [10]:

- Es un **Espacio Construido**, es decir, hecho por el hombre, que se compone de superficies rectilíneas (por ejemplo, paredes).
- Es **Cerrado** ya que está compuesto por paredes, techo, piso.
- Se puede observar su dimensión mirando a su alrededor, por ejemplo, si hablamos de un edificio, con solo levantar la vista podemos observar cuantos pisos tiene sin necesariamente conocer qué compone cada nivel.
- Contiene **objetos de pequeña escala** que varían en tamaño, desde lo suficientemente pequeño como para recoger (por ejemplo, lápices) hasta **objetos más grandes que el cuerpo humano** (por ejemplo, escritorios).

Un conjunto de Espacios Indoor generalmente está:

- conectado a través de jerarquías de contención (por ejemplo, las habitaciones y los pasillos están dentro de los límites de los edificios)
- modelados con sólidos geométricos o con representaciones de contorno en 2D de los espacios en 3D

2.3 Espacios Outdoor

2.3.1 Definición

Se pueden definir los Espacios Outdoor como espacios geográficos abiertos de gran escala, ubicados en el exterior, al aire libre.

Por lo general son ambientes naturales (parques, bosques, cuevas), aunque también podría incluir ambientes artificiales (tales como bosques o reservas naturales creadas por el hombre).

También se encuentran dentro de estos espacios las carreteras, los caminos, las autopistas, etc.; estos elementos están sumamente estudiados y analizados, ya que el mayor ejemplo de posicionamiento de Espacios Outdoor es la navegación por satélite GPS que se basa en estos elementos.

2.3.2 Características

Los Espacios Outdoor son espacios geográficos a gran escala, como los modelados en los Sistemas de Información Geográfica (GIS), y contiene una mezcla de ambientes naturales y artificiales (por ejemplo, los bosques frente a las ciudades).

Un Espacio Outdoor generalmente cumple con las siguientes características:

- Es un **espacio no construido**: dentro de él existen pocos objetos realizados por el hombre, con características irregulares, por ejemplo, las líneas de árboles que no son lineales.
- Es **abierto**: no tiene techo. Por lo general están ubicados al aire libre, en el exterior.
- Son espacios de **gran escala**: es difícil ver todo el espacio sin realizar un movimiento considerable.
- Contiene objetos que varían en tamaño desde lo suficientemente pequeño como para recoger (por ejemplo, los frutos de árboles), hasta objetos mucho más grandes que el cuerpo humano (por ejemplo, los árboles).

Un conjunto de Espacios Outdoor suele estar:

- Conectado a través de redes (por ejemplo, rutas, caminos, ríos)
- Modelado en 2,5 dimensiones (por ejemplo, 2 representaciones geométricas tridimensionales con un tercer atributo como altitud)

2.4 Espacios Indoor vs. Espacios Outdoor

En esta sección se describen las principales diferencias que existen entre los dos tipos de espacios.

Un espacio indoor es esencialmente diferente a un espacio outdoor en varios aspectos [1,2]:

- Los ***espacios indoor*** son espacios artificiales, construidos por el hombre, mientras que, por lo general, los ***espacios outdoor*** son espacios creados por la naturaleza.

- Los ***espacios indoor*** son espacios cerrados, es decir que están delimitados por paredes, techos, puertas, etc.; los ***espacios outdoor*** son espacios abiertos, ubicados en el exterior, sin paredes ni techos.

- Las mediciones en ambos espacios son distintas: en el ***espacio outdoor***, se suele utilizar el espacio euclidiano o una red espacial; mientras que el ***espacio indoor*** se relaciona con el concepto de espacio de celdas.

  El ***espacio indoor*** es un ambiente que contiene diferentes entidades tales como habitaciones, puertas y pasillos que permiten o restringen el movimiento. En consecuencia, la incapacidad o limitación de movimientos debe ser considerada basándose en la estructura de los espacios indoor. Por lo tanto, las estructuras de datos utilizadas para representar los ***espacios outdoor*** no se pueden aplicar a los ***espacios indoor***, por lo menos sin una mejora razonable.

- Otra diferencia entre ambos espacios se determina en base a las tecnologías que permiten el posicionamiento dentro de los mismos. El posicionamiento de objetos/personas se verá en mayor detalle en la siguiente sección.

  En el ***espacio outdoor***, los sistemas globales de navegación por satélite como el GPS son capaces de informar de forma continua la velocidad y la localización de un objeto en movimiento con diferentes precisiones. Lamentablemente, el GPS no puede ser utilizado en los ***espacios indoor***, ya que la estructura de éstos, basada en celdas, no permite obtener la señal satelital adecuada para que el GPS funcione correctamente. Es decir, la señal satelital se debilita al querer atravesar las construcciones que un ***espacio indoor*** posee; techo, paredes, etc.

  Por lo general, en los ***espacios indoor*** se utilizan dispositivos tales como WiFi, Bluetooth o lector de RFID, que están basados en la detección o en la activación, por lo tanto deberían estar situados en posiciones fijas dentro de los espacios indoor.

  - Ejemplos de ***espacios indoor***: facultades, hospitales, edificios, casas.

  - Ejemplos de ***espacios outdoor***: bosques, parques, cuevas.

2.4.1 Clasificación de los Espacios

Una de las clasificaciones que mejor describe los espacios se basa en la clasificación de los *"espacios Zubin"*, definidos por el lingüista David A. Zubin [11] como un modelo de objetos y conceptos espaciales en función de cómo las personas interactúan con el mundo exterior.

Los espacios Zubin se designan por letras de la A a la D, donde los tipos A y B describen categorías de objetos en función de sus tamaños respecto al cuerpo humano, y los tipos C y D se refieren a los espacios percibidos a través de la visión (ya sea real o imaginaria):

- **Tipo A**: este tipo incluye objetos pequeños y manipulables, como tazas, libros, plantas y animales.
- **Tipo B**: los objetos de este tipo son normalmente más grandes que el cuerpo humano, y no son manipulables ni movibles, por ejemplo los edificios o los árboles.
- **Tipo C**: son espacios que incluyen escenas que se perciben con la vista, tales como campos, cuevas, interiores o salas de los edificios.
- **Tipo D** (transperceptual): son espacios que no pueden ser percibidos como unidades y deben ser visualizados a partir de componentes tales como espacios geográficos, por ejemplo, el interior de un edificio de varias habitaciones.

Se puede notar que una representación puede pertenecer a más de un tipo, por ejemplo:

- Un mapa puede considerarse como una representación de tipo A, ya que se puede ver como un objeto manipulable y movible, o un espacio de tipo D, ya que representa un conjunto de espacios geográficos.
- Una habitación puede ser tanto de tipo B (cuando es considerada como un objeto), o de tipo C (cuando es considerada como un espacio).

Sin embargo, los *"espacios Zubin"* son insuficientes para distinguir entre espacios indoor u outdoor, ya que los espacios u objetos del mismo tipo pueden aparecer tanto en interiores como en exteriores.

Si consideramos el movimiento humano además de la visión, los espacios también se pueden describir en términos de sus propiedades funcionales:

- **Espacio Figurativo:** es más pequeño que el cuerpo humano y puede ser percibido sin movimiento (por ejemplo, tazas y puntos de referencia distantes), incluye un espacio vectorial pequeño (2D) y objetos espaciales (3D).
- **Espacio Vista**: es más grande que el cuerpo humano y puede ser percibido sin movimiento (por ejemplo, pequeñas habitaciones).

- **Espacio Ambiental**: es más grande que el cuerpo humano y no puede ser percibido sin un movimiento considerable, por ejemplo edificios, ciudades, barrios.

- **Espacio Geográfico**: es más grande que el cuerpo humano y no puede ser percibido a través del movimiento, solamente a través de la representación simbólica, tales como mapas.

Una de las partes más importantes del contexto de un objeto, ya sea indoor u outdoor, es su **ubicación espacial.** La misma se clasifica en:

- **Absoluta**: Se conoce la ubicación geométrica exacta del objeto y todos los objetos comparten un sistema de referencia común.

- **Relativa**: Se conoce la ubicación relativa espacial de un objeto para uno o más sensores, pero no se conoce su ubicación absoluta. Cada objeto puede tener su propio sistema de referencia.

- **Simbólica**: su ubicación no tiene una forma geográfica (por ejemplo, si alguien está escribiendo en un teclado particular, solo se sabe el nombre de la máquina o la dirección IP, pero su posición absoluta o relativa no se conoce).

Tenemos que tener en cuenta que estas categorías de ubicaciones no son mutuamente excluyentes. En algunos sistemas, uno puede ser derivado desde otro y su distinción se hace sobre todo para indicar qué información está disponible y cuándo el sistema lo está utilizando.

Una vez más no hay una separación particular de esta taxonomía en categorías indoor y outdoor. Parte de nuestro trabajo es determinar qué claves de la ubicación son más usadas típicamente en espacios indoor contra outdoor. Además de la ubicación física de un objeto hay otras propiedades importantes:

- **Ubicación temporal**: podría ser descrita como absoluta, (fecha y hora universal), relativa (marca de tiempo local) o simbólica (por ejemplo, en algún momento, algo estaba aquí).

- **Grado de movilidad**: Por ejemplo, los muebles que rara vez se mueven, o los humanos que son altamente movibles.

- **Grado de anonimato**: por ejemplo, una identidad única puede o no puede ser conocida.

- **Estado funcional**: ¿cuál es la actividad actual o rol de un objeto? (por ejemplo, persona caminando, asistente de conferencia, equipo en tránsito o fijo en un lugar); y cualquier comportamiento relevante asociado (propiedades dinámicas especificas).

2.5 Posicionamiento

Las personas pasan la mayor parte de su tiempo en ambientes indoor, por este motivo, estos espacios son cada vez más grandes y más complejos, logrando complicar la capacidad de las personas de recorrerlos sin perderse.

En consecuencia, se comenzó a estudiar alguna manera de obtener una fácil ubicación y traslado dentro de estos espacios. Para lograr estos objetivos, se deben tener en cuenta dos grandes factores:

1. Lograr almacenar (en una base de datos, por ejemplo) toda la información relevante del espacio que se quiere estudiar; es decir, guardar la información de los ambientes que lo componen, los accesos que permiten recorrerlo, las estructuras, etc.

2. Obtener información del posicionamiento de las personas dentro del espacio que se quiera analizar. Necesitamos conocer esta información, para poder saber en qué lugar (lo más exacto posible) del espacio, está ubicada la persona que realiza una consulta. Esta consulta puede ser, entre otras:

    i. ¿Cómo llegar del ambiente donde estoy a otro ambiente?

    ii. ¿Cómo llegar a la salida?

    iii. Conocer las salidas de emergencia

Para poder responder estas consultas, necesitamos utilizar la información que se almacenada en el punto 1.

Hoy en día, es conocido por todos el sistema GPS. Este sistema es gestionado por el Departamento de Defensa de los Estados Unidos, y está formado por una red de 24 satélites que permiten mediante triangulación de señales, posicionar en cualquier lugar del globo un dispositivo receptor.

Como se presentó en la sección 2.4, el posicionamiento en espacios outdoor está mucho más desarrollado que en espacios indoor. En el espacio outdoor, los sistemas globales de navegación por satélite como el GPS son capaces de informar de forma continua la velocidad y la localización de un objeto en movimiento con diferentes precisiones. Además de este sistema, existen otros sistemas de posicionamiento global no tan conocidos como son el *Galileo* europeo, el *GLONASS* ruso y el *Beidou* chino. Los sistemas *Galileo* y *Beidou* aún están en fase de despliegue [12].

Todos estos sistemas requieren que nuestro dispositivo reciba al menos señal de tres satélites diferentes (cada señal lleva codificado entre otros datos el código del satélite y la hora de transmisión) para que combinándolas, se pueda establecer la zona posible de situación.

Lamentablemente el GPS no puede ser utilizado en los espacios indoor, ya que la estructura de éstos, basada en celdas, no permite obtener la señal satelital

adecuada para que el GPS funcione correctamente. En espacios indoor, las paredes de los edificios atenúan y dispersan la señal del GPS, lo que aumenta el error de medición hasta el punto de ser inservible para ubicar al usuario. Es decir, la señal satelital se debilita al querer atravesar las construcciones que un espacio indoor posee; techo, paredes, etc.

2.5.1 Posicionamiento Indoor

A continuación, se presentan las tecnologías disponibles para desarrollar sistemas de posicionamiento indoor [3]:

- **Marcadores fijos**
  Los primeros sistemas que aparecieron para la localización indoor fueron los marcadores fijos en distintas partes del edificio. Este sistema consiste en distribuir en ciertos puntos marcadores reconocibles por un dispositivo específico. Un ejemplo de marcador sería un código QR, reconocible por la cámara de un Smartphone. La principal ventaja de estos sistemas es que son sencillos y baratos de implantar. Sin embargo, tienen varios inconvenientes: no son sistemas de localización propiamente dichos, ya que no permiten localizar el usuario de forma dinámica; y precisan de la acción del usuario para localizarle ya que es el usuario es quién decide cuándo leer el código.

- **Sistemas inalámbricos**
  Los sistemas inalámbricos usan ondas electromagnéticas para obtener la localización del usuario. Se envían señales entre sensores estáticos (emisores) y el objeto a ser localizado (receptor). La posición del receptor se determina con respecto a los emisores, ya que la posición de los emisores es conocida de antemano. Por lo tanto, son necesarias dos herramientas diferentes para usar estos sistemas:
    1) un receptor de señal inalámbrica ubicado en el objeto en movimiento y
    2) sensores estáticos instalados en diferentes partes del edificio. Dependiendo de la frecuencia de las ondas electromagnéticas,

  Estas tecnologías se clasifican en: infrarrojos, radio frecuencia (RFID, Wifi, Bluetooth, UWB), y ultrasonidos.

  La ventaja de los sistemas inalámbricos es que son relativamente fáciles de implementar. Por el contrario, el inconveniente es que precisan de una infraestructura de sensores montada en el edificio y de receptores con hardware específico.

  Dado que la mayoría de los celulares actuales disponen WiFi y Bluetooth, éstas son las tecnologías adecuadas. Para el caso de la tecnología WiFi, la infraestructura requerida ya está presente hoy en día en muchos edificios, así que es mucho más práctica y barata de implementar que el Bluetooth. Aun así, presenta dos problemas:

    1) no se puede contar con que esté siempre presente;

2) aunque lo esté, no siempre cubre todas las zonas, ya que deja zonas oscuras donde las ondas de radio no llegan.

Esto limita las capacidades de posicionamiento de la tecnología WiFi y la condiciona a la existencia de una infraestructura previa.

Actualmente, hasta donde sabemos, sólo encontramos un proyecto, llamado Google Maps Floor Plans 2, que está empezando a implementar un sistema que usa tecnologías inalámbricas WiFi para localizar al usuario en interiores de edificios a través del celular. Este sistema todavía está en fase experimental y sólo se puede usar en algunos países.

- **Sistema de navegación inercial**
  Los sistemas de navegación inercial (INS) surgen para evitar la dependencia de la infraestructura y de los sensores específicos que tienen los sistemas de posicionamiento con tecnologías inalámbricas. Los INS usan sensores inerciales como acelerómetros, giroscopios y brújulas para determinar la distancia recorrida y la orientación de movimiento del objeto y así obtener el movimiento del usuario. La gran ventaja de los INS es que no precisan de referencias externas para conocer el movimiento del usuario, así son totalmente independientes del entorno. Esto permite que estos sistemas puedan posicionar al usuario sólo con el uso de su celular sin necesidad de ninguna infraestructura externa, lo que ofrece la posibilidad de usar este sistema en cualquier entorno indoor.

  Hoy en día todavía no existen aplicaciones GIS que utilicen INS para posicionamiento indoor. La inexistencia de INS implementados en celulares es debida, principalmente, a los errores de medida que dan estos sistemas. Un sistema de posicionamiento indoor debe ofrecer la mayor precisión posible cuando ofrezca la localización del usuario. Se tiene que tener en cuenta que en la localización indoor requiere mucha más precisión que en los espacios outdoor. Los errores no deben exceder un metro de error para permitir una diferenciación entre los niveles, los ambientes o los elementos de interés. De hecho, los INS todavía están en fase de investigación y les falta dar el salto a la implementación en aplicaciones [13].

2.6 Resumen

Luego de lo expuesto y lo estudiado en este capítulo, podemos definir los Espacios Indoor como cualquier espacio cerrado construido por el hombre, excluyendo los espacios naturales, que son lo que comúnmente denominamos Espacios Outdoor.

La mayor parte de las personas pasan gran parte de su tiempo en ambientes indoor, por esta razón, estos espacios son cada vez más grandes y más complejos por lo que están tomando mayor importancia en el mundo del posicionamiento geográfico. Como se mencionó a lo largo de este capítulo, actualmente no existe un sistema que permita a las personas posicionarlas dentro de un espacio indoor, y en consecuencia, no existe la manera de permitir el recorrido del mismo o encontrar un camino desde

un ambiente a otro, entre otras cosas. Por lo tanto, es necesario obtener alguna forma de ayuda para las personas que concurren a estos espacios. Esta ayuda consiste en mostrarles la forma correcta de recorrer el espacio, entre otras cosas.

En los espacios outdoor, el posicionamiento se considera una tecnología relativamente madura y altamente desarrollada. Los sistemas globales de navegación por satélite como el GPS son capaces de informar de forma continua la velocidad y la localización de un objeto en movimiento con diferentes precisiones. En espacios indoor, las paredes de los edificios atenúan y dispersan la señal del GPS lo que aumenta el error de medición hasta el punto de ser inservible para ubicar al usuario. Por lo tanto, estos sistemas no son factibles en el posicionamiento indoor.

Existen algunas tecnologías disponibles para desarrollar sistemas de posicionamiento indoor, como marcadores fijos, sistemas inalámbricos, sistemas de navegación inercial, todos con sus ventajas y desventajas. Actualmente, las investigaciones indican que la mejor opción, cuando hablamos de posicionamiento indoor es elegir los sistemas inalámbricos.

Capítulo 3

Técnicas para la definición de lenguajes

Capítulo 3 - Técnicas para la definición de lenguajes

3.1 Introducción

En este capítulo se presentan algunos de los conceptos principales relacionados con el desarrollo de esta tesina: el propósito y las características del metamodelado, junto con la definición de metamodelo. Además se exponen las diferentes metodologías existentes para definir lenguajes.

Se describen los conceptos y propósitos principales de los diferentes mecanismos para la definición de lenguajes, en particular se presenta una investigación más exhaustiva acerca de los lenguajes gráficos que son aquellos con los cuales se definen los metamodelos, éstos últimos son la base de nuestro estudio.

Finalmente, se procede a realizar una comparación entre los mecanismos mencionados anteriormente, analizando ventajas de cada uno de ellos y los contextos en los cuales es más beneficioso utilizar cada técnica.

3.2 Metamodelo

3.2.1 Definición

Existen varias terminologías utilizadas para definir el concepto de metamodelo lo cual puede generar ciertas confusiones para un óptimo entendimiento del mismo. A continuación se hará hincapié en los conceptos y características más relevantes que se obtuvieron a lo largo de nuestro estudio para poder brindar una definición concreta y de fácil entendimiento.

En principio podemos definir metamodelo de la manera que se utiliza usualmente, como un *modelo de modelos* que especifica un lenguaje, es decir, define los procesos y lenguajes entre los que se forma un modelo.

Desde el punto de vista del estándar MOF, se define como el *modelo de un lenguaje de modelado*. El metamodelo define la estructura, semántica y restricciones para una familia de modelos.

Los metamodelos son modelos que especifican los modelos, y un modelo es una simplificación de la realidad obtenido a partir de la aplicación de una serie de abstracciones de la misma, por medio de la cual podemos organizar y entender su estructura, datos y dinámica.

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

Para lograr obtener una mejor y más concreta definición acerca del concepto de metamodelo, podemos destacar ciertos autores que lo definen de las siguientes maneras [14]:

- Según Mellor, Scott, Uhl & Weise en su libro “Principles of Model-Driven Architecture”, se define al metamodelo como un modelo de un lenguaje de modelado. El metamodelo define la *estructura*, *semántica* y *limitaciones* para una familia de modelos.

- Según Clark, Sammunt & Willians en su libro “A foundation for Language Driven Development”, se define al metamodelo como un modelo de un lenguaje que capta sus propiedades y características esenciales. Esto incluye los *conceptos del lenguaje* que soporta, su *sintaxis* textual y/o gráfica y su *semántica*.

- Según Quasar Tecnología [15], el metamodelo es un *modelo con la capacidad de almacenar diferentes modelos*. Partiendo de esta definición se puede pensar en la posibilidad de construir un producto de software con la capacidad para registrar información en el metamodelo y otro producto con la capacidad de realizar consultas sobre el mismo. Por lo tanto el metamodelo está dividido en dos áreas:

    - **Área de Metadatos:** Esta área almacena el metamodelo en sí y, en un conjunto de tablas relacionales, se guarda una descripción detallada del modelo de los datos a ser almacenados en el área de datos.

    - **Área de Datos**: Esta área almacena la información acorde a la estructura definida en el área de metadatos. Es aquí donde se registra la información proveniente del mundo real, estructurada de acuerdo a los formatos que forman parte del modelo almacenado en el área de metadatos.

En conclusión, podemos decir que un metamodelo *define la sintaxis abstracta de un lenguaje* que establece los conceptos y relaciones entre ellos, e incluye las reglas que determinan qué es un modelo bien formado.

El metamodelo restringe el conjunto de modelos posibles a un conjunto de modelos válidos. Es decir que, por ejemplo, no cualquier combinación de letras que se escribe es una sentencia es válida en español; son las restricciones que especifican cuáles lo son y cuáles no. Por lo tanto, un modelo de sintaxis concreta restringe el número de modelos válidos.

En otras palabras, el metamodelo de un dominio específico realiza una abstracción de los elementos estáticos y dinámicos de los modelos que se pueden instanciar en la realidad, para de esta forma permitir crear nuevos modelos que retomen las características genéricas y las personalicen al ámbito según los requerimientos específicos.

El metamodelo no sólo es utilizado como un proceso de comportamiento en el tiempo de los modelos, sino también como un proceso de evaluación del metamodelo con respecto a los casos que no puede resolver y así redefinirlo si es necesario. Es decir, el metamodelo debe ser a su vez redefinido constantemente para evitar su estancamiento y logre seguir siendo válido a través del tiempo.

Un metamodelo se expresa con notación gráfica y con un lenguaje formal, como OCL (ver capítulo 4, sección 4.4.3), para aumentar su precisión y eliminar ambigüedades.

El metamodelo consta de:

- **Dominios:** Contiene los dominios en los que se divide el modelo.
- **Entidades:** Las entidades del modelo clasificadas por dominios.
- **Relaciones:** Las relaciones que existen entre las entidades. Se caracterizan por el nombre de la relación, y la cardinalidad de la misma.
- **Atributos:** Los atributos de las entidades y de las relaciones que van a formar parte del modelo. Un atributo en el metamodelo puede ser compartido por una o más de las entidades que forman el modelo. Así, por ejemplo, el atributo nombre es probable que forme parte de muchas de las entidades del modelo y, por tanto puede estar asociado a más de una entidad. Tiene una relación directa con el modelo físico de los datos que se utilice.

A partir de las definiciones analizadas anteriormente podemos definir al mecanismo de modelado, el *metamodelado*, como el análisis, la construcción y el desarrollo de los marcos, reglas, restricciones, modelos y teorías aplicables y útiles para el modelado de una clase predefinida de problemas.

El modelado es un ejercicio de abstracción, entendido como una simplificación y generalización de la realidad. El uso de la abstracción permite describir, representar, manejar y resolver problemas complejos, pudiendo después aplicar los resultados a casos concretos. Normalmente, se construyen jerarquías de abstracción, en las que cada nivel de abstracción se apoya en los inferiores.

3.2.2 Características del metamodelado

En la sección anterior se mencionaron conceptos tales como *modelo* e *instancia* que son importantes de entender al momento de realizar metamodelado, por lo que en este inciso se procederá a explicarlos más detalladamente.

Desde el punto de vista de la ingeniería de software, y en torno al contexto que se estudia en esta tesina, se define ***modelo*** como una instancia del metamodelo. Cuando se crea un modelo se está definiendo un lenguaje para describir el área que se está analizando o el sistema que se está diseñando. Es una abstracción de los fenómenos en el mundo real, que se ajusta a su metamodelo en la forma en que un programa de ordenador se ajusta a la gramática del lenguaje de programación en el que está escrito. Vale la pena aclarar que un modelo siempre se ajusta a un metamodelo único [16].

El ***modelo*** de un sistema es una representación conceptual obtenida a partir de la identificación, clasificación y abstracción de los elementos que constituyen el problema y su posterior organización en una estructura formal. Simplifica la realidad,

suprimiendo detalles irrelevantes y reteniendo los aspectos esenciales, de modo que la esencia del sistema sea mejor conocida y podamos hacer frente a su complejidad.

De esta forma, el modelo de un sistema actúa como una especificación de los requerimientos que el sistema debe satisfacer, proveyendo un medio de comunicación y negociación entre usuarios, clientes, analistas y desarrolladores, así como también un documento de referencia durante la verificación y validación, y durante la evolución del producto. El modelo del sistema se expresa utilizando un lenguaje de modelado. Los metamodelos y los modelos tienen una relación clase-instancia; cada modelo es una instancia de un metamodelo.

Un metamodelo se expresa con notación gráfica (como los diagramas de clases UML) y con un lenguaje formal, como OCL, para aumentar su precisión y eliminar ambigüedades.

Suelen señalarse 5 (cinco) características que un modelo debe cumplir para que sea útil. A continuación se describen:

- Debe ser **abstracto**, es decir, debe ser una versión reducida del sistema que representa.
- Debe ser **comprensible**, es decir, debe expresarse de tal forma que se pueda entender fácilmente
- Debe ser **preciso**, es decir que debe representar fielmente el sistema modelado
- Debe ser **predictivo**, lo que significa que se puede utilizar para obtener conclusiones correctas sobre el sistema
- Debe ser **barato**

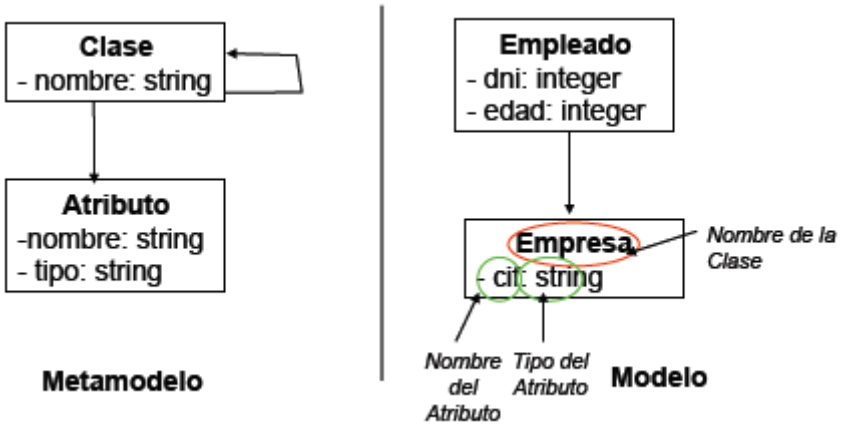


Figura 3.1 - *Relación clase-instancia /metamodelo-modelo*

Como hemos explicado anteriormente, un **metamodelo** es una instancia de un meta-metamodelo. La principal responsabilidad de la capa del metamodelo es definir un lenguaje para especificar modelos.

Por otra parte, el **modelo** es una instancia de un metamodelo. La principal responsabilidad de la capa del modelo es definir un lenguaje que describe la información del dominio.

Finalmente queda explicar el concepto de **instancia**, que representa una instancia (valga la redundancia) de un modelo. Su primera responsabilidad es describir la información del dominio. Una instancia es válida sólo cuando se cumplen todas las restricciones del modelo [17,18].

Etapa	Descripción	Ejemplo
Metamodelo	Una instancia de un meta-metamodelo. Define el lenguaje para especificar un modelo.	Clase, Atributo, Operación, Componente
Modelo	Una instancia de un metamodelo. Define un lenguaje para describir la información del dominio.	Persona, edad
Instancia	Una instancia de un modelo. Define una información específica del dominio.	Juan, 28

3.2.3 Ejemplos

Los usos más comunes para metamodelos son:

- Como un esquema para los datos semánticos que necesita ser intercambiado o almacenado.
- Como un lenguaje que soporte un método o proceso en particular.
- Como un lenguaje para expresar semántica adicional de la información existente.
- Como mecanismo para crear herramientas que funcionan con una amplia clase de modelos en tiempo de ejecución.

La Figura 3.2 muestra el metamodelo simplificado del lenguaje UML, donde se especifica que un modelo UML está formado por paquetes (*Package*), donde cada paquete está integrado por clases (*Clase*) que poseen atributos (*Attribute*). Los atributos tienen un nombre (*name*) y un tipo (*type*), que puede ser un tipo de dato primitivo (*PrimitiveDataType*) o una clase.

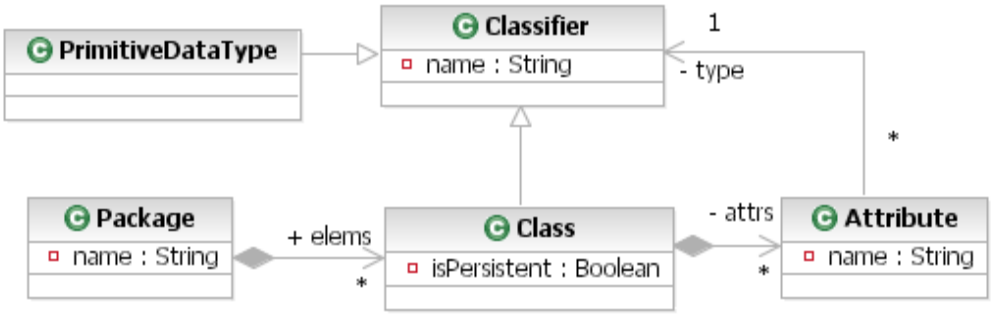


Figura 3.2 - *Metamodelo simplificado de UML*

3.3 Lenguajes textuales

3.3.1 Definición

El lenguaje textual representa la comunicación a través de letras, por medio de distintos sistemas de escritura. El texto es una unidad de lenguaje en uso. El lenguaje textual ha dado lugar a la comunicación escrita. En la comunicación escrita encontramos dos elementos importantes:

- Lo **sintáctico**: Hace referencia a la forma o estructura de las expresiones, sentencias y programas.

- Lo **semántico**: Hace referencia al significado de las expresiones, sentencias y programas.

Ambos conceptos proveen la definición de un lenguaje.

La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan secuencias de texto que incluyen palabras, números y puntuación, de manera similar a los lenguajes naturales escritos.

3.3.2 Backus Naur Form: BNF

En esta sección se presenta brevemente uno de los lenguajes textuales más conocidos para describir la sintaxis de un lenguaje de programación.

Backus Naur Form (BNF) es una meta sintaxis usada para expresar gramáticas libres de contexto, es decir, una manera formal de describir cuáles son las palabras básicas del lenguaje y cuáles secuencias de palabras forman una expresión correcta dentro del lenguaje de programación.

Como toda gramática, una gramática de este tipo contiene cuatro partes fundamentales:

- Un conjunto finito de símbolos conocido como ***terminales***. Son los símbolos pertenecientes al lenguaje que genera la gramática, estos pueden ser palabras de lenguaje, símbolos, etc. Cualquier unidad léxica que sea parte del lenguaje.

- Un conjunto finito de símbolos intermedios conocidos como ***no terminales*** que se utilizan en el proceso de generación de la gramática. A estos también se les suele conocer como variables o símbolos auxiliares. La nomenclatura típica para diferenciar los no terminales de los terminales es encerrar los primeros entre brackets, por ejemplo <*T*> es un ***no terminal*** y *T* es un ***terminal***.

- Un ***no terminal especial*** único con el que se comienza el proceso de generación. Debe de ser un símbolo perteneciente al conjunto de ***no terminales***. No tiene un nombre estándar pero suele llamarse <*S*>.

- Un conjunto finito de reglas conocido como ***producciones*** de la forma:

  <A> ::= secuencia

  donde:

  - El símbolo **::=** se lee como "está definido por" o "genera".
  - El lado izquierdo de la producción, **<A>**, debe ser un ***no terminal***.
  - El lado derecho, **secuencia**, debe ser una forma sentencial, es decir una secuencia finita de símbolos ***terminales*** y ***no terminales*** que mantienen un orden particular. Este lado derecho puede ser una secuencia vacía en cuyo caso se utiliza el símbolo ***ε*** para referenciarlo.

Para las formas sentenciales se suele utilizar letras griegas como nomenclatura estándar. Una vez que se tiene claramente definidas las cuatro partes mencionadas anteriormente, este define un lenguaje.

Este lenguaje está compuesto por un conjunto de textos. Para poder saber si un texto particular pertenece o no a este conjunto hay que seguir el proceso de generación que consiste en lo siguiente:

- Se tiene como ***forma sentencial*** inicial al ***no terminal*** inicial de la gramática. Con el objetivo de tratar de llegar a producir el texto del que se desea probar su pertenencia al lenguaje, se debe seleccionar algún ***no terminal*** de la forma sentencial actual y cambiarlo por alguna de las ***producciones*** de la gramática que lo definen a él. Este proceso se debe seguir iterando hasta que se genere el texto deseado o se compruebe la imposibilidad de hacerlo.

Al escribir la gramática de un lenguaje en BNF se puede llegar a notar que es un poco largo y tedioso el describir algunas de las reglas. Para solventar alguno de esos problemas (como puede ser la recursividad), se utiliza la notación EBNF que permite realizar descripciones más fáciles de los lenguajes.

EBNF es una meta sintaxis que deriva de BNF, e incluye ciertos operadores que facilitan la descripción de las ***producciones***. Ellos son:

- ?: Expresa que el símbolo (o grupo de símbolos entre paréntesis) del lado izquierdo del operador puede ser ***opcional***. Puede aparecer 0 (cero) ó 1 (una) vez.
- *: Expresa 0 (cero) o más repeticiones del elemento anterior
- +: Expresa 1 (una) o más repeticiones del elemento anterior.

EBNF no es más potente que BNF en términos de qué lenguajes puede definir, sino que es más flexible. Cualquier producción escrita en EBNF, puede ser traducida a una producción equivalente en la meta sintaxis BNF.

3.3.3 Ejemplo

El siguiente es un ejemplo de una gramática escrita en BNF, que define números enteros, decimales y negativos.

```
S ::= '-' numero | numero

numero ::= entero | entero '.' entero

entero ::= digito | digito entero

digito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

A continuación se presenta la gramática definida anteriormente, escrita en lenguaje EBNF. Puede observarse que ambas definen lo mismo y la gramática escrita en EBNF es más legible.

```
S ::= ('-')? digito+ ('.' digito+)?

digito := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

3.4 Lenguajes gráficos

3.4.1 Definición

Los lenguajes de modelado gráfico se refieren a un conjunto de símbolos gráficos y reglas, utilizados para transmitir la información al usuario a través de diagramas. Cada uno cuenta con una serie de modelos y diagramas para representar un sistema. Un modelo contiene los métodos o procesos asociados a los lenguajes de modelado, que describen un sistema a través de diferentes tipos de diagramas. A su vez, cada modelo se compone de varios diagramas para representar un sistema a partir de sus características particulares [19].

El éxito de los lenguajes gráficos de modelado, como el UML, se basa principalmente en el uso de construcciones gráficas que transmiten un significado intuitivo. Estos lenguajes son atractivos para los usuarios porque son claros y

entendibles; sin embargo, es fundamental contar con un lenguaje que permita expresar restricciones semánticas adicionales sobre los objetos del modelo, pudiendo obtener modelos más precisos y verificables.

OCL es un lenguaje de especificación formal fácil de leer y escribir, que fue definido por la OMG. Permite expresar restricciones semánticas del sistema que no se pueden expresar a partir de una notación gráfica (por ejemplo un diagrama de clases UML). De esta forma, los diagramas complementados con expresiones OCL son más precisos, su documentación es más clara y la compresibilidad del sistema en etapas iniciales del desarrollo de software es mayor.

Otra característica que es muy importante para la utilización de los lenguajes de modelado gráfico, es contar con técnicas de refinamientos permitiendo un desarrollo por etapas con distintos niveles de abstracción y postergando los detalles del problema en etapas posteriores.

3.4.2 Diferencias con lenguajes textuales

Los lenguajes de modelado, tanto los gráficos como los textuales, permiten la representación del comportamiento y la estructura de los sistemas del mundo real. Los lenguajes de modelado gráfico utilizan modelos visuales y diagramas que realizan esa representación de manera precisa, entendible y económica, lo que facilita su uso en las herramientas CASE convencionales.

Las principales diferencias entre los lenguajes basados en texto y los lenguajes basados en gráficos son las siguientes:

- **Contenedor vs. Referencia:** En los lenguajes textuales una expresión puede formar parte de otra expresión, que a su vez puede estar contenida en otra expresión mayor. Esta relación de contenedor da origen a un árbol de expresiones. En el caso de los lenguajes gráficos, en lugar de un árbol se origina un grafo de expresiones ya que una sub-expresión puede ser referenciada desde dos o más expresiones diferentes.

- **Sintaxis concreta vs. Sintaxis abstracta:** En los lenguajes textuales la sintaxis concreta coincide (casi) exactamente con la sintaxis abstracta, mientras que en los lenguajes gráficos se presenta una marcada diferencia entre ambas.

- **Ausencia de una jerarquía clara en la estructura del lenguaje**. Los lenguajes textuales en general se aprenden leyéndolos de arriba hacia abajo y de izquierda a derecha, en cambio los lenguajes gráficos suelen asimilarse de manera diferente dependiendo de su sintaxis concreta (por ejemplo, comenzamos prestando atención al diagrama más grande y/o colorido). Esto influye en la jerarquía de la sintaxis abstracta, ocasionando que no siempre exista un orden entre las categorías sintácticas.

3.4.3 La arquitectura de 4 capas de modelado del OMG

El OMG usa una arquitectura de cuatro niveles o capas de modelado para sus estándares. En la terminología del OMG estas capas se llaman **M0**, **M1**, **M2** y **M3**. A continuación se desarrollaran más detenidamente, explicando las características principales de cada una de ellas.

- **Nivel M3: Meta-metamodelo**

  La capa del meta-metamodelo es la capa fundamental de la arquitectura para el metamodelo. La principal responsabilidad de esta etapa es definir el lenguaje para especificar un metamodelo. Un meta-metamodelo define un modelo en un alto nivel de abstracción mayor que un metamodelo; es decir es más compacto que un metamodelo, y a menudo define varios metamodelos

  Es un modelo que sirve para definir los conceptos del metamodelo y se especifica usando sus propios elementos.

  Dentro del OMG, MOF es el lenguaje estándar de la capa M3. No existe otro meta-nivel por encima de MOF. Básicamente, el MOF se define a sí mismo.

- **Nivel M2: Metamodelo**

  Un metamodelo es una instancia de un meta-metamodelo y significa que cada elemento del metamodelo es una instancia de un elemento del meta-metamodelo; es decir, es un modelo que sirve para definir los conceptos del modelo y se especifica en base a los elementos definidos en el meta-metamodelo.

  La responsabilidad primaria de la capa del metamodelo es definir un lenguaje para especificar modelos. UML y OCL son ejemplos de metamodelos. En general estos son más detallados que los meta-metamodelos que los describen, sobre todo cuando ellos definen semántica dinámica. Para dar un ejemplo, en el metamodelo de UML se definen los objetos del lenguaje unificado como Class, Property, Operation, etc.

- **Nivel M1: Modelo**

  Un modelo es una instancia de un metamodelo. Cuando creamos un modelo estamos definiendo un lenguaje para describir el área que estamos analizando o el sistema que estamos diseñando. Por la tanto, la responsabilidad de esta capa es definir un lenguaje que describa los dominios semánticos, para permitirles a los usuarios modelar una variedad de dominios diferentes, como procesos, requerimientos, etc.

  El modelo del usuario contiene los elementos del modelo y los snapshots de instancias de dichos elementos que sirven de moldes para los datos que vamos a introducir, manipular, almacenar y procesar en nuestras aplicaciones.

- **Nivel M0: Instancias**

  Esta capa representa las instancias de los elementos del modelo (instancias "reales" del sistema) en tiempo de ejecución. Estos describen el área o dominio a los que está dedicada la aplicación. Los snapshots que son modelados en M1 son versiones reducidas de las instancias en tiempo de ejecución.

  En este nivel es en donde se especifican datos reales de una aplicación. Los datos particulares son especificados por el desarrollador acorde a la aplicación que se está representado. Aquí no se habla de clases, ni atributos, sino de entidades físicas que existen en el sistema.

La figura a continuación muestra un ejemplo completo donde se aprecia la relación existente entre las capas de modelado definidas por el OMG.

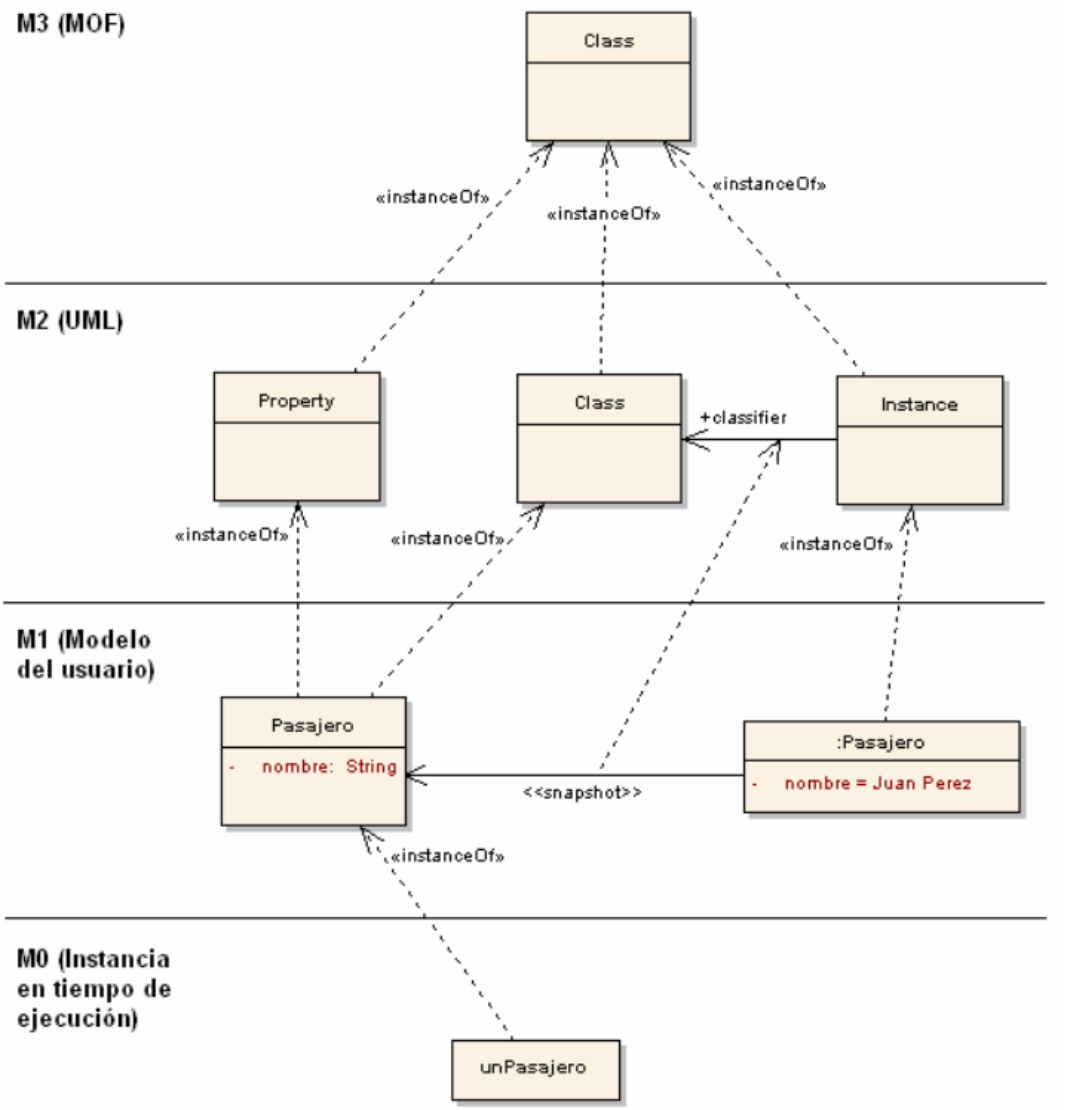


Figura 3.3 - *Ejemplo de modelo de 4 capas*

La Figura 3.3 muestra las cuatro capas de la arquitectura de modelado, indicando las relaciones entre los elementos en las diferentes capas. Puede verse que en la capa M3 se encuentra el meta-metamodelo MOF, a partir del cual se pueden definir distintos metamodelos en el nivel M2, como UML, OCL, etc. Instancias de estos metamodelos serán los elementos del nivel M1, como modelos UML, o modelos Java. A su vez, instancias de los elementos M1 serán los objetos que cobran vida en las corridas del sistema.

3.4.4 El lenguaje de modelado más abstracto: MOF

Para el grupo del OMG el problema no consistía únicamente en solicitar una serie de normas para crear diagramas, sino que estos diagramas y sus componentes deberían ser compatibles con los servicios necesarios para transportarlos de una herramienta de diagramación a otra (lo cual podría significar también el transportar diagramas de una plataforma a otra).

El lenguaje de modelado debería ser coherente con el servicio necesario para proporcionar y transportar los componentes de los modelos. Este servicio es conocido como Meta-Object Facility (MOF).

MOF es un meta-metamodelo que se utiliza para especificar metamodelos orientados a objetos. Define los elementos comunes y estructuras de metamodelos que son utilizados para construir modelos de sistemas orientados a objetos. Es independiente de la plataforma.

La especificación de MOF proporciona:

- Una definición formal del meta-metamodelo MOF; es decir un lenguaje abstracto para especificar metamodelos MOF.
- Un conjunto de reglas para el mapeo de metamodelos MOF a interfaces independientes del lenguaje de programación, definidos por medio del estándar de CORBA IDL. Una implementación de estas interfaces para un modelo concreto podría ser utilizada para acceder y modificar cualquier modelo basado en ese metamodelo.
- Un conjunto de interfaces reflexivas para manipular metadatos independientes del metamodelo.
- Un formato XMI para el intercambio de metamodelos MOF.
- Soporte para los modelos arbitrarios y paradigmas de modelado.
- Posibilidad de relacionar diferentes tipos de metadatos.
- Posibilidad de adicionar nuevos metamodelos y metadatos.

MOF es útil para la definición del metamodelo de UML, es decir, ayuda en la correcta construcción de los diagramas para modelar los requisitos, pero no contribuye en la selección adecuada del lenguaje de modelado. Los metamodelos instanciados

de MOF se emplean para definir cuáles deben ser los componentes utilizados en cada diagrama de UML y cómo debe ser la forma de relacionarlos.

Entre otros, el metamodelo de OCL está especificado mediante dicho meta-metamodelo.

Como se mencionó en la sección anterior, MOF se encuentra en la capa superior de la arquitectura de 4 capas. Se basa en el paradigma de *Orientación a Objetos*; por este motivo usa los mismos conceptos y la misma sintaxis concreta que los diagramas de clases de UML.

Actualmente, la definición de MOF está separada en dos partes fundamentales, EMOF (Essential MOF) y CMOF (Complete MOF), y se espera que en el futuro se agregue SMOF (Semantic MOF). La Figura 3.4 muestra la relación entre EMOF y CMOF. Ambos paquetes importan los elementos de un paquete en común, del cual utilizan los constructores básicos y los extienden con los elementos necesarios para definir metamodelos simples, en el caso de EMOF y metamodelos más sofisticados, en el caso de CMOF.

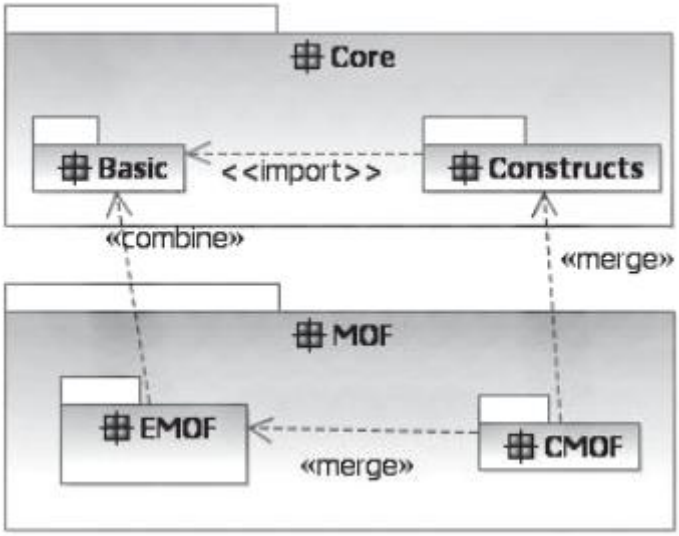


Figura 3.4 - *El lenguaje MOF*

3.4.4.1 Construcciones básicas utilizadas por MOF

MOF es un estándar del OMG que establece un lenguaje común y abstracto para definir lenguajes de modelado, y cómo acceder e intercambiar modelos expresados en dichos lenguajes. MOF usa cinco construcciones básicas para definir un lenguaje de modelado:

- **Clases:** usadas para definir tipos de elementos en un lenguaje de modelado. Por ejemplo, la relación de dependencia de UML es una clase definida en MOF, que representa el tipo de todas las dependencias que pueden crearse en un modelo UML.

- **Generalización:** define herencia entre clases. Por ejemplo UML::Classifier es una generalización de UML::Class. La subclase hereda todas las características de la clase padre.
- **Atributos:** usados para definir propiedades de elementos del modelo. Los atributos tienen un tipo y una multiplicidad.
- **Asociaciones:** definen relaciones entre clases. Una relación tiene dos extremos, cada uno de los cuales puede tener definido un nombre de rol, navegabilidad y multiplicidad
- **Operaciones:** definen operaciones dentro del ámbito de una clase, junto con una lista de parámetros.

3.5 Resumen

En este capítulo se han desarrollado conceptos importantes que facilitarán la comprensión de los capítulos posteriores.

Luego del análisis realizado podemos concluir que un metamodelo es una definición precisa de los constructores y reglas necesarias para definir la semántica de los modelos. Además, el metamodelado puede verse como una actividad que sirve para organizar los modelos en diferentes niveles, de tal modo que un modelo se describe por otro modelo que está situado en un nivel superior, su metamodelo. El metamodelo se suele especificar usando notaciones visuales como diagramas de clases o entidad-relación, más restricciones adicionales expresadas en un lenguaje de restricciones que suele ser textual como OCL.

A la hora de definir metamodelos han surgido diferentes propuestas, entre ellas hemos estudiado particularmente el meta-metamodelo MOF. Dicho meta-metalenguaje es un lenguaje gráfico para el cual la OMG ha estandarizado una serie de correspondencias que especifican cómo se gestionan los metadatos en una tecnología determinada. XMI es el estándar para definir metamodelos MOF en XML.

Capítulo 4

Representación de Espacios Indoor

Capítulo 4 - Representación de Espacios Indoor

4.1 Introducción

A lo largo de este capítulo se describen detalladamente los pasos que se fueron atravesando hasta obtener el metamodelo abstracto que representa los espacios indoor, ellos son:

- Análisis de los lenguajes utilizados para la definición del metamodelo y sus restricciones.
- Obtención de los conceptos comunes de diferentes tipos de espacios indoor junto con sus características.
- Desarrollo de la construcción del metamodelo abstracto *"Espacio Indoor"* de manera iterativa e incremental. Este mecanismo muestra cómo el metamodelo puede ir mejorándose progresivamente a través de la aplicación de patrones de diseño y otras buenas prácticas de modelado, hasta encontrar el que más se adecue a nuestras necesidades.
- Definición de restricciones OCL sobre el metamodelo final desarrollado.

Luego se plantea un caso de estudio sobre el metamodelo abstracto con el objetivo de demostrar su capacidad para representar cualquier espacio indoor que se requiera. Para esto refinamos el metamodelo creado, generando un metamodelo concreto al que llamamos *"Facultad"*, mediante un nuevo análisis, reutilizando todos los conceptos comunes del metamodelo abstracto y agregando nuevos, particulares del dominio. Para enriquecer y brindar mayor robustez a estos metamodelos, se definieron restricciones OCL sobre cada uno.

4.2 Lenguajes utilizados

4.2.1 UML

UML es un lenguaje gráfico unificado utilizado para visualizar, especificar, construir y documentar los artefactos de sistemas de software. Ofrece una forma estándar para escribir modelos de sistemas [20, 21].

Una de las ventajas que se pretende lograr mediante un lenguaje unificado es permitir el intercambio de diagramas y de formas de representación de sistemas entre diversas herramientas. Por ejemplo, si un grupo de desarrolladores utilizara una herramienta CASE o una herramienta de modelado visual, debería existir la facilidad de transportarla a otra herramienta, independientemente del proveedor o fabricante de cada una de ellas.

Pero el problema de estandarización no tiene que ver sólo con la transportabilidad de los diagramas de una herramienta de software a otra. Para mediados de los noventa las llamadas “metodologías” habían proliferado de tal forma

que los proyectos y equipos de trabajo se topaban constantemente con dificultades para seleccionar un método de análisis y diseño. En otras palabras, cada uno de estos métodos contaba con su propio lenguaje de modelado. Esta falta de estandarización impedía la reutilización de soluciones de un proyecto a otro, y muchas veces inhibía la inversión en capacitación de personal y en herramientas para diagramar.

Para poder llegar a un lenguaje estándar, los proponentes de UML buscaron definir con precisión el significado de los símbolos, así como también las formas de usarlos y relacionarlos. Fue necesario que se alcanzara un amplio acuerdo en torno a cuáles debían ser los artefactos a usar durante el análisis y el diseño; es decir, cuáles debían ser los componentes de los modelos. Esta es precisamente la función del metamodelo.

En el desarrollo de software los modelos juegan el mismo papel que los planos y las especificaciones en la industria de la construcción debido a que:

- Representan la forma de ir plasmando las aproximaciones e ideas necesarias para resolver los requerimientos del cliente
- Los modelos son esenciales para la comunicación entre los distintos equipos de trabajo y entre las distintas disciplinas que participan en un proyecto
- Es mucho más económico hacer correcciones sobre un modelo “en papel” que hacerlas durante la construcción misma.
- Las facilidades que brinda la documentación para dar mantenimiento a una aplicación.

Las causas de la crisis del software son variadas, pero entre las más importantes se encuentra la falta de una cultura común a quienes realizan los desarrollos. Esta carencia está íntimamente relacionada con la falta de medios de comunicación para, por ejemplo, intercambiar técnicas y soluciones exitosas. Es precisamente en estos aspectos donde UML nos ayuda.

Además de obviarnos el tener que decidir entre varias formas de representación, UML nos proporciona un lenguaje visual de modelado, y nos permite aprovechar componentes, plantillas y patrones a partir de soluciones exitosas que ya han sido probadas.

Para mantener su característica de estándar abierto, UML es independiente de los lenguajes, de las bases de datos, de la marca del software y del equipo de cómputo. Deliberadamente, los proponentes de este lenguaje unificado lo han dejado únicamente como un medio para especificar, visualizar y documentar los artefactos del desarrollo e implantación de sistemas. Es importante hacer hincapié en el hecho de que UML no es un método de análisis y diseño, ni pretende representar un proceso para guiar las actividades de un desarrollo. Los usuarios de distintos métodos podrán mantener su forma actual de trabajo, adaptando sus diagramas de acuerdo a los símbolos utilizados en UML. En la mayor parte de los casos esta adaptación se va a poder realizar con facilidad.

Se debe tener siempre presente el objetivo central del desarrollo de sistemas: lo que esperan los clientes que solicitan una solución informática es llegar a contar con un sistema confiable y amigable. El modelado es sólo un medio para alcanzar esta meta, y no un fin en sí mismo. Actualmente UML es sólo un lenguaje de modelado, y no pretende ser un lenguaje para la programación mediante símbolos visuales.

4.2.2 OCL

Un diagrama UML, como un diagrama de clases, no está lo suficientemente refinado para proveer todos los aspectos relevantes de una especificación. Existe, entre otras cosas, una necesidad por describir restricciones adicionales sobre objetos en el modelo. Tales restricciones son a veces descriptas en lenguaje natural, pero esto puede resultar en ambigüedades. Para escribir restricciones no ambiguas han sido desarrollados los lenguajes formales, que presentan la desventaja de ser utilizados por personas con un gran conocimiento matemático, siendo difíciles de usar por los modeladores de sistemas. OCL (*Object Constraint Language*) ha sido desarrollado para llenar este hueco. Es un lenguaje de especificación formal que es fácil de leer y escribir. Al ser un lenguaje de especificación puro se garantiza que toda expresión OCL es libre de efectos laterales; es decir, cuando una expresión OCL es evaluada simplemente retorna un valor, sin modificar nada en el modelo [22].

OCL no es un lenguaje de programación, no es posible escribir lógica de programas o flujo de control. Es un lenguaje que se puede emplear para especificar restricciones y otras expresiones adjuntas a los modelos MOF.

OCL es un lenguaje tipado, lo que significa que cada expresión tiene un tipo. Para ser bien formada, una expresión debe concordar con los tipos de reglas del lenguaje; por ejemplo, no puede compararse un Integer con un String.

4.2.2.1 Restricciones OCL

Las restricciones que podemos especificar con OCL son:

- **Invariantes**

  Es una restricción que se liga a un Classifier (Clase, Interface, etc). El propósito del invariante es definir una condición que debe ser válida siempre para todas las instancias de un Classifier.

  La restricción tiene el estereotipo ***<<invariant>>***. Los invariantes se ligan a un solo Classifier, y este es el tipo de la variable contextual.

- **Definiciones**

  Una definición es un Constraint que se liga a un Classifier. La variable o función definida puede utilizarse como una propiedad o una operación del correspondiente Classifier. El propósito de esta restricción es definir expresiones OCL reusables.

  La restricción tiene el estereotipo ***<<definition>>***. Las definiciones se ligan a un solo Classifier, y este es el tipo de la variable contextual.

- **Precondiciones**

  Una precondición es una restricción que se liga a un Operation de un Classifier. Esta restricción establece una condición que debe cumplirse antes de ejecutar la operación.

  La restricción tiene el estereotipo ***<<precondition>>.*** Las precondiciones se ligan a una sola Operation, y el tipo de la variable contextual es el Classifier que define la operación.

- **Postcondiciones**

  Una postcondición es una restricción que se liga a un Operation de un Classifier. El propósito de esta restricción es definir la condición que debe cumplirse luego de ejecutar la operación.

  Una postcondición consiste en una expresión OCL de tipo Boolean. En el caso de los invariantes las restricciones se deben cumplir en todo momento, en cambio en las precondiciones y postcondiciones deben cumplirse antes y después de ejecutar la operación. En una expresión OCL utilizada como postcondición los elementos se pueden decorar con el postfijo ***"@pre"*** para hacer referencia al valor del elemento al comienzo de la operación. La variable ***result*** se refiere al valor de retorno de la operación.

  La restricción tiene el estereotipo ***<<postcondition>>***. Las postcondiciones se ligan a una sola Operation, y el tipo de la variable contextual es el Classifier que define dicha Operation.

Además de restricciones a nivel modelo se pueden definir reglas a nivel metamodelo. Estas son útiles para definir reglas de buena formación (Well – Formedness Rules) y reglas de diseño [23].

4.3 Espacios Indoor

4.3.1 Análisis de conceptos comunes

El comienzo de nuestra investigación se basó en seleccionar diferentes tipos de espacios indoor que podemos encontrar en la vida cotidiana; ellos fueron: casas particulares, edificios, hospitales y facultades. Con la elección de dichos espacios pudimos realizar un análisis, encontrando características generales y particulares de cada uno. Este estudio nos permitió refinar las características y conceptos que son comunes a todos los espacios estudiados y de esta manera obtuvimos las siguientes características fundamentales que podemos destacar acerca de los mismos:

- Son **Espacios Construidos**, es decir, hechos por el hombre, que se componen generalmente de superficies rectilíneas (por ejemplo, paredes).
- Son **Cerrados** ya que están compuestos por paredes, techo, piso.
- Se pueden ver completos mirando a su alrededor, es decir son espacios captables por el hombre.

- Contienen **objetos de pequeña escala**, que varían en tamaño desde lo suficientemente pequeño como para recoger (por ejemplo, lápices) hasta **objetos más grandes que el cuerpo humano** (por ejemplo, escritorios).

El primer paso que realizamos para definir un metamodelo que permita representar los espacios indoor, consistió en una investigación acerca de cómo están compuestos los espacios mencionados anteriormente, descubrir cuáles son sus conceptos generales y cuáles son propios de cada espacio indoor para finalmente, realizar una correcta clasificación de conceptos.

En base a nuestra investigación podemos nombrar los siguientes conceptos comunes a todos los espacios indoor:

1. **Ambiente:** Luego de analizar cada uno de los espacios seleccionados, pudimos concluir que todos ellos están constituidos por diferentes secciones que comparten ciertas características. En una casa particular podemos encontrar diferentes espacios tales como living, cocina, comedor, baños, entre otros; en un edificio encontramos departamentos, salón de usos múltiples; en un hospital consultorios, quirófanos, baños, oficinas, entre otros; y en una facultad podemos encontrar aulas, oficinas, biblioteca, fotocopiadora, etc. Todos los espacios mencionados anteriormente comparten características comunes tales como ser espacios encerrados por paredes, fáciles de transitar entre ellos a través de aberturas en las paredes. A dichos espacios los denominamos *"Ambientes"*, y los clasificamos de la siguiente manera:

   1.1 **Ambiente Simple:** Representa cualquier espacio al que se pueda acceder a través de uno o más accesos.

      1.1.1 **Pasillo:** Representa un ambiente que contiene más de un acceso.

   1.2 **Ambiente Compuesto:** Son aquellos que están compuestos por dos o más ambientes.

2. **Nivel:** Continuando con el refinamiento de conceptos comunes entre los espacios seleccionados, encontramos que cada uno de ellos posee al menos un nivel (planta baja o nivel 0 (cero)), como puede ser una casa particular, aunque puede suceder que el espacio contenga más de un nivel como es el caso de edificios, hospitales y universidades. Un nivel está compuesto por un conjunto de ambientes y accesos, y los mismos se comunican entre ellos mediante accesos del tipo *"EntreNivel"*.

3. **Acceso:** Todo espacio indoor posee la cualidad de ser navegable, esto significa que una persona debe ser capaz de transitar entre los distintos niveles y ambientes que existen dentro del espacio. Para poder cumplir con esta cualidad definimos a los accesos, como aquellos componentes a través de los cuales se accede a los distintos ambientes o niveles del espacio indoor, permitiendo la navegación del mismo. Los clasificamos de la siguiente manera:

3.1 **EntreNivel:** Son aquellos accesos que permiten la movilidad de una persona entre los distintos niveles del espacio indoor. Entre ellos:

3.1.1 **Rampa:** Es un acceso de tipo *"EntreNivel"* que permite vincular dos ambientes que se encuentran en diferentes niveles. Lo que permite la rampa es descender o ascender a uno u otro ambiente a través de su superficie. Este tipo de accesos confiere mayor accesibilidad dentro de los espacios indoor debido a que facilitan el desplazamiento dentro del mismo a personas con movilidad reducida (ancianos, personas que utilizan silla de ruedas, etc).

3.1.2 **Escalera:** Es un acceso de tipo *"EntreNivel"* que comunica dos ambientes diferentes que se hallan separados por una distancia determinada en diferentes niveles. Los escalones son los elementos más importantes de una escalera, pueden variar en términos de grosor, anchura, material, superficie y distancia entre uno y otro. Un acceso de este tipo, además, puede ser mecánica.

3.1.3 **Ascensor:** Es un acceso de tipo *"EntreNivel"* que se desempeña como la principal vía de transporte de personas dentro del espacio indoor, permitiéndoles a través de él, subir o bajar a través de los niveles que disponga el espacio indoor en cuestión. Por lo general son eléctricos, por lo tanto, cuando no hay energía eléctrica dejan de funcionar.

3.2 **EnNivel:** Son aquellos accesos que permiten que sea posible la movilidad de una persona entre los distintos ambientes conectados del espacio indoor dentro del mismo nivel. Entre ellos:

3.2.1 **Arcada:** Es un acceso de tipo *"EnNivel"* que se caracteriza por ser una abertura en una pared ofreciendo paso libre entre los ambientes que conecta.

3.2.2 **Puerta:** Es un acceso de tipo *"EnNivel"* que se caracteriza por ser un objeto generalmente de madera, vidrio o metal, ubicado en una abertura en una pared, que se abre y se cierra permitiendo de esta manera el paso o bloqueo de una persona entre los ambientes que conecta. También permite conectar al espacio indoor con el espacio outdoor.

4. **Sensores:** Son dispositivos físicos capacitados para detectar acciones o estímulos externos y responder en consecuencia. Están ubicados en los ambientes y permite obtener información de lo que sucede dentro de los mismos. Podemos nombrar distintos tipos de sensores, entre ellos, de humo, de alarma, de temperatura. Para los fines de esta tesina solo nos interesan los Sensores de Posicionamiento.

4.1 **Posicionamiento:** Determinan la posición aproximada de una persona dentro del ambiente.

4.3.2 Características de conceptos

Luego de clasificar y obtener los conceptos comunes entre los espacios indoor seleccionados (mencionados en la sección anterior), nuestra investigación continuó en refinar y destacar las propiedades más relevantes de dichos conceptos. El propósito de esta sección es nombrar y definir dichas propiedades.

- El concepto **Espacio Indoor** posee las siguientes características:
    - **Descripción:** el valor de esta propiedad indica el propósito general del Espacio Indoor.
    - **Nombre:** el valor de esta propiedad indica el nombre del Espacio Indoor.
    - **Teléfono:** el valor de esta propiedad indica el/los teléfono/s de contacto existentes en el Espacio Indoor.
    - **Ubicación:** el valor de esta propiedad indica la ubicación física (domicilio) del Espacio Indoor.
- El concepto **Ambiente** posee las siguientes características:
    - **Alto:** el valor de esta propiedad indica la altura en metros del ambiente.
    - **Ancho:** el valor de esta propiedad indica el ancho en metros del ambiente.
    - **Profundidad:** el valor de esta propiedad indica la profundidad en metros del ambiente.
    - **Descripción:** el valor de esta propiedad indica el propósito del ambiente.
    - **Nombre:** el valor de esta propiedad indica el nombre del ambiente.
- El concepto **Nivel** posee las siguientes características:
    - **Descripción:** el valor de esta propiedad indica el propósito del nivel.
    - **Número:** el valor de esta propiedad indica el número del nivel.
- El concepto **Acceso** posee las siguientes características:
    - **ConectaExterior:** el valor “true” en esta propiedad indica que el acceso conecta el espacio indoor con el espacio outdoor; en caso contrario el valor de esta propiedad será “false”.
    - **EsDeEmergencia:** el valor “true” en esta propiedad indica que el acceso puede utilizarse en situaciones de emergencia; en caso contrario el valor de esta propiedad será “false”.

- **Posición:** El valor de esta propiedad indica la coordenada geográfica en la cual se ubica el acceso.
- **Nombre:** el valor de esta propiedad indica el nombre del acceso.

- El concepto **EnNivel** posee las siguientes características:
  - **Alto:** el valor de esta propiedad indica la altura en metros del acceso.
  - **Ancho:** el valor de esta propiedad indica el ancho en metros del acceso.
  - **AptoDiscapacitados:** el valor "true" en esta propiedad indica que cualquier persona independientemente de sus capacidades puede atravesar el acceso; en caso contrario el valor de esta propiedad será "false". Por ejemplo, una puerta con un escalón no es apta para ser atravesada por alguien que utiliza silla de ruedas.
    - **Puerta**
      - **ModoApertura:** el valor de esta propiedad indica el modo de apertura de la puerta (hacia adentro, hacia afuera o corrediza).

- El concepto **EntreNivel** posee las siguientes características:
  - **Rampa**
    - **Longitud:** el valor de esta propiedad indica la longitud en metros de la rampa.
    - **Pendiente:** el valor de esta propiedad indica lo grados de inclinación que posee la rampa.
  - **Escalera**
    - **CantEscalones**: el valor de esta propiedad indica la cantidad de escalones que posee la escalera.
    - **Mecánica:** el valor "true" en esta propiedad indica que la escalera funciona mecánicamente; en caso contrario el valor de esta propiedad será "false".
    - **Pendiente:** el valor de esta propiedad indica lo grados de inclinación de la escalera.
  - **Ascensor**
    - **cantAccesos:** el valor de esta propiedad indica la cantidad de accesos que posee el ascensor.

- **capacidadMaxima:** el valor de esta propiedad indica la cantidad máxima de personas que permite el ascensor.
- **pesoMaximo:** el valor de esta propiedad indica la cantidad de kilos que soporta el ascensor.
- **eléctrico:** el valor "true" en esta propiedad indica que el ascensor es eléctrico; "false" indica que es hidráulico.

- El concepto **Sensor** posee las siguientes características:
  - **Descripción:** el valor de esta propiedad indica el propósito del sensor.
  - **Nombre:** el valor de esta propiedad indica el nombre del sensor.
    - **SensorDePosicionamiento**
      - **Posición:** El valor de esta propiedad indica la coordenada dentro del ambiente en la cual se ubica el sensor.

4.4 Metamodelo propuesto

4.4.1 Desarrollo en etapas

Uno de los objetivos principales de esta tesina es lograr obtener un metamodelo que represente y permita trazar diferentes caminos para recorrer los espacios indoor. Para ello, se realizó un análisis exhaustivo de los conceptos comunes junto con las características propias de cada uno, las cuáles fueron desarrolladas en la sección 4.3, y la forma en que estos conceptos se relacionan entre sí.

El desarrollo del metamodelo se realizó de manera iterativa e incremental, atravesando distintas etapas en las cuales nos propusimos diferentes metas, en las cuales se muestra como el metamodelo puede ir mejorándose a través de la aplicación de patrones de diseño y otras buenas prácticas de modelado. En un principio, sólo nos concentramos en modelar los conceptos comunes y lograr que éstos se relacionen entre sí. Una vez refinado este modelo, pasamos a la siguiente meta que se basa en poder recorrer y obtener distintos caminos en el espacio indoor.

En esta sección explicaremos las distintas aproximaciones que obtuvimos hasta alcanzar el metamodelo final, que cumple con los objetivos mencionados en el párrafo anterior.

Primera Aproximación

El objetivo de esta primera aproximación fue definir las metaclases principales que integran un espacio indoor y las relaciones que existen entre cada una de ellas.

- Como se nombro en la sección anterior, un **EspacioIndoor** es un espacio construido, es decir, está compuesto por paredes, pisos y techos, por lo tanto, debe estar comprendido por uno o varios **Ambientes**. Por ejemplo, si pensamos que el espacio indoor representa una casa particular, los ambientes podrían ser: cocina, baño, comedor, dormitorio, entre otros; en cambio, si representa un hospital, estos ambientes podrían ser: consultorio, quirófano, habitación. Si un espacio indoor no contara con ningún ambiente, entonces no cumpliría con la propiedad de ser construido y no se lo podría denominar EspacioIndoor. Para poder representar todos los ambientes que existen en el espacio indoor creamos la relación **espacioAmbientes** que indica qué ambientes contiene el espacio.

  Como dijimos en el apartado anterior, un **EspacioIndoor** debe estar compuesto por al menos un **Ambiente**, y para poder acceder a ese ambiente se necesitarán uno o varios **Accesos**. Por lo tanto, definimos una relación **espacioAccesos** que representa todos los accesos que existen en el espacio indoor. Como se mencionó anteriormente, dichos accesos pueden pertenecer a la metaclase **EnNivel**, por ejemplo, si pensamos que el espacio indoor representa una casa particular, los accesos podrían ser: puerta, arcada, pasillo; o a la metaclase **EntreNivel**, que si pensamos en un espacio indoor con más de un nivel; generalmente en una casa particular un acceso de este tipo estaría representado por una escalera; y en un hospital podría existir tanto una escalera como un ascensor.

- Para poder lograr ubicar a una persona dentro del espacio indoor o para dar seguridad a dicho espacio, un **EspacioIndoor** puede contener uno o varios **Sensores** que pueden ubicarse tanto en **Ambientes** como en **Accesos**. En una casa particular, podría existir un sensor de humo en la cocina, sensores de alarma en las puertas, de temperatura en los dormitorios, etc. Los sensores que existen dentro del espacio indoor están representados por la relación **espacioSensor**,

- Para poder tanto acceder como salir de un **Ambiente** necesitamos un acceso del tipo **EnNivel**. Por lo tanto, creamos la relación **ambienteAcceso** que nos indica para un ambiente cuáles son sus posibles accesos.

- Un **Ambiente** puede estar compuesto por otros ambientes. Por ejemplo, en una casa podría existir un ambiente compuesto, llamado “living comedor” con un único acceso, pero dentro de ese ambiente, existen dos ambientes simples: el living, y el comedor, los cuales podrían estar conectados por una arcada. Esta situación la representamos con la relación **compuestoPor**.

- Notamos que una información útil sería que un **Nivel** conociera todos los **Ambientes** que se encuentran en él. Por ejemplo, en un hospital podría existir el nivel número 5 en donde se encuentran todas las habitaciones del tipo internación. Con este fin, definimos la relación **nivelAmbiente**.

- Coincidimos en que sería importante que un **Nivel** pueda contar con la información que indica a través de qué **Accesos** de tipo **EntreNivel** se puede llegar a él. Por ejemplo, en una casa particular se puede acceder al nivel número 1 a través del acceso **Escalera**. Esta información la brida la relación **nivelAcceso**.

De esta manera logramos cumplir con el objetivo de la primera aproximación, modelar los conceptos comunes, y la manera en que se comunican entre sí.

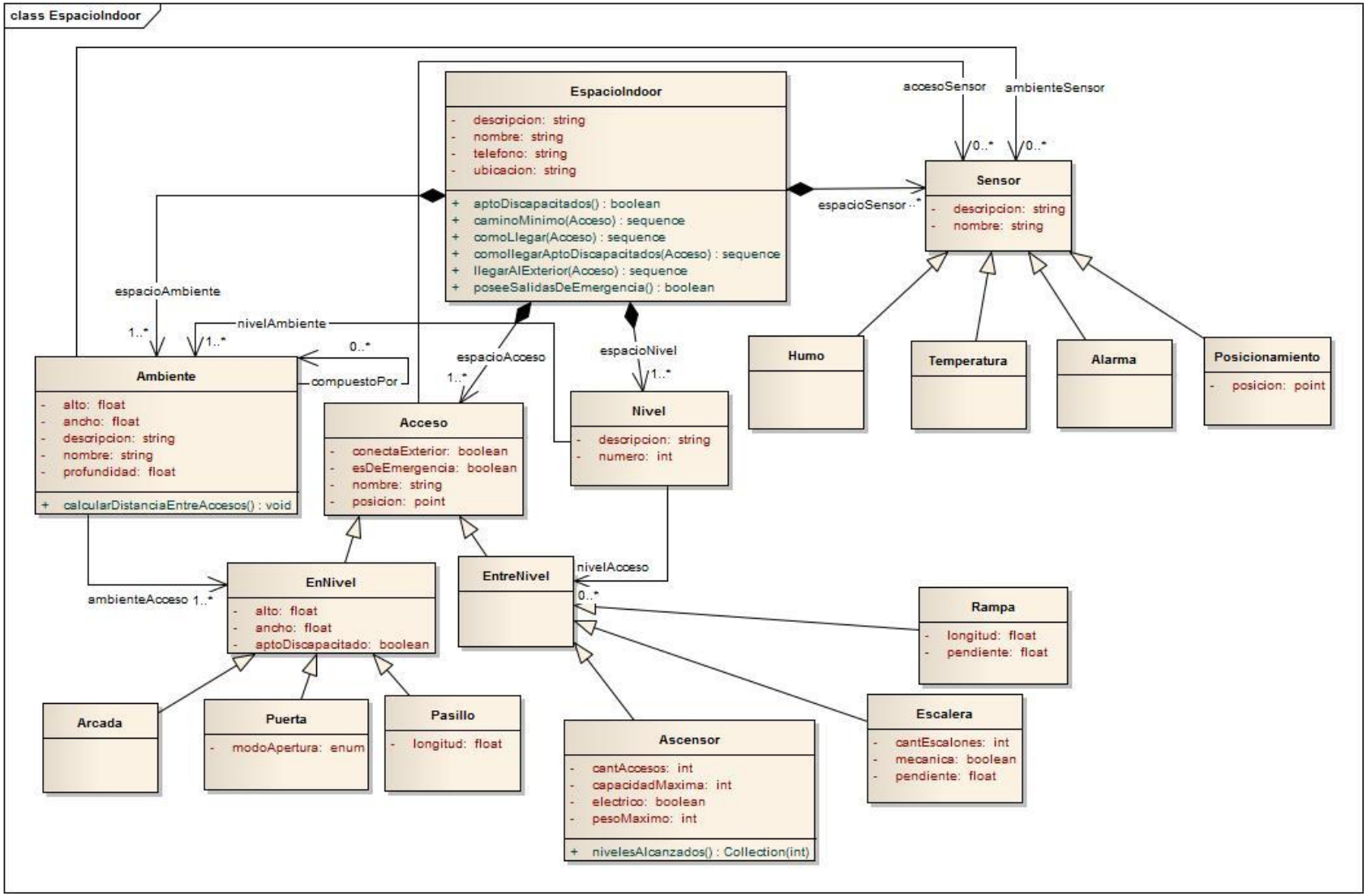


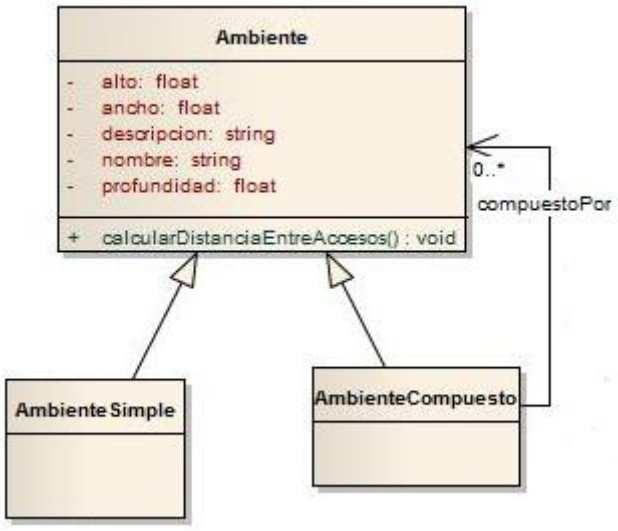
Figura 4.1 - *Metamodelo "Espacio Indoor": Primera Aproximación*

Segunda Aproximación

El objetivo de la segunda aproximación es refinar el modelo presentado en la primera aproximación. Con este fin realizamos dos grandes modificaciones:

1. Aplicamos el Patrón de Diseño Composite[1] en la metaclase **Ambiente** ya que notamos que un ambiente puede formar parte de otro, es decir, puede ser un contenedor de ambientes más pequeños. Los ambientes que contienen a otros ambientes son los que llamamos **AmbienteCompuesto** y posee una relación **compuestoPor** que permite conocer los ambientes por los que está compuesto. Los ambientes que componen un ambiente compuesto pueden ser tanto compuestos como simples.

   Tomamos esta decisión luego de consultar y analizar el modelo que implementa la "Máquina de Estados UML" [24], en donde una máquina de estado puede contener una o más máquinas de estados dentro.



[1] El patrón Composite se utiliza para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que, al poseer todos ellos una interfaz común, se tratan todos de la misma manera (Ver Anexo I).

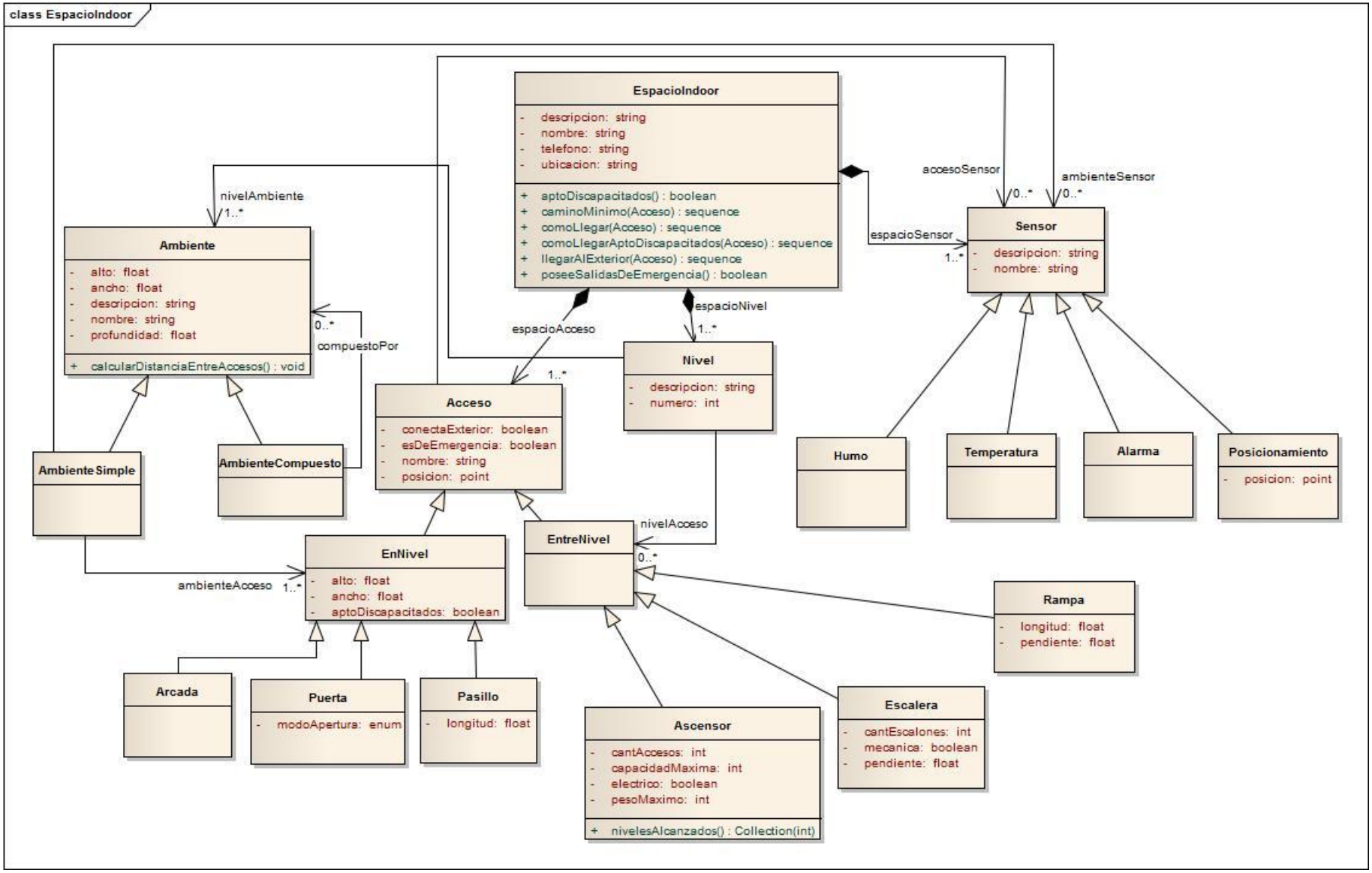
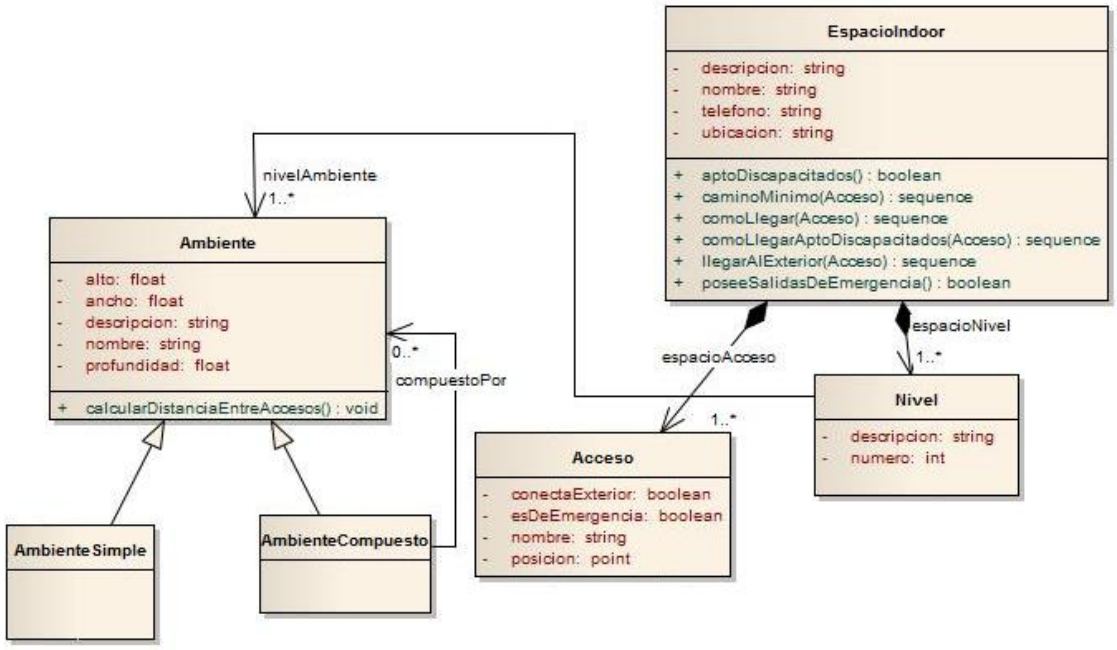


Figura 4.2 - *Metamodelo "Espacio Indoor": Segunda Aproximación*

2. Modificamos la cardinalidad entre las metaclases **EspacioIndoor** y **Nivel**: en la primera aproximación esta cardinalidad era (0,*), lo que indicaba que podían existir espacios indoor que no tuvieran niveles, obligando al **EspacioIndoor** a conocer todos los ambientes que lo componen. En la segunda aproximación cambiamos la cardinalidad entre las metaclases **EspacioIndoor** y **Nivel** a (1,*), con lo cual cualquier espacio indoor tiene al menos un nivel, ya sea planta baja o nivel 0 (cero). Con esta modificación pudimos eliminar la relación **espacioAmbiente** entre **EspacioIndoor** y **Ambiente**, ya que cada nivel conoce los **Ambientes** que lo componen.



Tercera Aproximación

En esta aproximación comenzamos a incorporar el concepto de posicionamiento dentro del modelo.

Uno de los objetivos principales de la tesina es lograr que nuestro metamodelo represente espacios indoor y que, a través del comportamiento que tengan las clases, logremos resolver problemas como, por ejemplo, obtener caminos que indiquen como llegar desde un ambiente origen a un ambiente destino, alcanzar la salida del espacio indoor, poder conocer las rutas de emergencia, etc. Con este fin consultamos el metamodelo presentado en el paper denominado: *"Representaciones enriquecidas para la navegación indoor-outdoor en aplicaciones móviles"* [4], en el cual aparecen ciertos conceptos que ayudan a la navegación dentro de un espacio indoor.

1. Dividimos el concepto de **Acceso** en **Acceso** y **Conectores**, donde los conectores nos permiten movilizarnos de un ambiente a otro, y los accesos nos permiten entrar y salir de dicho ambientes.

2. Incorporamos el concepto de **RedDeCirculacion** que almacena el camino que conecta el espacio indoor con el resto de los componentes. Es decir, almacena el grafo del espacio indoor.

3. Definimos la metaclase **Tramo** que habilita la posibilidad de que un conector esté compuesto por varios accesos; un tramo queda definido como la distancia que hay entre cada acceso del conector.

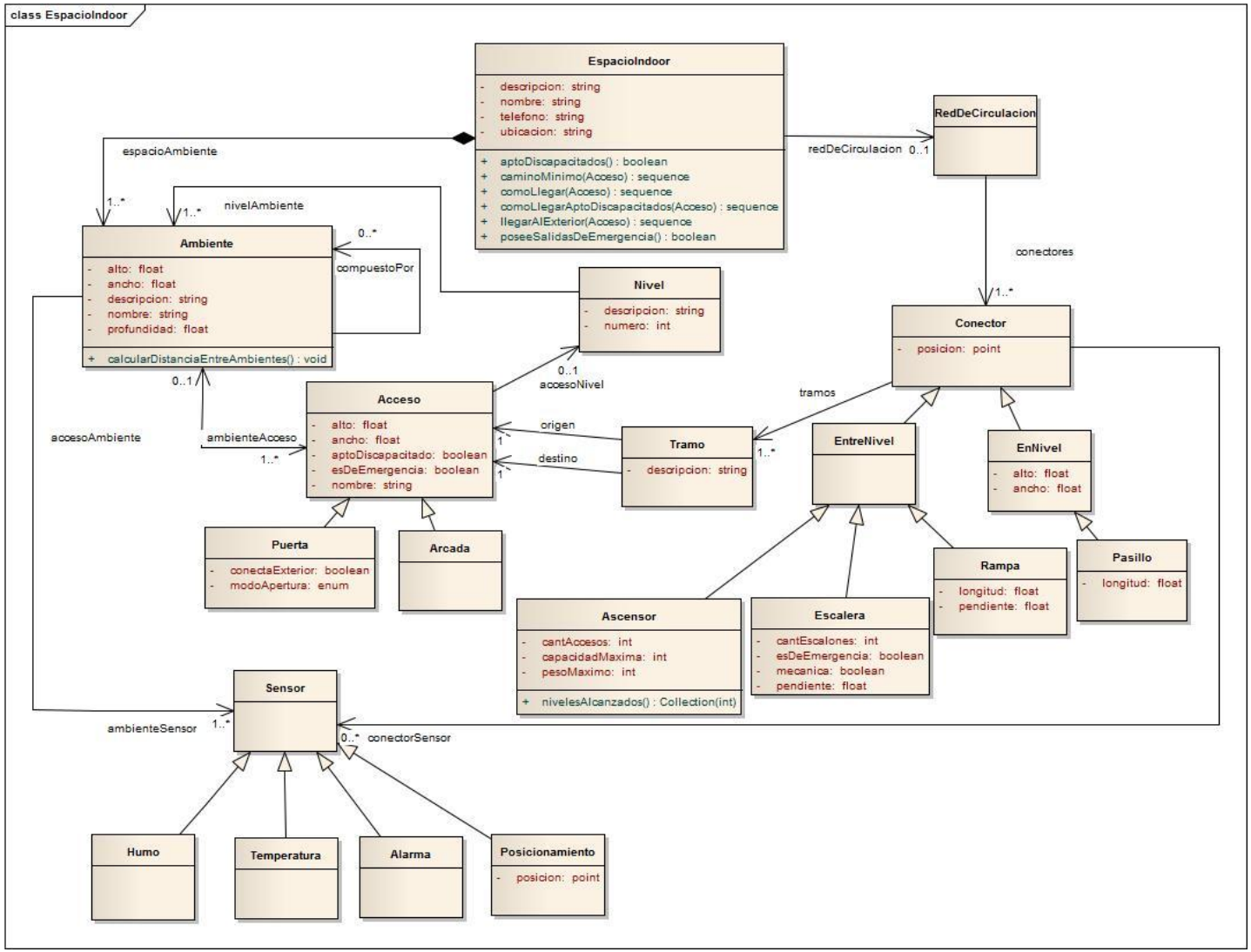


Figura 4.3 - *Metamodelo "Espacio Indoor": Tercera Aproximación*

Cuarta Aproximación

El objetivo de esta instancia es comenzar a refinar el metamodelo presentado en la aproximación anterior. Para lograrlo realizamos los siguientes cambios:

1. Eliminamos la metaclase **RedDeCirculacion** ya que notamos que a nivel modelo de clases no tiene relevancia. El grafo **RedDeCirculacion** representado y sobre el que aplicaremos los algoritmos de búsqueda de caminos se generará al momento en que se persistan los datos del espacio indoor en la base de datos.

2. Eliminamos el concepto **Conector** y **Tramo** ya que para los fines de nuestra tesina no necesitamos conocer los distintos tramos que forman un conector; es suficiente saber que existen accesos que nos permiten transitar dentro y entre los ambientes.

3. Modificamos el concepto de **Pasillo**: en todas las aproximaciones anteriores, representamos al pasillo como un conector/acceso ya que nos permitía alcanzar los ambientes. En esta aproximación lo representamos como un **AmbienteSimple**, ya que respeta las características de un ambiente. Por lo tanto un **Pasillo** ahora es un ambiente simple con todas las propiedades del mismo.

4. Eliminamos de la jerarquía **Sensores** aquellos que no son necesarios para cumplir con nuestros objetivos, dejando solamente **Posicionamiento**.

5. Basándonos en la modificación 3 y 4, podemos decir que los sensores de posicionamiento siempre van a están ubicados en un ambiente, ya que no tendría sentido que estos tipos de sensores estén conectados en los accesos.

6. Incorporamos la metaclase **TipoDeAcceso** con el fin de poder realizar una clasificación más exacta de los accesos.

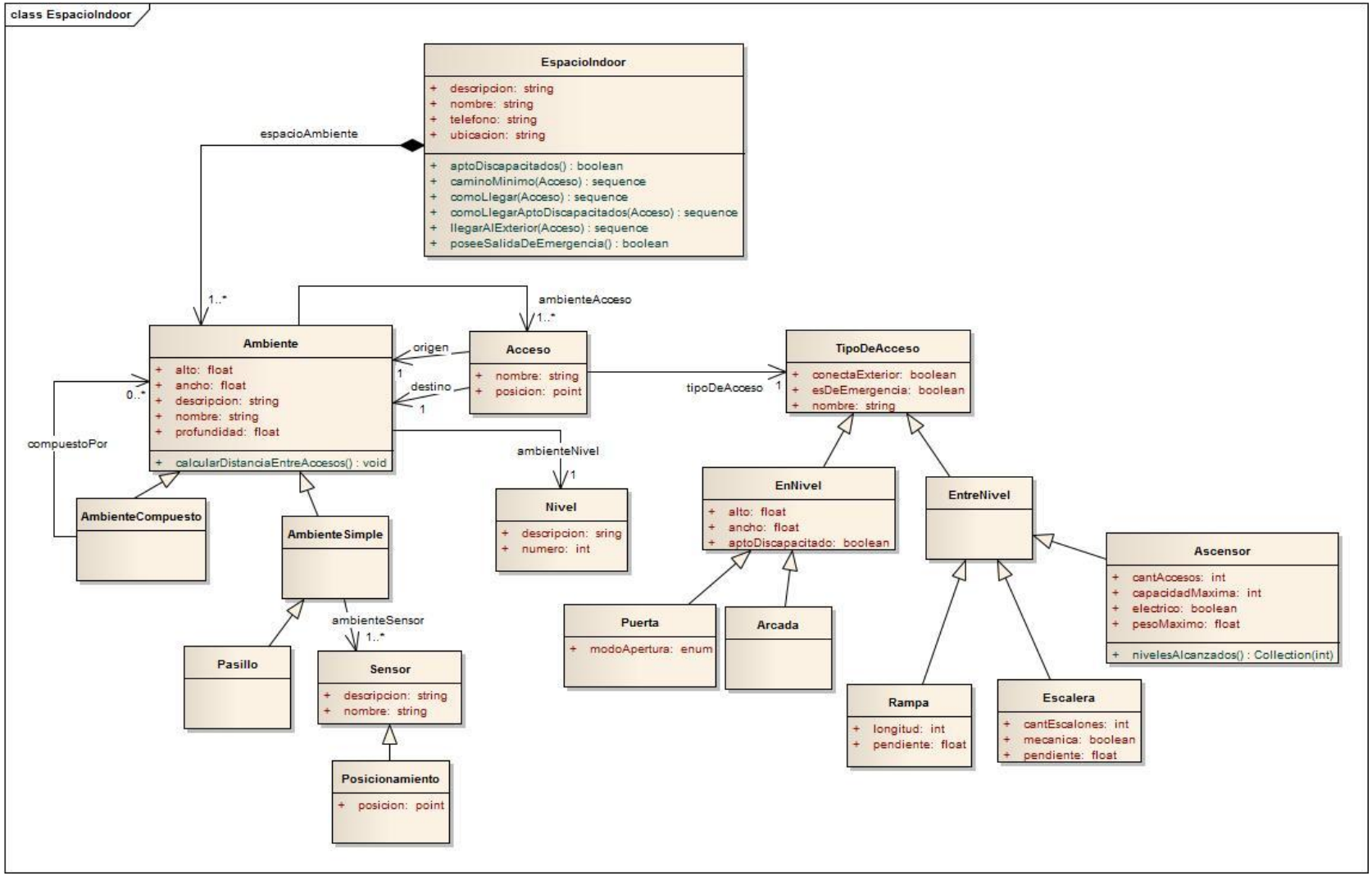


Figura 4.4 - *Metamodelo Espacio Indoor: Cuarta Aproximación*

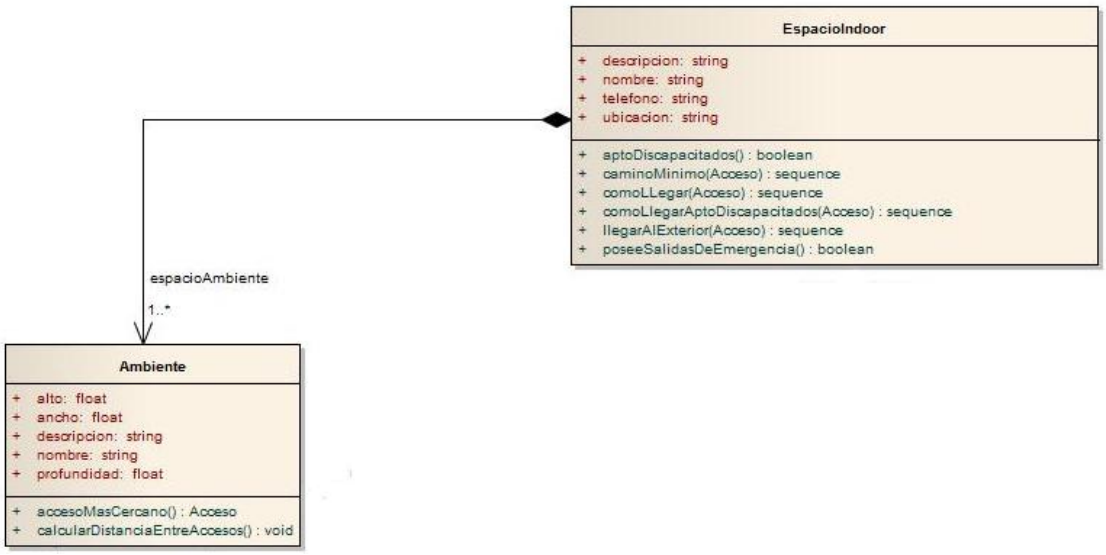
4.4.2 Metamodelo final

En cada una de las aproximaciones que se nombraron en la sección anterior, se encontraron falencias al modelo presentado, las cuales se fueron subsanando en las siguientes instancias hasta obtener el metamodelo que finalmente cumple con el objetivo principal de esta tesina, permitir modelar cualquier espacio indoor en general, es decir, lograr que sea lo suficientemente flexible como para poder diseñar cualquier espacio indoor que exista.

En la Figura 4.5 presentamos nuestro metamodelo final y, en las siguientes secciones describimos su comportamiento.

4.4.2.1 Relaciones entre metaclases del metamodelo

- Un **EspacioIndoor** posee al menos un **Ambiente**, especificado en la relación **espacioAmbiente**.



- Un **EspacioIndoor** posee al menos un **Acceso** especificado en la relación **espacioAcceso**. Si bien esta relación contiene información redundante (información a la que se podría acceder a través de otras relaciones), concluimos que es igualmente válida ya que permite optimizar ciertos métodos de búsqueda dentro del espacio indoor. Por ejemplo si quisiéramos conocer si el espacio indoor cuenta con salidas de emergencia, nos bastaría con recorrer la colección **espacioAccesos** hasta que encontremos un acceso que cumpla con dicha característica. Si esta relación no existiera deberíamos obtener esta información a través de la relación **espacioAmbiente** y luego preguntarle a cada uno de los ambiente si alguno de sus accesos, a través de la relación **ambienteAcceso**, cumple con la condición de ser de emergencia.

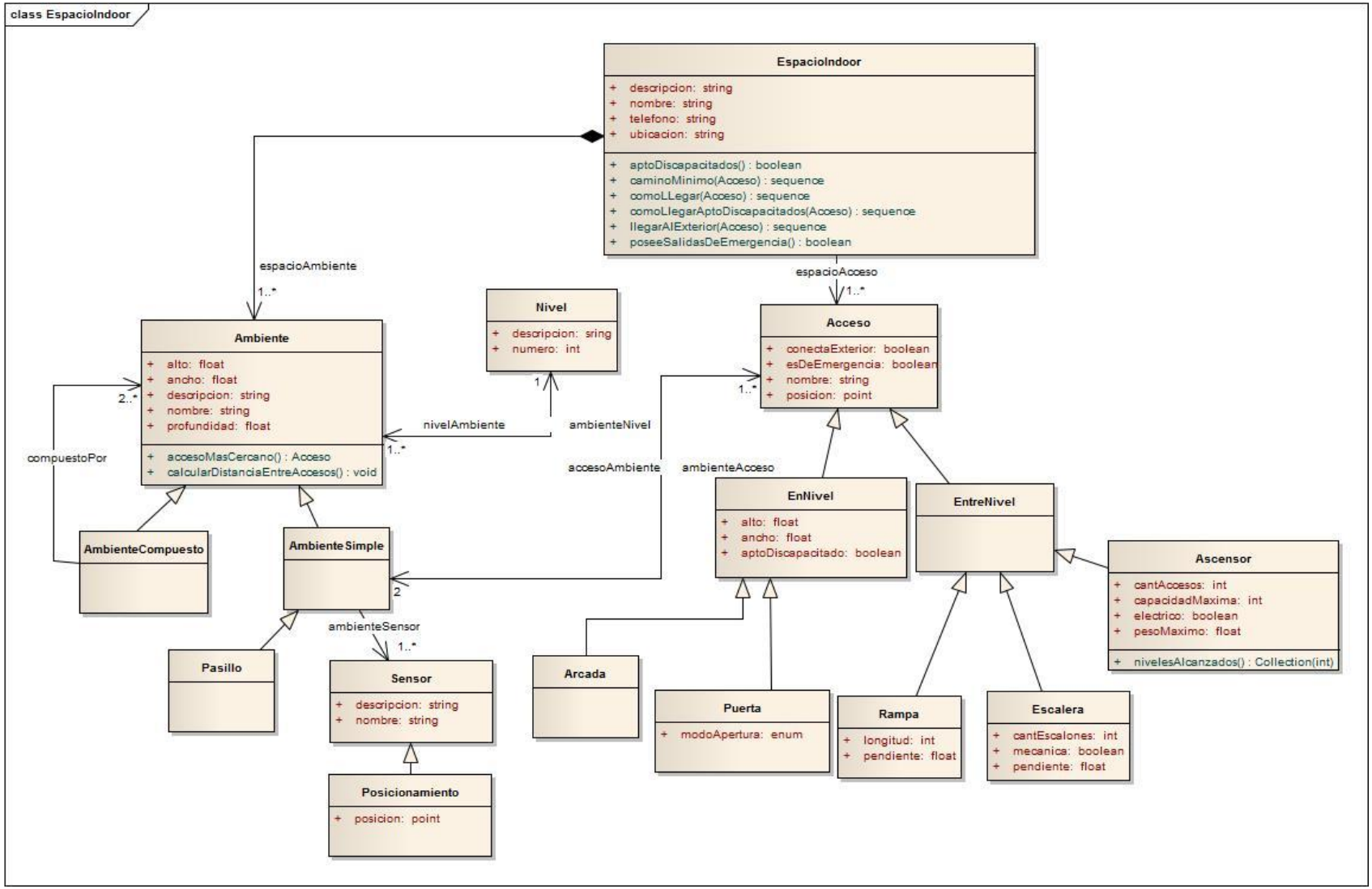
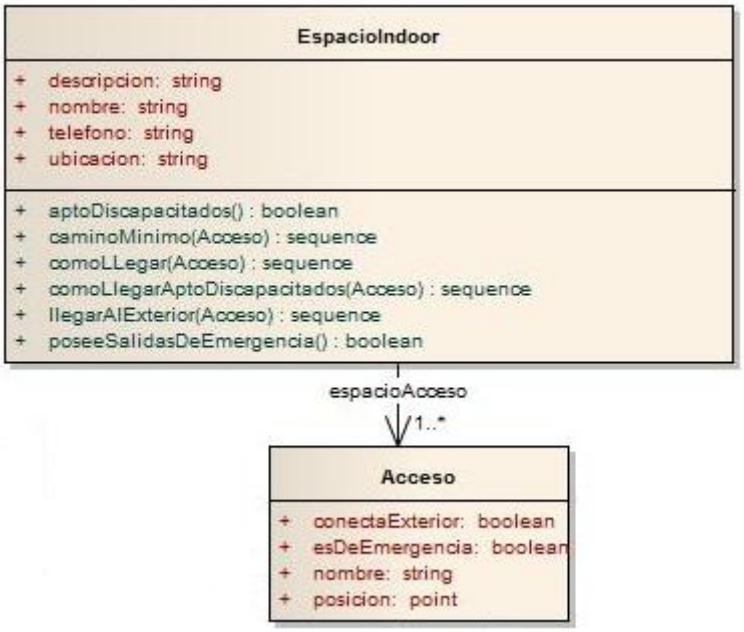
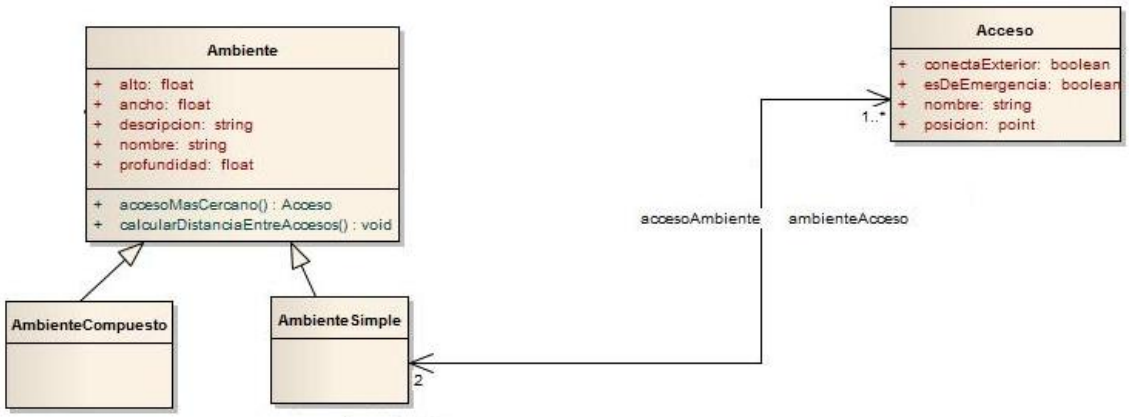


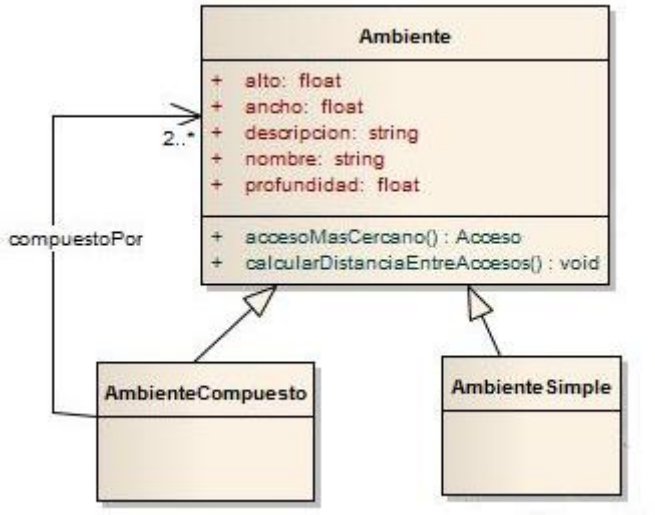
Figura 4.5 - *Metamodelo "Espacio Indoor": Metamodelo Final*



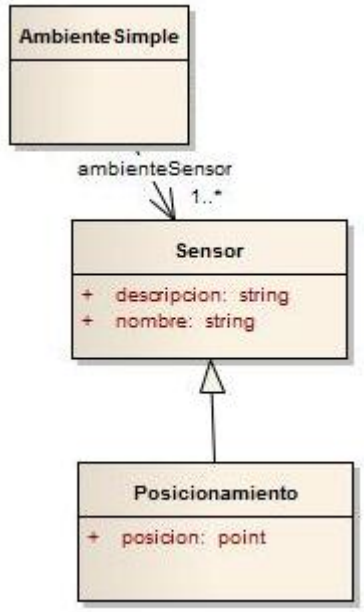
- Entre las metaclases **AmbienteSimple** y **Acceso** existe una relación bidireccional que indica que un ambiente tiene que tener al menos un acceso (relación **accesoAmbiente**), y que un acceso conecta al menos dos ambientes (relación **ambienteAcceso**). Cabe aclarar que existe una excepción para esta condición que indica que el acceso que tiene seteado su atributo **conectaExterior** en true, solo va a conectarse con un ambiente ya que el exterior no cuenta como tal. Esta relación fue creada en reemplazo de las tres relaciones existentes en las aproximaciones 3 y 4: accesos, origen y destino, con el fin de simplificar la lectura del metamodelo.



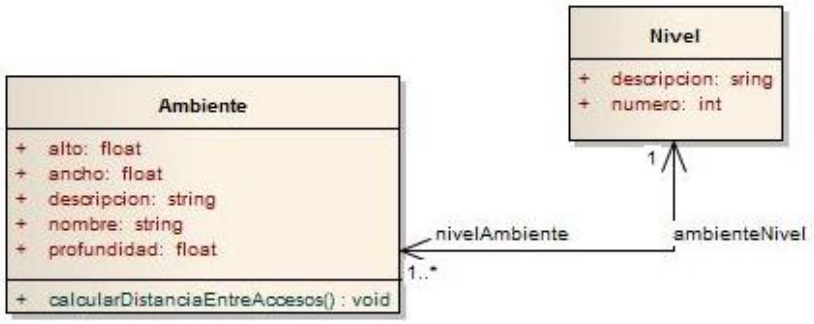
- Los ambientes que contienen a otros ambientes son llamados **AmbienteCompuesto** y poseen una relación **compuestoPor** (ver Anexo I) que permite conocer los ambientes por los que está compuesto. Dichos ambientes pueden ser compuestos o simples. Además, agregamos la restricción de que, para que un ambiente cumpla con las condiciones de ambiente compuesto, debe contener como mínimo dos ambientes (sean simples o compuestos), sin restricción de la cantidad máxima.



- Un **AmbienteSimple** debe tener al menos un **SensorPosicionamiento**, especificado en la relación **ambienteSensor**, el cual brindará información sobre la ubicación del usuario dentro del ambiente.



- Entre las metaclases **Ambiente** y **Nivel** existe una relación bidireccional que indica que un ambiente pertenece a un único nivel (relación **nivelAmbiente**), y que un Nivel conoce todos los ambientes que forman parte de él (relación **ambienteNivel**).



4.4.2.2 Comportamiento del metamodelo

En esta sección se desarrolla el propósito de los métodos declarados en cada metaclase del metamodelo final:

- **EspacioIndoor**
  - *aptoDiscapacitados():boolean* -> retorna “true” si el espacio indoor puede ser recorrido en su totalidad por personas con incapacidades motoras, “false” en caso contrario.
  - *caminoMinimo(acceso):sequence* -> de todos los caminos posibles para llegar desde la posición donde se encuentra el usuario, la cual es captada por el sensor de posicionamiento más cercano a ella, hasta el acceso recibido por parámetro, retorna aquel que es mínimo respecto a la distancia.
  - *comoLlegar(accesoOrigen, accesoDestino):sequence* -> retorna aleatoriamente un camino posible para llegar desde el accesoOrigen al accesoDestino.
  - *comoLlegarAptoDiscapacitados(accesoOrigen, accesoDestino):sequence* -> retorna, si existe, un camino que permita llegar desde el accesoOrigen al accesoDestino, teniendo en cuenta que todos los accesos intermedios deben tener seteado el atributo aptoDiscapacitados en true.
  - *llegarAlExterior(acceso):sequence* -> retorna aleatoriamente un camino posible para llegar desde el acceso que indica el usuario, hasta algún acceso que se comunique con el exterior.
  - *poseeSalidasDeEmergencia():boolean* -> retorna “true” si el espacio indoor posee algún acceso que pueda ser utilizado en caso de emergencia, “false” en caso contrario.

- **Ambiente**
    - *calcularDistanciaEntreAccesos():void* -> calcula la distancia entre todos los accesos de un ambiente.
    - *accesoMasCercano():void* -> en base a la posición donde se encuentra el usuario, la cual es captada por el sensor de posicionamiento, calcula cuál es el acceso más cercano al usuario.
- **Ascensor**
    - *nivelesAlcanzados():Collection(int)* -> retorna una colección con los números de los niveles que el ascensor puede alcanzar.

4.4.3 Restricciones

UML es un lenguaje para especificar, construir, visualizar y documentar los objetos de un sistema intensivo de software.

Un diagrama UML, como puede ser un diagrama de clases, normalmente no está lo suficientemente refinado y, por tanto, no logra reflejar todos los aspectos relevantes de una especificación. Por esta razón surge, entre otras cosas, la necesidad de describir restricciones adicionales sobre los objetos en el modelo. Estas restricciones son particularmente útiles, en la medida en que permiten a los desarrolladores crear un amplio conjunto de reglas que rigen el aspecto de un objeto individual, logrando así, enriquecer el modelo sobre el que se definen.

4.4.3.1 Restricciones sobre el metamodelo final

A continuación nombramos las restricciones que definimos para enriquecer el metamodelo final de espacio indoor obtenido en la sección anterior, junto con su notación en leguaje OCL:

- Para todas las metaclases el atributo **nombre** debe ser obligatorio.

```
Context EspacioIndoor inv:

   self.nombre<> ''
```

```
Context Ambiente inv:

   self.nombre<> ''
```

```
Context Acceso inv:

   self.nombre<> ''
```

- Un **EspacioIndoor** debe tener al menos un **Acceso** cuyo atributo **conectaExterior** este seteado con el valor true

```
Context EspacioIndoor inv:

self.acceso -> exists (a: Acceso | a.conectaExterior)
```

- Cada **Ambiente** debe tener al menos un **Acceso**

```
Context Ambiente inv:

   self.acceso -> size() > 0
```

- Cada **Ambiente** debe tener al menos un **SensorDePosicionamiento**

```
Context Ambiente inv:

 self.sensor -> exists(s:Sensor /
                       s.isTypeOf(Posicionamiento())
```

- Un **Acceso** de tipo **EntreNivel** debe conectar dos ambientes de distinto **Nivel**.

```
Context EntreNivel inv:

     self.ambiente -> forAll(a1, a2 : Ambiente
       |a1 <> a2 implies a1.nivel.numero <> a2.nivel.numero)
```

- Un **Acceso** de tipo **EntreNivel** debe conectar dos ambientes con **Niveles** sucesivos

```
Context EnNivel inv:

   self.ambiente -> forAll((a1, a2 : Ambiente
              | ((a1.nivel.numero + 1) = a2.nivel.numero)
             OR((a1.nivel.numero - 1) = a2.nivel.numero)
```

- Un **Acceso** de tipo **EnNivel** debe conectar dos **Ambientes** que se encuentren en el mismo **Nivel**.

```
Context EnNivel inv:

   self.ambiente -> forAll(a1, a2 : Ambiente
      |a1 = a2 implies a1.nivel.numero = a2.nivel.numero)
```

- Cada **Acceso** debe conectar dos **Ambientes** distintos.

```
Context Acceso inv:

  self.ambiente -> self.ambiente at(1) <> self.ambiente at(2)
```

- El atributo **ancho** de un **TipoDeAcceso EnNivel** debe ser mayor o igual que 0.80 mts.

```
Context EnNivel inv:

   self.ancho >= 0.8
```

- El atributo **alto** de un **TipoDeAcceso EnNivel** debe ser mayor o igual que 2 mts.

```
Context EnNivel inv:

   self.alto >= 2
```

- El atributo **cantEscalones** de un **TipoDeAcceso Escalera** debe ser mayor que 0

```
Context Escalera inv:

   self.cantEscalones > 0
```

- El **TipoDeAcceso Escalera** no puede tener seteado el atributo **esDeEmergencia** en true si su atributo **mecánica** esta seteado en true

```
Context Escalera inv:

  if (self.mecanica) then self.esDeEmergencia = false
```

- El atributo **cantAccesos** de un **TipoDeAcceso Ascensor** debe ser mayor que 0

```
Context Ascensor inv:

   self.cantAccesos > 0
```

- El atributo **capacidadMaxima** de un **TipoDeAcceso Ascensor** debe ser mayor que 0

```
Context Ascensor inv:

   self.capacidadMaxima > 0
```

- El **TipoDeAcceso Ascensor** no puede tener seteado el atributo **esDeEmergencia** en true

```
Context Ascensor inv:

   self.esDeEmergencia = false
```

- En un mismo **Nivel** no pueden existir **Ambientes** con el mismo nombre.

```
Context Nivel inv:
   self.ambiente -> forAll(a1, a2 : Ambiente
      | a1<>a2 implies (a1.nombre <> a2.nombre))
```

- El atributo **ubicación** de **EspacioIndoor** debe ser obligatorio

```
Context EspacioIndoor inv:

      self.ubicacion <> ''
```

- El atributo **alto** de **Ambiente** debe ser mayor que 0

```
Context Ambiente inv:

            self.alto > 0
```

- El atributo **ancho** de **Ambiente** debe ser mayor que 0

```
Context Ambiente inv:

      self.ancho > 0
```

- El atributo **profundidad** de **Ambiente** debe ser mayor que 0

```
Context Ambiente inv:

      self.profundidad > 0
```

- El atributo **posición** de **SensorDePosicionamiento** debe ser obligatorio

```
Context Posicionamiento inv:

      self.posicion <> null
```

- El atributo **posición** de **Acceso** debe ser obligatorio

```
Context Acceso inv:

      self.posicion <> null
```

- El atributo **longitud** de **Rampa** debe ser mayor que 0

```
Context Rampa inv:

      self.longitud > 0
```

- El atributo **pendiente** de **Rampa** debe ser mayor que 0

```
Context Rampa inv:

      self.pendiente > 0
```

- El atributo **pendiente** de **Escalera** debe ser mayor que 0

```
Context Escalera inv:

      self.pendiente > 0
```

4.5 Extensión del Metamodelo "Espacio Indoor"

Como explicamos anteriormente, el metamodelo *"Espacio Indoor"* es un metamodelo abstracto que permite representar cualquier espacio indoor que exista. Por lo tanto, es necesario realizar una extensión del mismo con las características particulares del espacio indoor para representar un dominio particular, y así lograr obtener un metamodelo más concreto

Para ejemplificar lo anterior decidimos desarrollar un metamodelo que refine el metamodelo *"Espacio Indoor"* para representar una institución universitaria. Para lograrlo, realizamos un análisis de las Facultades que componen a la Universidad Nacional de La Plata, con el fin de obtener los conceptos comunes entre ellas, junto con sus características. Se puede notar que este análisis es el mismo que realizamos para los espacios indoor (ver sección 4.3) pero menos general, ya que ahora nuestro grupo de estudio se basó únicamente en las facultades.

Como resultado del análisis, podemos nombrar los siguientes conceptos comunes a todas las facultades de la Universidad Nacional de La Plata y las cualidades que las caracterizan:

- **Aula:** Son **Ambientes** en donde se dictan las clases de las materias de la facultad.
    - **Aula Especial:** posee las características definidas para todas las aulas, pero además se pueden definir otras que la diferencian. Por ejemplo, en la "Facultad de Medicina" existen laboratorios que son aulas especiales, lo mismo en la "Facultad de Informática" existe la "Sala de PC", que también es un aula especial.
- **Oficina:** en todas las instituciones existen departamentos administrativos en donde los alumnos y docentes pueden realizar sus trámites. Cuentan con un horario de atención, y una manera de atención.
- **Biblioteca:** todas las facultades poseen un ambiente en donde tanto los alumnos como los profesores pueden consultar distinta bibliografía. Puede contener o no una sala de lectura.

- **Baño:** es necesario que todas las instituciones posean al menos un baño para damas y otro para caballeros. Se puede definir la capacidad y si es apto para discapacitados.

- **Servicios:** este concepto se refiere a todos aquellos beneficios que se brindan a los alumnos en la institución. Todos cuentan con un horario de atención, y entre ellos podemos nombrar:

    - **Buffet:** Ambiente destinado a la venta de comestibles en la institución. Puede poseer mesas o no.

    - **Fotocopiadora**

- **Estacionamiento:** por lo general, todas las instituciones poseen un lugar donde al menos los profesores pueden estacionar su vehículo. A este **Ambiente** se le puede definir una capacidad, una tarifa, e informar si es techado o no.

Una vez definidos estos conceptos pasamos a la siguiente instancia que consiste en realizar el diseño del metamodelo *"Facultad"* extendiendo el metamodelo *"Espacio Indoor"*. Esto significa que debemos realizar un nuevo metamodelo que herede todo el diseño y las restricciones del metamodelo *"Espacio Indoor"*, y agregar los conceptos específicos definidos para las facultades. Para ello, realizamos un estudio acerca de las distintas formas que existen para realizar este tipo de extensiones, las cuales se explican a continuación.

Al momento de extender un metamodelo podemos adaptarnos a uno de los dos mecanismos existentes para realizar esta tarea. Por un lado existe el método conservativo y por el otro el denominado método libre ó no conservativo.

Cuando hablamos de método conservativo hacemos referencia a los modelos en los que los elementos del metamodelo UML (modelo padre) no se modifican, por ejemplo agregándole comportamiento o asociaciones a las clases. Las nuevas metaclases se obtienen por herencia de las metaclases del metamodelo, a las cuales se les pueden agregar nuevos atributos, métodos, relaciones entre las otras clases del metamodelo, y restricciones OCL para especificar semántica adicional [25]. En la Figura 4.6 se muestra un ejemplo que refleja la manera de crear una nueva metaclase en el modelo extendido utilizando el método conservativo.

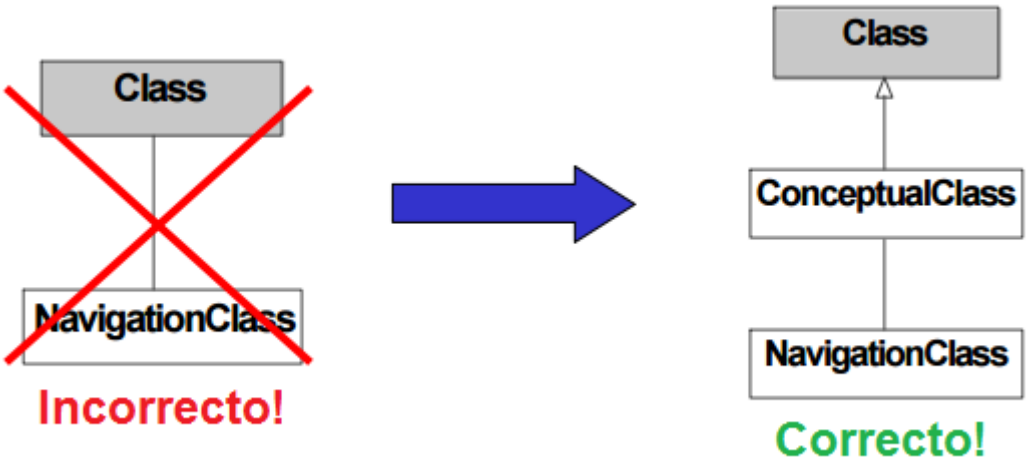


Figura 4.6 - *Extensión de metamodelo: Método Conservativo*

En contraposición al mecanismo mencionado anteriormente, el método libre o no conservativo permite, al igual que el método conservativo, crear nuevas metaclases con su propio comportamiento heredando de las metaclases del metamodelo (modelo padre), así como también permite que se modifiquen las metaclases existentes agregándoles nuevo comportamiento, nuevas características y nuevas relaciones entre las metaclases existentes. Esta técnica también permite eliminar comportamiento existente en el metamodelo padre que no se desea tener en el metamodelo extendido.

Luego de analizar los dos mecanismos, optamos por extender el metamodelo *"Espacio Indoor"* utilizando el método conservativo para obtener el metamodelo *"Facultad"* debido a que no tiene sentido modificar las metaclases del metamodelo padre ya que el mismo es genérico para cualquier espacio indoor existente.

Como se verá con más detalle en el capítulo 6, ADOxx, la herramienta de metamodelado utilizada en esta tesina, nos permite realizar una extensión del metamodelo utilizando la técnica no conservativa o liberal. ADOxx se rige bajo una licencia open source, es decir de código abierto, con lo cual todas las librerías creadas para definir los metamodelos son propiedad de la comunidad, por lo tanto, todos ellos pueden realizar modificaciones tanto del metamodelo *“Espacio Indoor”* como al metamodelo *“Facultad”*.

El metamodelo *"Facultad"* obtenido que hereda todo el comportamiento del metamodelo *"Espacio Indoor"*, diseñado con el lenguaje UML se presenta a continuación, en la Figura 4.7.

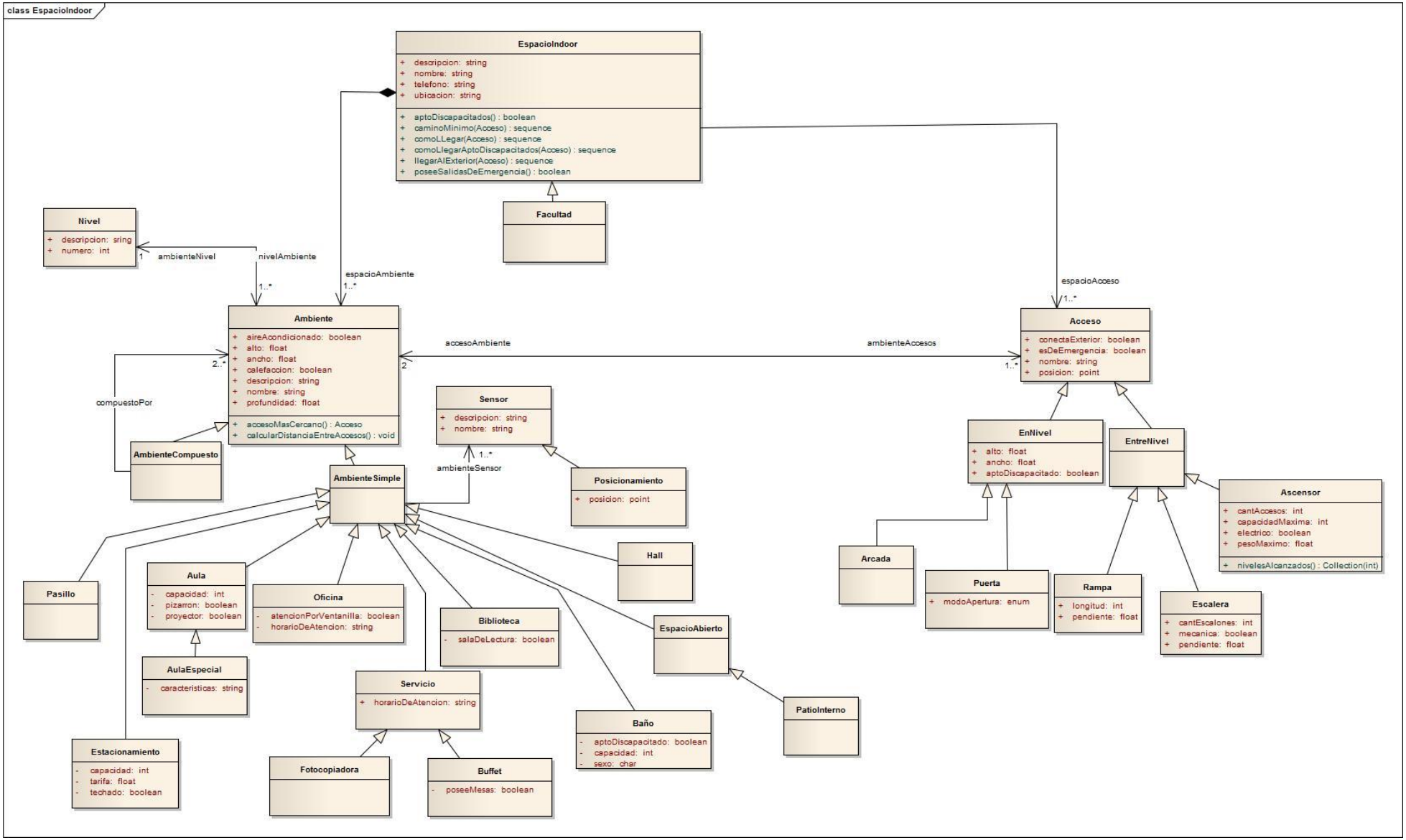


Figura 4.7*- Metamodelo "Facultad"*

Podemos notar que todos los conceptos comunes a las Facultades nombrados anteriormente están presentes en el modelo. Estos conceptos cumplen con las características definidas para los ambientes simples, por lo tanto, a través de la jerarquía definida en el metamodelo, heredan todo el comportamiento y, en cada metaclase se agrega el comportamiento propio para cada uno de ellos.

4.5.1 Restricciones sobre el metamodelo Facultad

Como explicamos anteriormente, un diagrama de clases UML no logra especificar en su totalidad los aspectos relevantes de una especificación. Por este motivo, para completar el metamodelo *"Facultad"* expuesto en la sección anterior, definimos las siguientes restricciones que lo enriquecen:

- La **Facultad** debe tener al menos una **Biblioteca**

```
Context Facultad inv:

self.ambiente -> exists (a:Ambiente /a.isTypeOf(Biblioteca())
```

- La **Facultad** debe tener al menos un **Aula**

```
Context Facultad inv:

     self.ambiente -> exists (a:Ambiente /a.isTypeOf(Aula())
```

- La **Facultad** debe tener al menos un **Baño** con el atributo **sexo** seteado en "F"

```
Context Facultad inv:

   self.ambiente exists (a:Ambiente / a.isTypeOf(Baño()) and
a.sexo = "F"
```

- La **Facultad** debe tener al menos un **Baño** con el atributo **sexo** seteado en "M"

```
Context Facultad inv:

   self.ambiente exists (a:Ambiente / a.isTypeOf(Baño()) and
a.sexo = "M"
```

- El atributo **capacidad** de **Aula** debe ser mayor que 0

```
Context Aula inv:

   self.capacidad > 0
```

- El atributo **capacidad** de **Estacionamiento** debe ser mayor que 0

```
Context Estacionamiento inv:

   self.capacidad > 0
```

- El atributo **capacidad** de **Baño** debe ser mayor que 0

```
Context Baño inv:

    self.capacidad > 0
```

4.6 Resumen

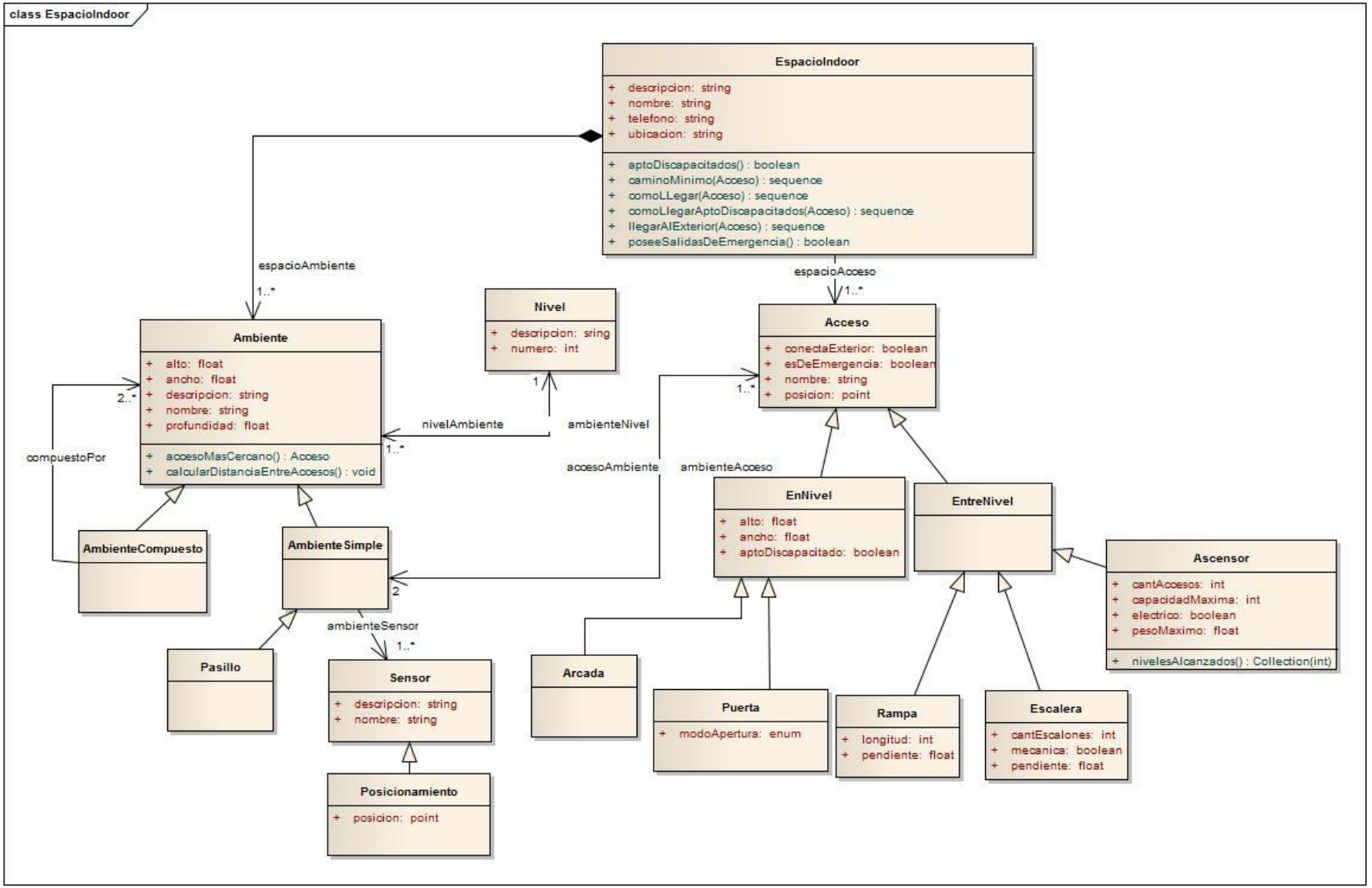
En este capítulo se detalló paso a paso cómo alcanzamos uno de los principales objetivos de esta tesina, lograr diseñar un metamodelo abstracto que represente cualquier espacio indoor.

El primer paso que atravesamos fue realizar un análisis exhaustivo de los espacios indoor en general, para lograr extraer cuáles son los componentes comunes que los integran, logrando enumerar los siguientes conceptos:

- Ambiente (simple y compuesto)
- Acceso (de tipo entre nivel o en nivel)
- Nivel
- Sensores (de posicionamiento)

Luego analizamos cada uno de ellos y definimos sus características particulares.

El siguiente paso fue diseñar el metamodelo que representa correctamente los conceptos definidos en el paso anterior. Para ello atravesamos cuatro aproximaciones donde fuimos refinando el diseño en cada una de ellas hasta lograr, en la quinta aproximación, el metamodelo final que se adecua a nuestras necesidades:


class EspacioIndoor
EspacioIndoor
+ descripcion: string
+ nombre: string
+ telefono: string
+ ubicacion: string
+ aptoDiscapacitados() : boolean
+ caminoMinimo(Acceso) : sequence
+ comoLLegar(Acceso) : sequence
+ comoLlegarAptoDiscapacitados(Acceso) : sequence
+ llegarAlExterior(Acceso) : sequence
+ poseeSalidasDeEmergencia() : boolean
espacioAmbiente
1..*
espacioAcceso
1..*
Nivel
+ descripcion: sring
+ numero: int
Acceso
+ conectaExterior: boolean
+ esDeEmergencia: boolean
+ nombre: string
+ posicion: point
Ambiente
+ alto: float
+ ancho: float
+ descripcion: string
+ nombre: string
+ profundidad: float
+ accesoMasCercano() : Acceso
+ calcularDistanciaEntreAccesos() : void
2..*
1..*
1
1..*
nivelAmbiente
ambienteNivel
compuestoPor
accesoAmbiente
ambienteAcceso
EnNivel
+ alto: float
+ ancho: float
+ aptoDiscapacitado: boolean
EntreNivel
Ascensor
+ cantAccesos: int
+ capacidadMaxima: int
+ electrico: boolean
+ pesoMaximo: float
+ nivelesAlcanzados() : Collection(int)
AmbienteCompuesto
AmbienteSimple
2
ambienteSensor
1..*
Pasillo
Sensor
+ descripcion: string
+ nombre: string
Arcada
Puerta
+ modoApertura: enum
Rampa
+ longitud: int
+ pendiente: float
Escalera
+ cantEscalones: int
+ mecanica: boolean
+ pendiente: float
Posicionamiento
+ posicion: point

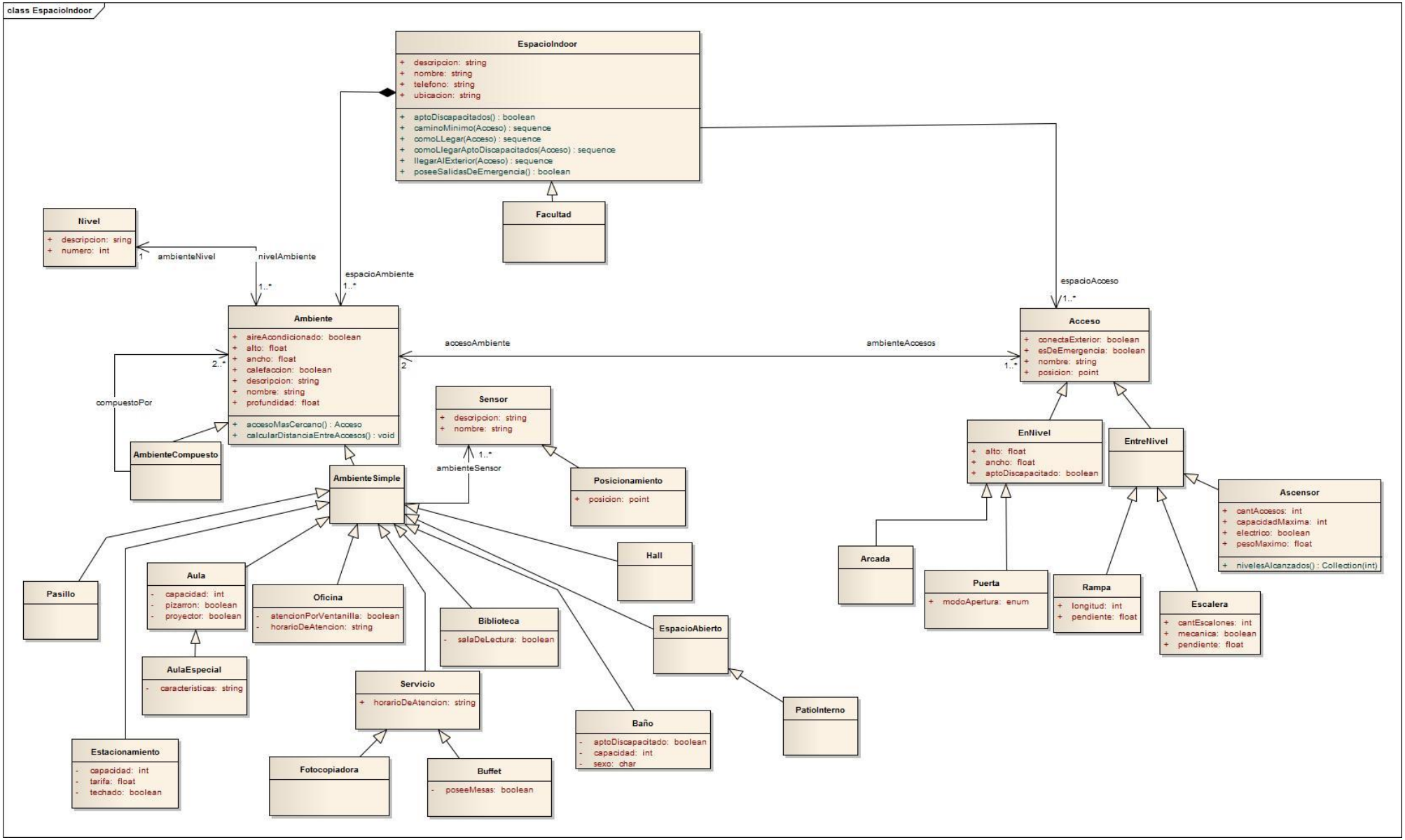
Finalmente, definimos un conjunto de restricciones escritas en lenguaje OCL con el objetivo de enriquecer el metamodelo generado.

Con el objetivo de demostrar la validez del metamodelo "*Espacio Indoor"*, decidimos realizar una extensión del mismo que logre representar las Facultades de la Universidad Nacional de La Plata. Para ello necesitamos realizar el mismo analizamos que utilizamos para crear el metamodelo de espacio indoor:

1. Obtuvimos los conceptos comunes a todas las Facultades:
    - Aula - Aula Especial.
    - Oficina.
    - Biblioteca.
    - Baño.
    - Servicio – Buffet, Fotocopiadora.
    - Estacionamiento.
2. Realizamos una extensión del metamodelo *"Espacio Indoor"* creado utilizando el método conservativo, el cual permite definir nuevas clases únicamente a través de la herencia de las clases del metamodelo, a las cuales se les pueden agregar nuevos atributos, métodos, relaciones entre las otras clases del metamodelo, y restricciones OCL para especificar semántica adicional
3. Obtuvimos el Metamodelo *"Facultad"* (ver en la siguiente hoja)
4. Definimos las restricciones OCL que enriquecen el metamodelo.

En este capítulo demostramos cómo un metamodelo puede estructurarse de manera que los conceptos más abstractos puedan ser reutilizados en varios dominios concretos diferentes. Esto se logró mediante la definición de un metamodelo abstracto "Espacios Indoor" que luego fue refinado mediante un metamodelo más concreto "Facultad", el cual reutiliza todos los conceptos comunes. Para este fin se analizaron las formas de relacionar metamodelos, en particular el método de herencia conservativa vs. el método de extensión libre, seleccionándose el primero de ellos para su aplicación en este trabajo.

Podemos concluir, que si se realiza un correcto análisis de lo que queremos representar, es posible construir una arquitectura de componentes genéricos mediante la definición de un metamodelo que contiene nuevas metaclases (los conceptos comunes) y metaasociaciones (las relaciones entre ellos). Si además a dicho metamodelo se le agregan restricciones OCL se logra un metamodelo enriquecido y robusto que cumple con los objetivos planteados.


class EspacioIndoor
EspacioIndoor
+ descripcion: string
+ nombre: string
+ telefono: string
+ ubicacion: string
+ aptoDiscapacitados() : boolean
+ caminoMinimo(Acceso) : sequence
+ comoLLegar(Acceso) : sequence
+ comoLlegarAptoDiscapacitados(Acceso) : sequence
+ llegarAlExterior(Acceso) : sequence
+ poseeSalidasDeEmergencia() : boolean
Facultad
Nivel
+ descripcion: sring
+ numero: int
1
ambienteNivel
nivelAmbiente
1..*
espacioAmbiente
1..*
espacioAcceso
1..*
Ambiente
+ aireAcondicionado: boolean
+ alto: float
+ ancho: float
+ calefaccion: boolean
+ descripcion: string
+ nombre: string
+ profundidad: float
+ accesoMasCercano() : Acceso
+ calcularDistanciaEntreAccesos() : void
2..*
compuestoPor
AmbienteCompuesto
accesoAmbiente
2
ambienteAccesos
1..*
Acceso
+ conectaExterior: boolean
+ esDeEmergencia: boolean
+ nombre: string
+ posicion: point
Sensor
+ descripcion: string
+ nombre: string
1..*
ambienteSensor
AmbienteSimple
Posicionamiento
+ posicion: point
EnNivel
+ alto: float
+ ancho: float
+ aptoDiscapacitado: boolean
EntreNivel
Ascensor
+ cantAccesos: int
+ capacidadMaxima: int
+ electrico: boolean
+ pesoMaximo: float
+ nivelesAlcanzados() : Collection(int)
Arcada
Puerta
+ modoApertura: enum
Rampa
+ longitud: int
+ pendiente: float
Escalera
+ cantEscalones: int
+ mecanica: boolean
+ pendiente: float
Pasillo
Aula
- capacidad: int
- pizarron: boolean
- proyector: boolean
Oficina
- atencionPorVentanilla: boolean
- horarioDeAtencion: string
Hall
Biblioteca
- salaDeLectura: boolean
EspacioAbierto
AulaEspecial
- caracteristicas: string
Servicio
+ horarioDeAtencion: string
Baño
- aptoDiscapacitado: boolean
- capacidad: int
- sexo: char
PatioInterno
Estacionamiento
- capacidad: int
- tarifa: float
- techado: boolean
Fotocopiadora
Buffet
- poseeMesas: boolean

Capítulo 5

Herramientas de Metamodelado y ADOxx

Capítulo 5 – Herramientas de Metamodelado y ADOxx

5.1 Introducción

La construcción de los modelos para el desarrollo de software se realiza usualmente con herramientas CASE. En estas herramientas los formalismos de cada modelo ya se encuentran plenamente definidos, lo que implica que no es posible agregarles nuevas restricciones ni comportamiento. Las herramientas de metamodelado surgieron como una manera de dar solución a este problema, ya que poseen formalismos propios que permiten la expresión de diferentes modelos, incluyendo sus restricciones. En este capítulo se presentan características y ventajas del uso de dichas herramientas.

5.2 Herramientas de metamodelado

5.2.1 Descripción

Una de las principales aplicaciones del meta-metamodelado es la generación de herramientas de modelado para un formalismo dado a partir de la descripción de su sintaxis mediante un metamodelo. Como se vio en el capítulo 3, el metamodelado es el mecanismo que permite definir rápida y fácilmente entornos visuales para la manipulación de lenguajes visuales que se ajustan tanto a dominios específicos como a las necesidades del desarrollador.

Las herramientas de metamodelado han surgido como alternativas a las herramientas CASE convencionales, con el fin de permitir a sus usuarios la creación de nuevos formalismos para diagramas que no se encuentren disponibles en la herramienta o para la complementación de las diferentes restricciones y reglas de consistencia internas de los diagramas que se pueden elaborar en la herramienta. Estas herramientas proveen formalismos para la especificación de los diferentes tipos de diagrama que van a ser elaborados con dicha herramienta [26, 27].

Con una herramienta de metamodelado es posible describir de manera completa un tipo de diagrama cualquiera de interés para un problema específico. Una vez que el tipo de diagrama ha sido descrito, la herramienta se puede usar para la elaboración de instancias del mismo.

Las herramientas de metamodelado se basan en una arquitectura de tres capas que se corresponden con los niveles de metamodelado M3 – M2 – M1 propuestos por la OMG (ver capítulo 3, sección 3.4.3):

- **M3**

  La capa más alta incluye el lenguaje utilizado para definir los metamodelos, el cual suele ser gráfico y está normalmente fijo.

- **M2**

  La capa intermedia contiene el metamodelo del formalismo, que es especificado por el usuario de la herramienta.

- **M1**

  En la capa inferior están las herramientas de modelado generadas automáticamente a partir de los metamodelos, las cuales permiten la manipulación de modelos que son instancias del formalismo descrito.

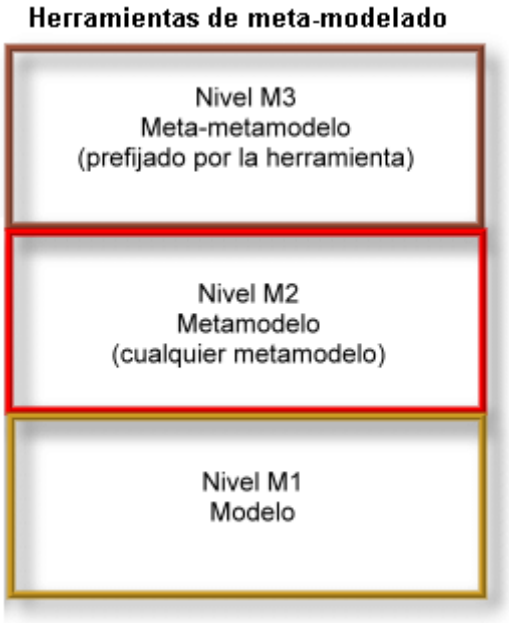


Figura 5.1 - *Herramientas de modelado en la Arquitectura 4 capas.*

Las herramientas de metamodelado permiten generar otras herramientas de modelado para un formalismo descrito, a partir de la información de los metamodelos. Esto permite construir entornos de modelado gráfico con gran rapidez ya que los formalismos que se utilizan para el metamodelado suelen ser gráficos.

Uno de los objetivos principales del uso de estas herramientas es facilitar al usuario la tarea de definir nuevos lenguajes de modelado, permitiendo definir la sintaxis del lenguaje, tanto la abstracta como la concreta, para facilitar la creación de modelos.

Las herramientas de metamodelado actuales fueron desarrolladas con el objetivo de facilitar el dibujo de los modelos especificados, pero poco se ha elaborado con respecto a las facilidades requeridas para la definición de las restricciones que deben verificarse en los diagramas de los tipos definidos.

5.2.2 Características

Las herramientas de metamodelado han surgido como evolución de las herramientas CASE, con el objetivo de eliminar las limitaciones y restricciones que dichas tecnologías brindan al desarrollador.

En esta sección se detallarán las características y funcionalidades esperadas que deben proveer las herramientas de metamodelado:

- Diferentes lenguajes posibilitan la adición de restricciones a los paradigmas de modelamiento; es posible lograr que tales mecanismos para el manejo de restricciones se puedan emplear para lo siguiente:
    - **Consistencia**: Reglas y relaciones adicionales que deben verificarse entre las partes de un modelo.
    - **Refinamiento**: Reglas de transformación que permiten ampliar los modelos en una fase.
    - **Simplificación**: Reglas que permiten derivar modelos reducidos a partir de uno o varios existentes para analizar aspectos particulares del modelado.
    - **Visualización**: Formas de visualizar los modelos y sus partes.
- No poseen un paradigma de modelamiento definido, es el desarrollador quien debe especificarlo por medio del metamodelo. La herramienta debe permitir especificar la sintaxis abstracta del lenguaje, es decir que debe permitir definir su metamodelo indicando cuáles son los elementos del lenguaje y sus relaciones, así como también las propiedades para cada elemento.
- La herramienta debe permitir especificar restricciones y reglas básicas que se deben verificar para que las instancias del nuevo lenguaje estén correctamente formadas. Un ejemplo de estas restricciones puede ser definir qué objetos se pueden conectar a través de cuál relación.
- Debe permitirse que en su definición sea posible especificar símbolos gráficos para los elementos, de manera gráfica, declarativa o en código.
- Debe ofrecer la funcionalidad básica esperada de cualquier herramienta de edición. Estas funciones incluyen almacenar y recuperar un modelo del disco, deshacer y rehacer, cortar, copiar, pegar y borrar. También deberían permitir manipular directamente los elementos a editar, imprimir y exportar.
- Debe ser posible el intercambio de metamodelos y modelos, es decir la importación y exportación de modelos y metamodelos para el intercambio de información entre otras instancias de la herramienta y de otras herramientas.

Lo más importante es que las herramientas de metamodelado reducen el trabajo requerido para desarrollar una herramienta que soporte un nuevo lenguaje de

modelado. Simplemente hay que definir la sintaxis del lenguaje y a partir de ella, la herramienta generará automáticamente el editor para crear instancias de ese lenguaje.

5.2.3 Ventajas de su uso

La utilización de herramientas de metamodelado trae consigo varias ventajas. A continuación se detallarán algunas de ellas [28]:

- **Facilita el versionado**
  Se puede iniciar la construcción de formalismos de modelos desde cero, lo cual implica la posibilidad de correcciones en modelos que cambian constantemente de versión o en otros que presentan modificaciones sutiles frecuentemente.

- **Facilita la comprensión**
  Facilidad gráfica de expresión del formalismo de los diferentes modelos, lo que reduce la complejidad en la comprensión de los mismos.

- **Permite definir restricciones**
  Posibilidad de creación de instancias de los modelos y de verificación de las restricciones planteadas en el metamodelo sobre esas instancias en particular.

- **Reducción en tiempo y recursos para el mantenimiento de las aplicaciones existentes.**
  La aplicación del metamodelado, implica que los pasos para realizar modificaciones en el modelo sean menores. Este cambio en el paradigma del mantenimiento de las aplicaciones genera sustanciales beneficios a la organización que toma la decisión de adoptar esta tecnología para la construcción y mantenimiento de sus sistemas de información.

- **Evita la introducción de errores en los programas.**
  La capacidad de introducir una nueva funcionalidad en un sistema de información sin escribir líneas de código adicionales, elimina la posibilidad de introducir errores de programación.

- **Generación de consultas a la medida.**
  El sistema generador de consultas para un metamodelo, se convierte en una herramienta muy poderosa en manos de las personas que dominan el contexto temático de la información registrada en este metamodelo, ya que puede consultar, ordenar, agrupar, graficar o producir información alfanumérica para la información registrada.

5.2.4 Herramientas analizadas

Para obtener un mayor conocimiento acerca de las herramientas de metamodelo, hemos realizado un pequeño estudio acerca de las más populares actualmente, además de la seleccionada para realizar la implementación del metamodelo propuesto en esta tesina. A continuación se listan las características generales de cada una de ellas:

- **AToM3**
  AToM^3 (A Tool for Multi-Formalism Modelling and Meta-Modelling) es una herramienta de metamodelado escrita en el lenguaje de programación Python. Posee un meta-metamodelo basado en el modelo entidad-relación, que permite la definición de los diferentes metamodelos en un entorno gráfico con las mismas características que emplea el usuario en la construcción de los diferentes modelos. Los elementos básicos para definir los paradigmas son las "entidades" que hacen parte del modelo y sus posibles interconexiones o "relaciones". Estos elementos pueden contar adicionalmente con imágenes asociadas a ellos para la construcción de los modelos.
  Esta herramienta brinda la posibilidad de definir restricciones en términos de gramáticas de grafos incorporadas a su entorno. Las gramáticas de grafos tienen similitudes con las gramáticas basadas en texto ya que pueden ser usadas para describir las transformaciones a un grafo determinado.

  Las gramáticas de grafos se definen como un conjunto de reglas que poseen un lado izquierdo (LHS) que contiene las precondiciones que deben ser cumplidas para activar una determinada regla y un lado derecho (RHS) que contiene el grafo que remplazará el que equivale al lado izquierdo de la regla.

- **Dome**
  DOME (Domain Modeling Environment) es una herramienta MetaCASE escrita en Smalltalk que utiliza un lenguaje gráfico para la definición de los diferentes elementos del paradigma de modelamiento. Los elementos básicos son diferentes tipos de clases y conexiones, además de otros elementos de control.

  Sus especificaciones son orientadas a objetos y pueden ser interpretadas on-the-fly, es decir, se realizan los cambios directamente en el metamodelo y se pueden usar inmediatamente en la zona de edición de instancias.

- **MetaEdit+**
  Como toda herramienta de metamodelado, MetaEdit+ permite al desarrollador definir los metamodelos que definen la estructura y el comportamiento de cualquier diagrama y después utilizar estos metamodelos para generar un entorno de trabajo para estos diagramas. Además permite elaborar informes asociados a los diagramas tanto para consultar sus características, verificar consistencia o listar errores y generar código en algún determinado lenguaje de programación.

Para la creación de estos metamodelos, los elementos que MetaEdit+ proporciona son:

- **Properties**: Elementos que permiten caracterizar los objetos que se representan en un diagrama.
- **Objects**: Entidades que aparecerán representando conceptos en los diagramas.
- **Relationships**: Relaciones existentes entre los distintos objetos.
- **Roles**: Papel que interpreta una Entidad dentro de una relación con otras.
- **Ports**: Puntos de conexión que se establecen en la representación gráfica de las entidades que sirven para definir dónde se puede establecer una relación con otra entidad.
- **Graphs**: Diagramas que podrán incorporar todos los elementos que se hayan definido anteriormente.

Además, es posible la generación de informes y código gracias a MERL, un lenguaje de script propio de MetaEdit+ que nos permite navegar por todos los componentes de un diagrama.

- **Eclipse Modeling Framework (EMF)**
  EMF es un framework que facilita el modelado y la generación de código para la construcción de herramientas y otras aplicaciones basadas en una estructura de modelos de datos.

  A partir de una especificación del modelo descripto en un archivo con extensión XML, EMF proporciona:

  - herramientas y soporte en tiempo de ejecución para producir un conjunto de clases Java que represente el modelo diseñado;
  - una serie de clases adaptables que permiten la visualización y la edición de comandos basados en el modelo;
  - Un editor básico para manipular las clases creadas.

  EMF permite usar un modelo como el punto de partida para la generación de código, e iterativamente refinar el modelo y regenerar el código, hasta obtener el código requerido. Aunque también prevé la posibilidad de que el programador necesite modificar ese código, es decir, se contempla la posibilidad de que el programador edite las clases generadas, para agregar o editar métodos y variables de instancia. Siempre se puede regenerar desde el modelo cuando se necesite, y las partes agregadas serán preservadas durante la regeneración [29].

EMF (core) es un estándar común para los modelos de datos, muchas tecnologías y frameworks están basado en él. Esto incluye servidores, frameworks de persistencia, frameworks de UI y soporte para transformaciones.

EMF está compuesto por tres piezas fundamentales:

- EMF - El framework EMF(core) incluye un metamodelo llamado Ecore, que permite describir modelos y un soporte en línea para ellos, que incluye notificaciones de cambios, soporte de persistencia con serialización en XML por defecto, y una API para manipular objetos EMF genéricamente.
- EMF.Edit - El framework EMF.Edit proporciona clases genéricas reutilizables que permiten construir los editores para los modelos EMF. Provee:
    - Clases etiquetadas con contenido, soporte para las propiedades del código y otras clases convenientes que permiten que los modelos EMF se visualicen me forma estándar.
    - Un framework de comandos, que incluye un conjunto de clases genéricas de implementación de comandos que permiten la construcción de editores que logran automatizar las acciones de deshacer y rehacer.
- EMF.Codegen - La generación de código EMF es capaz de generar todo lo necesario para construir un completo editor para diseñar un modelo de EMF. Incluye una interfaz gráfica de usuario desde la que se pueden especificar las opciones de generación, y los posibles generadores que se pueden invocar.

El código generado por EMF es eficiente, correcto y fácilmente modificable. El mismo provee un mecanismo de notificación de cambios de los elementos, una implementación propia de operaciones reflexivas y persistencia de instancias del modelo.

5.3 ADOxx

ADOxx es una herramienta de desarrollo de metamodelos implementada en la Universidad de Viena, Austria; y realizada bajo el marco del "Proyecto de Colaboración de la Facultad de Informática (UNLP) con la Universidad de Viena en Austria"; el cual será expuesto en la siguiente sección.

ADOxx permite la definición de diferentes metamodelos en un entorno gráfico con las mismas características que emplea el usuario en la construcción de los diferentes modelos. De esta manera, se puede definir cualquier tipo de metamodelo en términos de las entidades que forman parte del mismo y sus posibles interconexiones

o relaciones. Una vez definido el metamodelo, se puede emplear su definición para construir los modelos pertinentes a un problema específico del mundo. [8, 30]

5.3.1 Características

La plataforma ADOxx es un entorno que se basa en el desarrollo y metamodelado para crear herramientas de modelado de dominio específico. Algunas de sus características son:

- Opennes: de acceso no propietario, y de código abierto.
- Posee componentes tanto del lado del cliente como del lado del servidor.
- Multi-thread
- De fácil extensión
- Orientado a Componentes
- Interfaces estándar de los Servicios Web
- Cliente Web
- Se puede realizar el metamodelado con un rico conjunto de conceptos
- Personalización, scripting
- Multinivel de personalización
- Basado en eventos
- Concepto vista flexible
- Repositorio compartido
- Acceso basado en roles
- Extendida a soporte multi-idioma y Unicode
- Soporte SSO: Inicio de Sesión Único

5.3.2 ADOxx GraphRep Repository

El ADOxx GraphRep Repository recoge la representación gráfica de los diferentes componentes, escenarios y proyectos que participaron en los distintos proyectos ADOxx, y los proporciona a la comunidad. Si una persona es miembro de la comunidad, puede añadir, revisar, modificar, comentar y evaluar las GRAPHREPs disponibles en el repositorio. Si se accede a la siguiente url:

http://www.adoxx.org/live/adoxx-graphrep-repository-wiki/-/wiki/GRAPHREP+Repository/FrontPage

se pueden visualizar los gráficos que han participado en los distintos modelos implementados en ADOxx. Si se hace click en cualquiera de ellos, se muestra el "GraphRep Code" que se necesita para utilizar ese gráfico en la generación de un modelo ADOxx.

5.3.3 ADOxx Development Toolkit

ADOxx Development Toolkit permite diseñar el metamodelo que luego será instanciado con la herramienta ADOxx Modelling Toolkit [31].

A través de esta herramienta se pueden generar las clases junto con las relaciones que componen el metamodelo, y las restricciones que de imponen sobre el mismo.

Dentro de esta herramienta se pueden manipular principalmente los siguientes elementos:

- **Usuarios**
    - Crear usuarios nuevos.
    - Editar usuarios.
    - Asignarle a cada usuario la librería que contiene el metamodelo que se quiere instanciar.
- **Librerías**: una librería en ADOxx se corresponde con un metamodelo, es decir cuando queremos generar un nuevo metamodelo, lo que tenemos que hacer es crear una nueva librería ADOxx.

  Cada librería está compuesta por las clases y las relaciones que conforman el metamodelo. A cada clase o relación se le puede asignar una representación gráfica, la cual puede ser tanto una imagen, como un gráfico que se encuentre en el *ADOxx GraphRep Repository* o bien, se puede crear una nueva imagen en el editor que *ADOxx Development Toolkit* ofrece.

  Dentro de esta opción se puede:
    - Gestionar las distintas librerías que la herramienta ofrece en su instalación.
    - Crear nuevas librerías.
    - Crear nuevos elementos, como por ejemplo, clases o relaciones dentro de las librerías.
    - Agregar atributos a las clases o relaciones junto con su tipo de dato.
    - Inspeccionar y modificar los elementos que componen cada librería.
    - Aplicar restricción a los componentes de cada librería, tanto a las clases, como a las relaciones y a los atributos que las conforman.

Además, se puede realizar otro tipo de acciones que no son relevantes para los objetivos de nuestra tesina. Por ejemplo, cambiar la configuración de los componentes ADOxx, manipular, entre otras cosas.

5.3.4 ADOxx Modelling Toolkit

ADOxx Modelling Toolkit permite instanciar los metamodelos diseñados en ADOxx Development Toolkit. Para acceder es necesario ingresar un nombre de usuario y una password que deben coincidir con el usuario que tenga asignada la librería que se quiere utilizar.

Una vez dentro de la herramienta, se puede crear un modelo con todos los elementos del metamodelo generado anteriormente. Cada vez que se ingrese un elemento o que se intente utilizar una relación que une dos clases o elementos, ADOxx comprobará que se utilice de forma correcta según lo declarado en ADOxx Development Toolkit. Si ADOxx comprueba que existe algún caso en que el modelo que se está generando no es coherente con lo declarado en la herramienta, mostrará en la pantalla mensajes de error que advierten de esta situación.

Al hacer doble click en algún elemento en particular (ya sea una clase o una relación), ADOxx permite cargar información a los atributos que tenga declarados ese elemento, siempre y cuando cumpla con las restricciones declaradas en ADOxx Development Toolkit.

Por último, cuando el modelo ya fue generado correctamente y contiene toda la información que el usuario desea agregar, ADOxx permite exportarlo a un archivo XML. Este archivo puede ser utilizado para los distintos objetivos que el desarrollador necesite.

5.3.5 Proyecto de Colaboración

Para el desarrollo de esta tesina se decidió utilizar la herramienta de desarrollo de Metamodelado ADOxx con el objetivo de participar en el Proyecto de Colaboración existente entre la Facultad de Informática (UNLP) y la Universidad de Viena en Austria, denominado "ADOxx Metamodel Compiler" que tiene por objetivo contribuir con la iniciativa internacional OMI.

OMI en una iniciativa que tiene como objetivo concreto crear una comunidad que se ocupe de la creación, mantenimiento, modificación, distribución y análisis de modelos. Todo aquello que se considere que es un modelo útil para algún propósito por cualquier grupo de personas será básicamente un contenido potencial para la OMI [7].

Durante el transcurso del Proyecto de Colaboración, los integrantes de ambas universidades intercambiamos conocimientos adquiridos sobre la herramienta, a través de distintos tipos de comunicaciones, desde e-mails hasta viajes de intercambio al otro país.

En el “Formulario para la presentación de Proyectos Conjuntos de Investigación en el marco de Programas de Cooperación Bilateral”, dentro del Ministerio de Ciencia, Tecnología e Innovación Productiva Dirección Nacional de Relaciones Internacionales [32] se pueden observar los detalles del proyecto, junto con sus objetivos e integrantes que colaboran en ambos países.

5.4. Resumen

Como se ha visto en el capítulo 3 y complementado con lo estudiado en estas secciones, podemos concluir que los metamodelos son útiles para crear los elementos básicos que permiten modelar una realidad, mediante el uso de diagramas.

Para la creación de metamodelos, se suelen utilizar herramientas de metamodelado en las cuales se pueden definir los elementos de un metamodelo. Para esa definición, las herramientas se valen de dos tipos de especificaciones: una *gráfica*, que permite la interacción con el analista, y una *declarativa*, que define la lógica con la cual se implementa el metamodelo.

Por lo general, las herramientas de metamodelado combinan los dos tipos de especificaciones, lo cual requiere que el analista domine la sintaxis de ambas. Además, no se puede separar la información de los dos tipos de especificaciones, por lo cual, si se cambia la especificación gráfica de un metamodelo, hay que cambiar su especificación declarativa.

En conclusión, las herramientas de metamodelado permiten la definición de nuestra propia técnica de modelado. Los elementos permitidos del metamodelo generado se guardan en un repositorio y pueden ser usados por otros analistas, es decir, se define un UML particular, con los elementos, restricciones y relaciones posibles.

Esta tesina se desarrolló bajo el marco del Proyecto de Colaboración denominado: “ADOxx Metamodel Compiler”, realizado entre la Universidad de Viena y La Universidad Nacional de La Plata. ADOxx permite a través de la plataforma ADOxx Development Toolkit generar todos los componentes que se quieren representar en el metamodelo a generar; y a través de la plataforma ADOxx Modelling Toolkit permite instanciar el metamodelo generado en la herramienta de desarrollo.

Capítulo 6

Implementación

Capítulo 6 – Implementación

6.1 Introducción

En las siguientes secciones se detalla cómo se utiliza la tecnología mencionada en el primer capítulo para la construcción de la herramienta final que refleja el metamodelo obtenido en el capítulo 4. Se analizan las plataformas que posee ADOxx para la implementación de metamodelos y se detalla la manera para definir las clases que formarán parte de la librería que permite implementar cualquier espacio indoor que se desee.

A su vez se expone la herramienta "Modelador de facultades" desarrollada para un caso de estudio en particular, la cual extiende de la solución general, permitiéndonos realizar un análisis más profundo de la plataforma ADOxx y sus mecanismos de extensión de metamodelos. También se muestra la utilización de "Modelador de facultades" mediante la instanciación de un modelo en particular, en este caso una sección de la Facultad de Informática de la Universidad Nacional de La Plata.

Como complemento se presenta un prototipo de una herramienta que tiene como objetivo mostrar que es posible, a partir de un modelo instanciado en "Modelador de Facultades", realizar el recorrido del espacio indoor, encontrar un camino entre dos ambientes, verificar si el espacio indoor posee salidas de emergencia, entre otras consultas.

6.2 Metamodelo "Espacios Indoor"

Como se fue explicando a lo largo de los capítulos de esta tesina, uno de los objetivos principales de la misma es obtener un metamodelo que refleje todos los conceptos y características comunes de los espacios indoor.

Luego de realizar un profundo análisis sobre diferentes espacios indoor y luego de transitar por diferentes aproximaciones del metamodelo (ver capítulo 4), obtuvimos el diagrama de clases mostrado en la imagen 6.1 que refleja las clases junto con su comportamiento y relaciones entre ellas, que permiten definir cualquier espacio que posea las características de un espacio indoor (ver capítulo 2).

A su vez para enriquecer el metamodelo obtenido, definimos una serie de restricciones escritas en el lenguaje OCL, las cuales deben tenerse en cuenta al momento de definir el modelo de un espacio indoor cualquiera. Ellas son:

- Para todas las clases el atributo nombre debe ser obligatorio.
- Un EspacioIndoor debe tener al menos un Acceso cuyo atributo conectaExterior este seteado con el valor true.
- Cada Ambiente debe tener al menos un Acceso.
- Cada Ambiente debe tener al menos un SensorDePosicionamiento.

- Un Acceso de tipo EntreNivel debe conectar dos ambientes de distinto Nivel.
- Un Acceso de tipo EntreNivel debe conectar dos ambientes con Niveles sucesivos.
- Un Acceso de tipo EnNivel debe conectar dos ambientes que se encuentren en el mismo Nivel.
- Cada Acceso debe conectar dos Ambientes distintos.
- El atributo ancho de un TipoDeAcceso EnNivel debe ser mayor o igual a 0.80 mts.
- El atributo alto de un TipoDeAcceso EnNivel debe ser mayor o igual a 2 mts.
- El atributo cantEscalones de un TipoDeAcceso Escalera debe ser mayor a 0.
- El atributo cantAccesos de un TipoDeAcceso Ascensor debe ser mayor a 0.
- El atributo capacidadMaxima de un TipoDeAcceso Ascensor debe ser mayor a 0.
- En un mismo nivel no pueden existir Ambientes con el mismo nombre.
- El TipoDeAcceso Ascensor no puede tener seteado el atributo esDeEmergencia en true.
- El TipoDeAcceso Escalera no puede tener seteado el atributo esDeEmergencia en true si su atributo mecánica esta seteado en true.
- El atributo ubicación de EspacioIndoor debe ser obligatorio
- El atributo alto de Ambiente debe ser mayor que 0
- El atributo ancho de Ambiente debe ser mayor que 0
- El atributo profundidad de Ambiente debe ser mayor que 0
- El atributo posición de SensorDePosicionamiento debe ser obligatorio
- El atributo posición de Acceso debe ser obligatorio
- El atributo longitud de Rampa debe ser mayor que 0
- El atributo pendiente de Rampa debe ser mayor que 0
- El atributo pendiente de Escalera debe ser mayor que 0

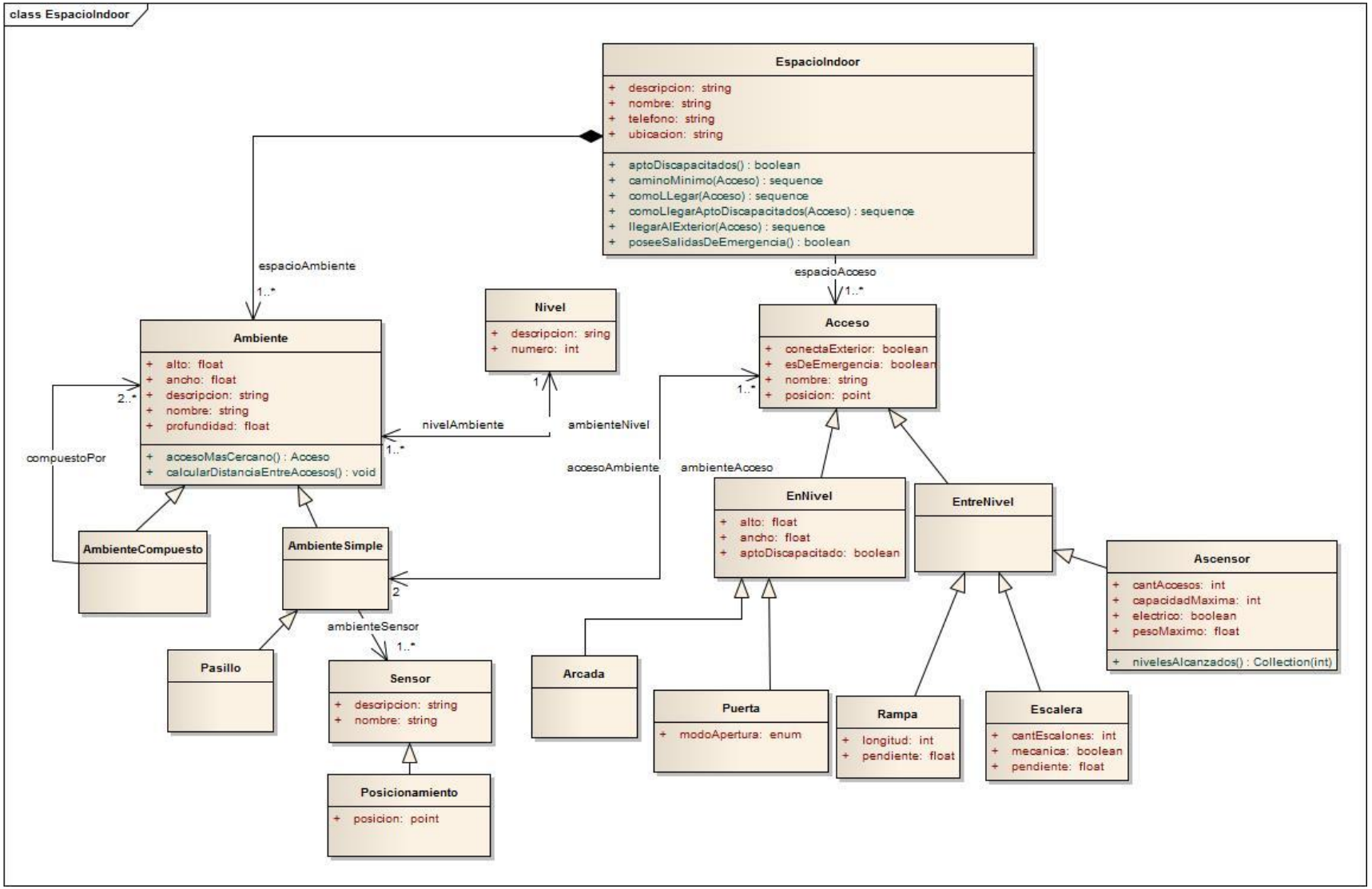


Figura 6.1 - *Metamodelo final "Espacio indoor"*

6.3 Proceso de instalación de plataforma ADOxx

ADOxx es una plataforma diseñada para realizar el desarrollo de metamodelos, la cual permite especificar la sintaxis de un lenguaje de modelado junto con su representación gráfica (ver capítulo 5).

Es necesario tener el entorno de desarrollo instalado localmente para poder importar las librerías que representan los metamodelos de *"Espacio indoor"* y *"Facultad"*. Para ello se deben seguir los siguientes pasos:

1. Ingresar a la plataforma por medio del siguiente link http://www.adoxx.org/live/download-15 y descargar el paquete haciendo click en el botón *"Download"*.

2. Enviar un mail a download@adoxx.org solicitando una licencia para uso académico de la herramienta. La misma es necesaria para poder continuar con la instalación.

3. Descomprimir en el directorio deseado el paquete ZIP descargado en el paso 1. El paquete contiene todos los archivos de instalación necesarios para configurar ADOxx en un entorno Windows.

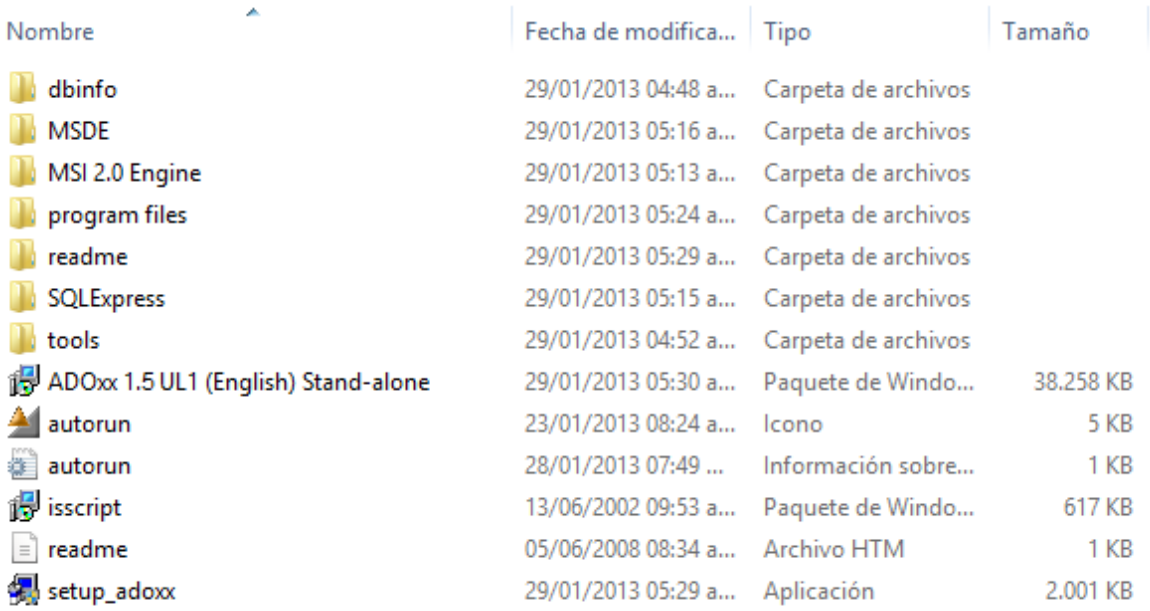


Figura 6.2 - *Contenido del paquete necesario para la instalación de ADOxx.*

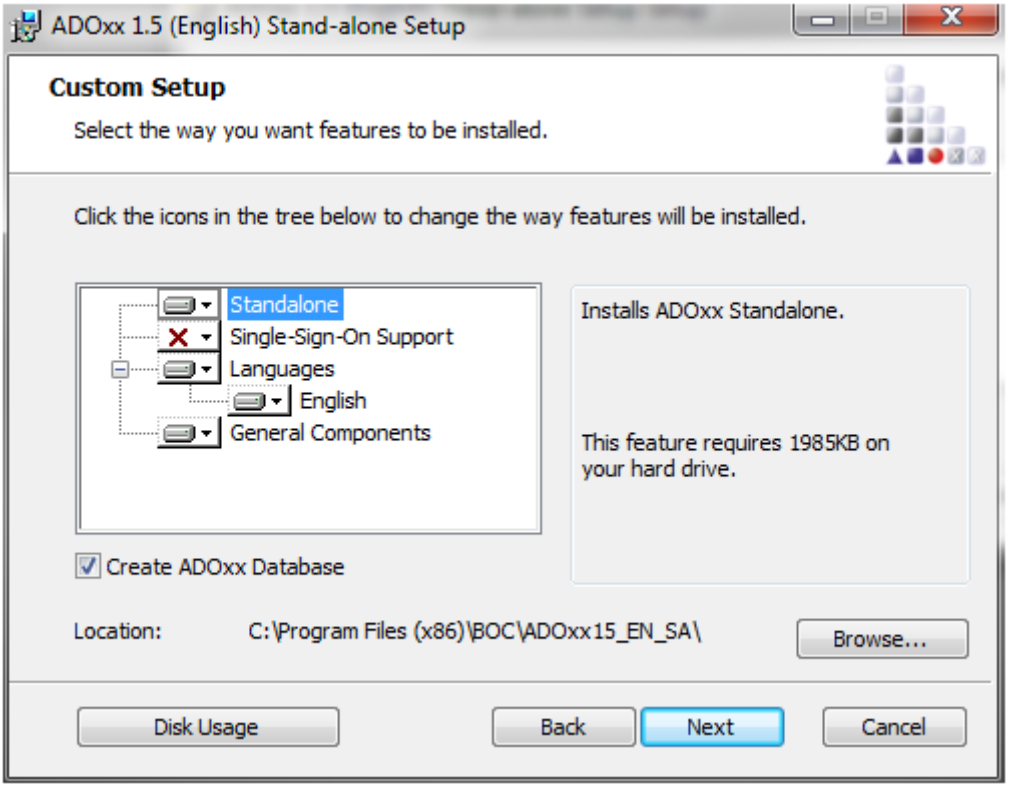
4. Debe tener instalado SQLExpress con la configuración mencionada a continuación. En caso de no poseerlo, el mismo se encuentra incluido para su instalación en el paquete descargado, bajo el directorio SQLExpress:

    - Autorización: mixta
    - Usuario: sa

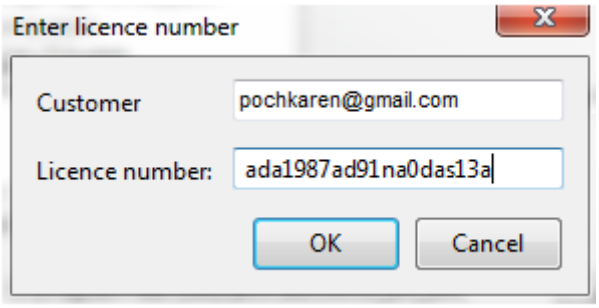
5. Comenzar con la instalación de ADOxx realizando doble click sobre el archivo *"setup_adoxx.exe"*. Se necesita tener una conexión a internet debido a que se realizan chequeos e instalaciones de paquetes que se descargan directamente del repositorio de Microsoft.

6. Luego de realizar las instalaciones adicionales requeridas, se deben seguir las instrucciones para continuar con la instalación haciendo click en el botón *"Next"*.

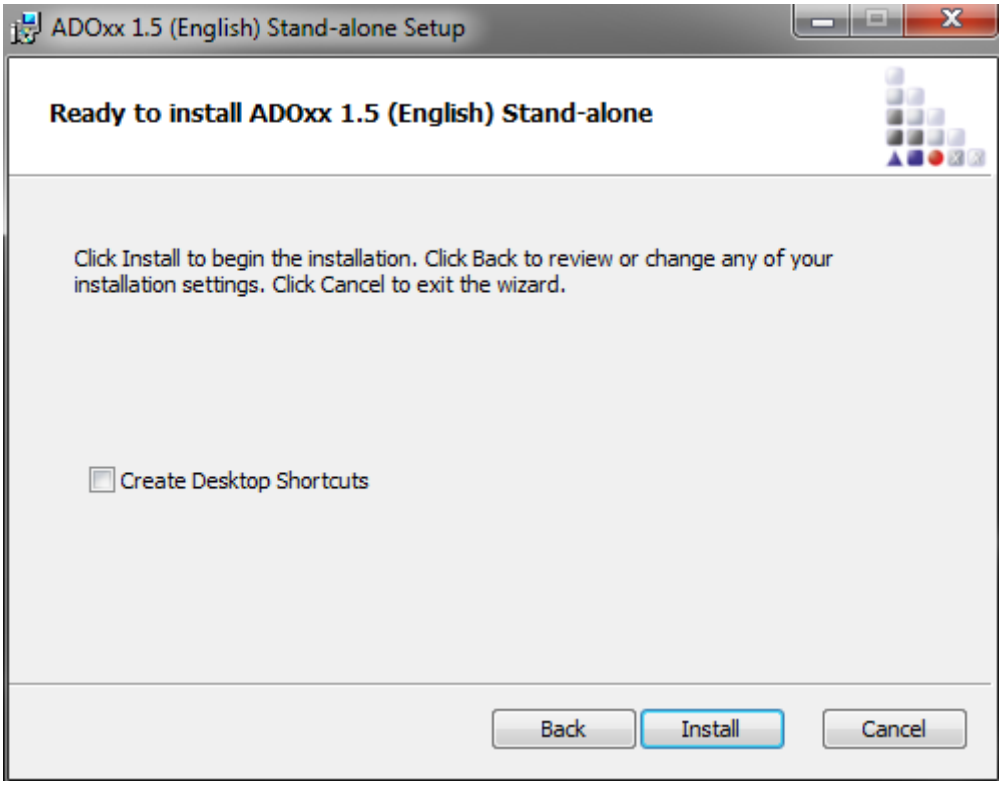
Seleccione las características que se instalarán. Debe crear una nueva base de datos y definir la ubicación de la instalación:



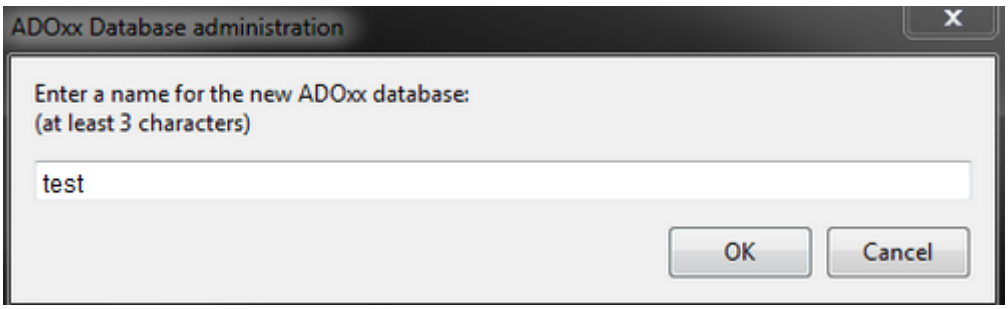
Ingrese su licencia personal, la cual fue solicitada en el paso 2:



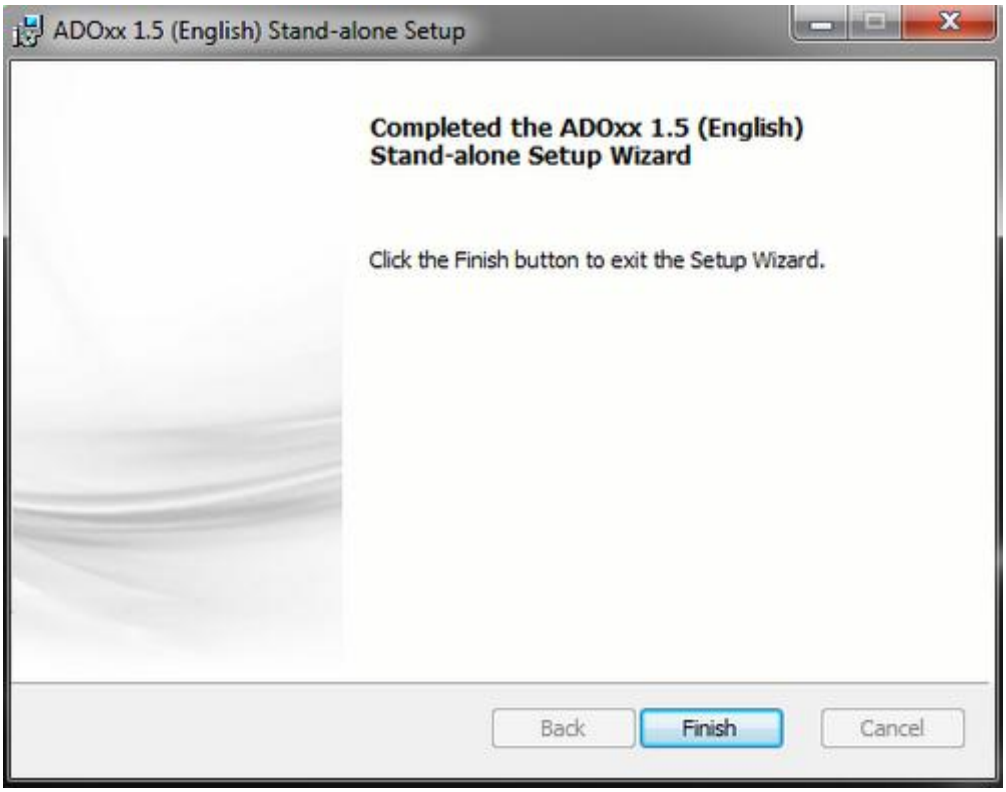
Inicie la instalación haciendo click en el botón *"Install"*



En caso que durante la instalación se encuentre una base denominada "adoxxdb", se le solicitará el ingreso del nombre de una nueva base de datos la cuál será utilizada para almacenar los metamodelos desarrollados en la misma:



Para finalizar con la instalación debe realizar click sobre el botón *"Finish"*. Luego se cerrará la ventana del wizard:



7. Una vez finalizados de manera correcta los pasos anteriores, ya se tiene instalada la plataforma ADOxx con sus entornos de desarrollo: *"ADOxx Development Toolkit"* y *"ADOxx Modelling Toolkit"* para comenzar con la implementación de los metamodelos.

6.4 Análisis del desarrollo de herramienta en ADOxx

Como se mencionó en la sección anterior la plataforma de metamodelado ADOxx posee dos entornos de desarrollo: *"Development toolkit" (Herramienta de desarrollo)* y *"Modelling toolkit" (Herramienta de modelado).*

Por un lado *"Development toolkit"* contiene las operaciones de desarrollo necesarias para definir los modelos que luego serán implementados, incluyendo todos los elementos visuales y las restricciones. Como ejemplo, en la Figura 6.3 se presenta la herramienta de desarrollo con la jerarquía de metaclases de la implementación del metamodelo de "Espacios Indoor", que muestra cómo se modelan los atributos en el atributo ***ATTRREP*** y propiedades de las metaclases. Esto abarca además los elementos visuales que se definen en el atributo ***GRAPHREP***.

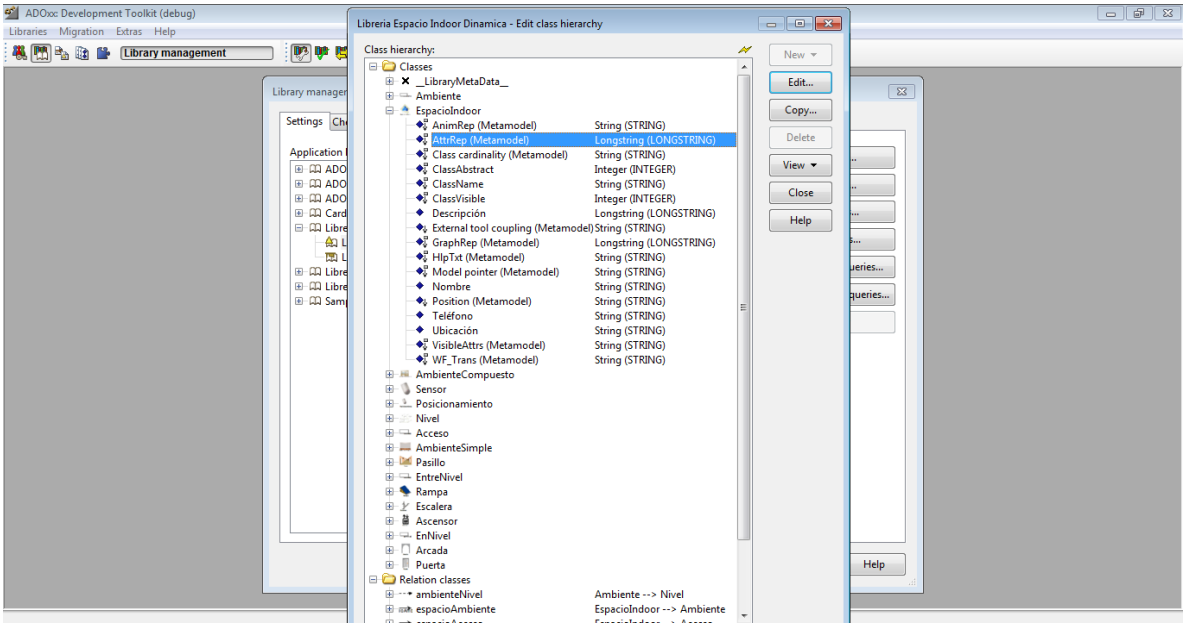


Figura 6.3 - *Jerarquía de clases de Espacio indoor implementado en ADOxx.*

Un punto importante a destacar es que ADOxx posee un conjunto de constructores predefinidos, los cuales son utilizados para la instanciación de los metamodelos propios del usuario; es decir que dichos metamodelos derivan del meta metamodelo ADOxx. La representación de metaclases, atributos y restricciones son definidas utilizando el lenguaje de programación ADOscript.

Principalmente, el conjunto de constructores predefinidos, está compuesto por clases y relaciones. Todo metamodelo debe poseer al menos una metaclase y puede contener una o más relaciones de metaclases, ambas tienen atributos que pueden ser detallados por facetas como un texto de ayuda o una expresión regular para restringir aún más los valores de los atributos. ADOxx define dos tipos de atributos de clase:

- **Notebook definition**: Son definidos en el atributo ATTRREP del meta metamodelo ADOxx, para la representación de los atributos de una metaclase determinada. Aquí se determina qué atributos son visibles para el modelador en los cuadros de diálogo de la herramienta desarrollada, y cuál es el formato en el que se presentarán. En la Figura 6.4 se muestra la definición de los atributos para la metaclase "Puerta" del metamodelo "Espacio Indoor" y su respectiva visualización en la herramienta final.

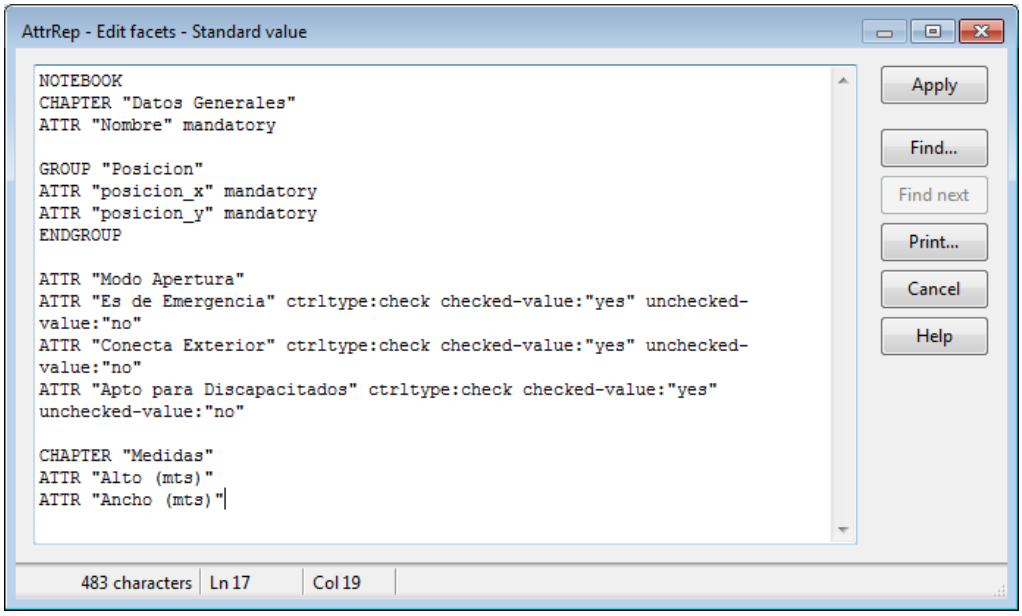


Figura 6.4.1 - *Definición en ADOscript de atributos de la metaclase "Puerta", en el atributo ATTRREP del meta metamodelo ADOxx.*

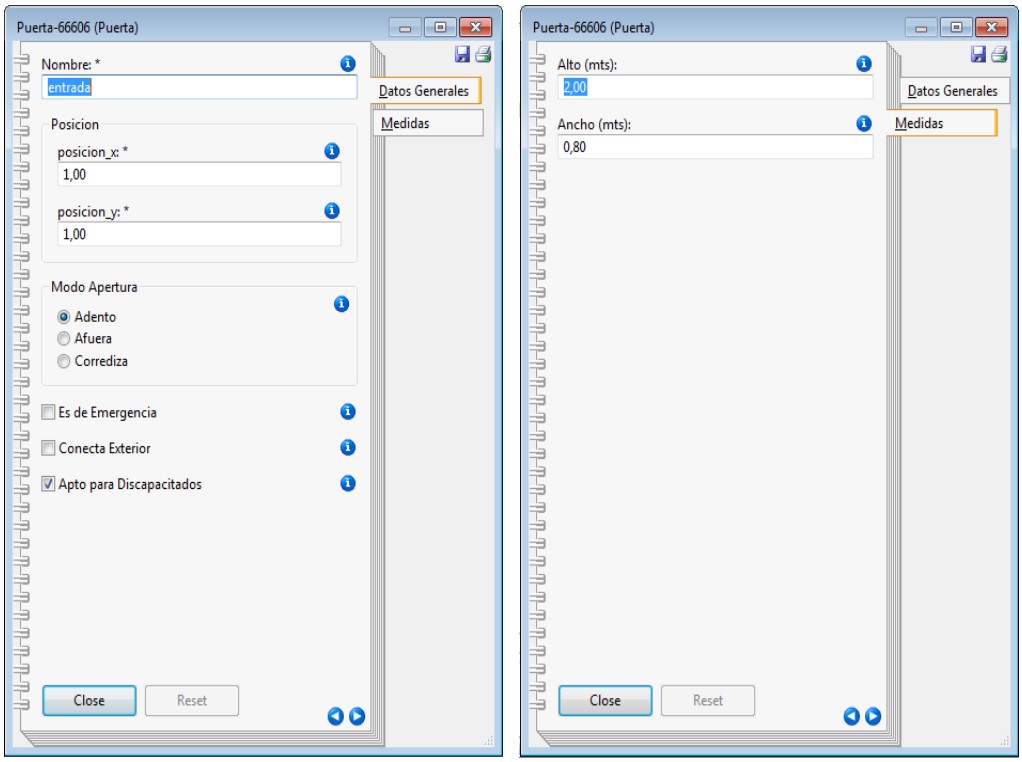


Figura 6.4.2 - *Visualización del cuadro de diálogo de atributos de la metaclase "Puerta" en la herramienta desarrollada.*

- **Graphical representations:** Son definidos en el atributo GRAPHREP del meta metamodelo ADOxx. Aquí se determina la representación visual de las metaclases y las relaciones de clases. En la figura 6.5 se muestra cómo se define la representación visual de la metaclase

"Espacio Indoor" perteneciente al metamodelo "Espacios Indoor" y su respectiva visualización.

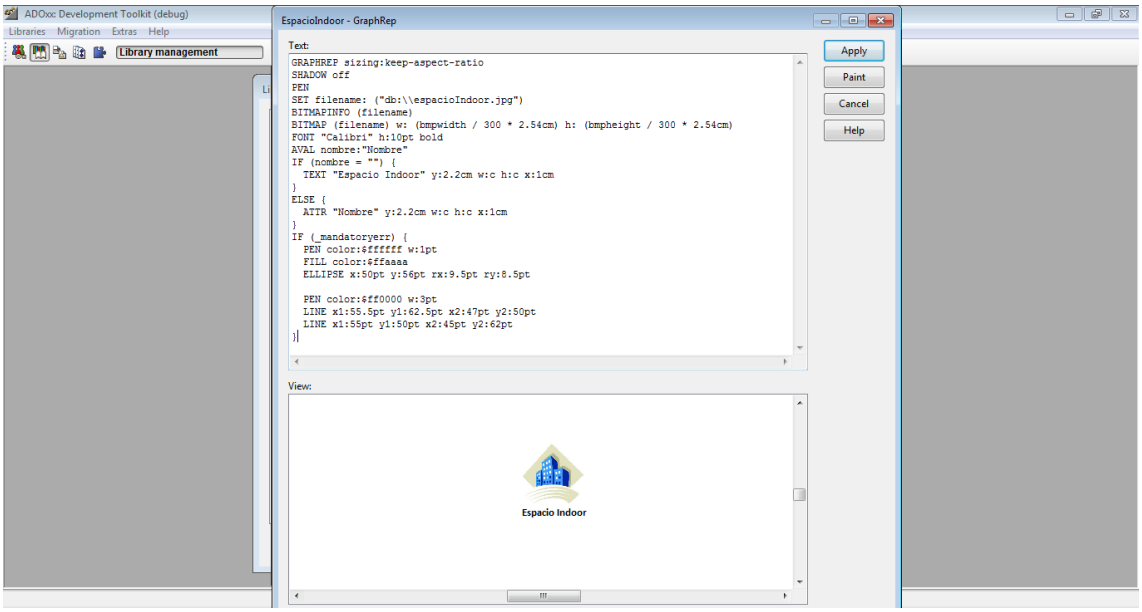


Figura 6.5 - *Definición en ADOscript de la metaclase "Espacio Indoor" en el atributo GRAPHREP del meta metamodelo ADOxx y su representación visual.*

Los atributos de instancia pueden ser de varios tipos, incluyendo tipos de datos simples como pueden ser *string*, *float* e *integer* o compuestos. Existen otros tipos de datos especiales como *record class*. Éstos últimos son colecciones de atributos de otros tipos y son representados en una tabla con filas y columnas.

Las metaclases pueden organizarse en una jerarquía definiendo una relación de subclases entre ellas. De esta manera los atributos de las superclases son heredados por las subclases. Las relaciones siempre tienen que definir exactamente una metaclase para su origen y una para su destino.

Además de los constructores predefinidos, ADOxx provee una serie de funcionalidades que facilitan la realización de métodos de modelado. Estas funcionalidades están disponibles en *"Modelling toolkit"*. Esta última herramienta comprende el entorno gráfico de modelado utilizado para generar las instancias de los modelos desarrollados en *"Development toolkit"*.

Entre las funcionalidades predefinidas se encuentran:

Modelling: Es el componente central para manipular los modelos que serán instanciados, debido a que es el responsable de la visualización de los mismos. Provee editores visuales de modelos que son generados automáticamente basándose en la definición del metamodelo en *"Development toolkit"*. Por un lado *"Model Editor"* es un editor gráfico en el cual se debe usar el mouse para dibujar los modelos, para acceder a esta vista se puede hacer click en el botón o desde el menú "*View >> Graphic"*. Por otra parte *"Tabular Model Editor"* provee un editor en forma de tabla para facilitar la

edición de los valores de los atributos de las clases y relaciones definidas en un modelo particular. Para acceder a esta vista de puede hacer click en el botón  o desde el menú *"View >> Table"*. En las Figuras 6.6.1 y 6.6.2 se presentan los editores mencionados.

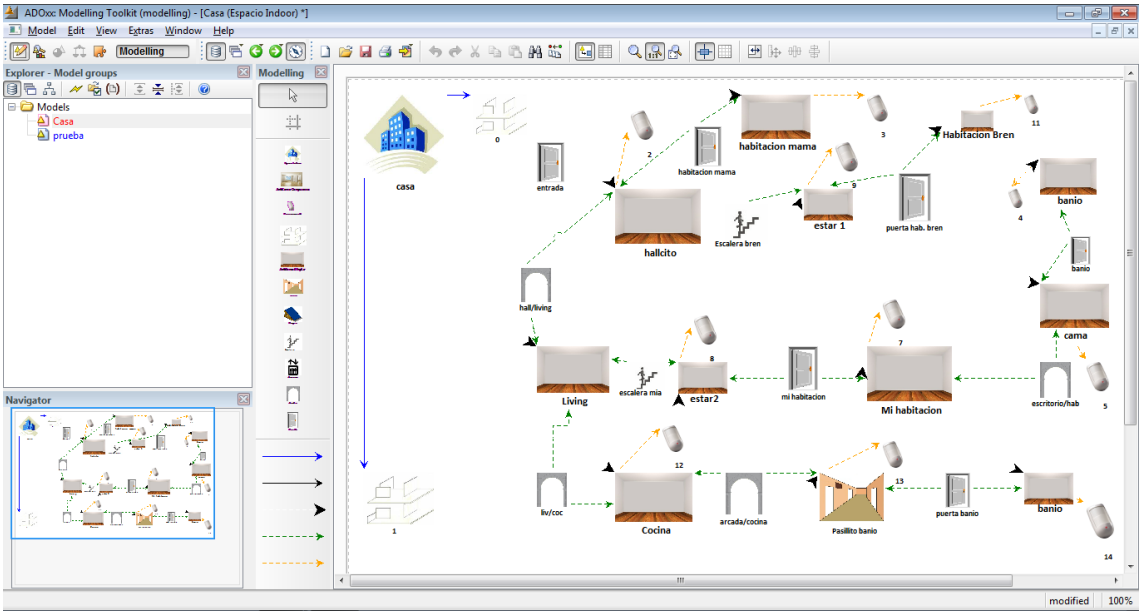


Figura 6.6.1 - *"Modelling toolkit - Model Editor".*

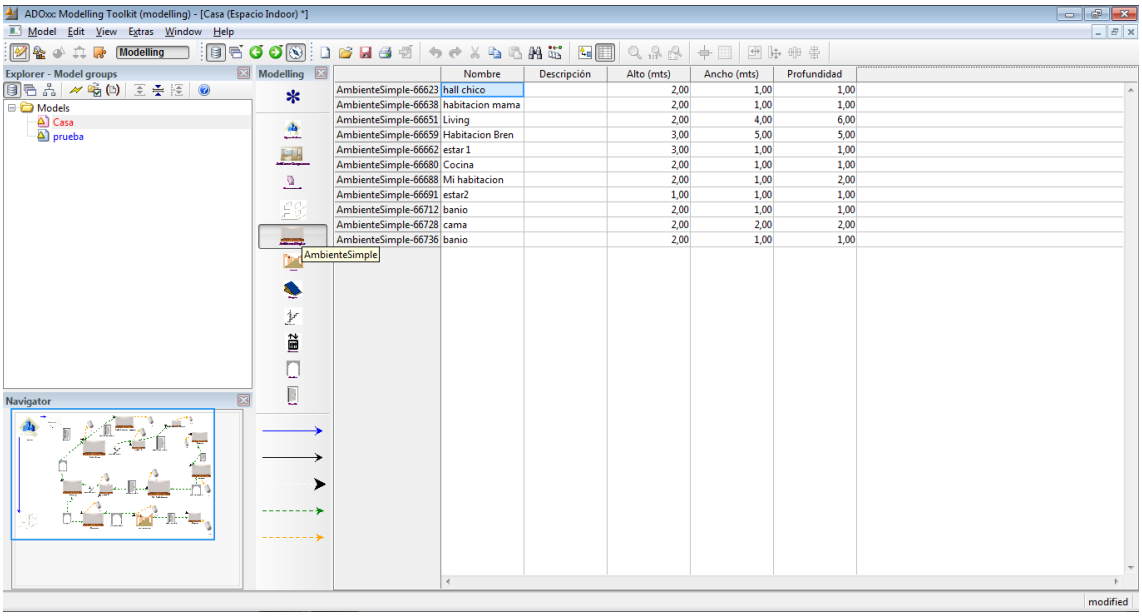


Figura 6.6.2 - *"Modelling toolkit - Tabular Model editor".*

 **Analysis:** Provee los mecanismos para formular y ejecutar consultas estáticas sobre el modelo instanciado. Las mismas están definidas en el lenguaje AQL y pueden ser definidas por el usuario o predefinidas. En el primer caso ADOxx provee un asistente para la composición de consultas, mientras

que en las predefinidas el usuario define las consultas en conjunto con una interface específica de consultas para poder acceder a las variables del modelo. La ventaja de este tipo de consultas es que el usuario que desee ejecutar una consulta predefinida sólo tiene que completar los valores deseados en los campos de texto o seleccionarlo desde un combo con valores pre cargados y no tiene que poseer ningún conocimiento acerca de la composición de las consultas AQL. Para acceder a la ventana de consultas debe hacerse click en el botón  .

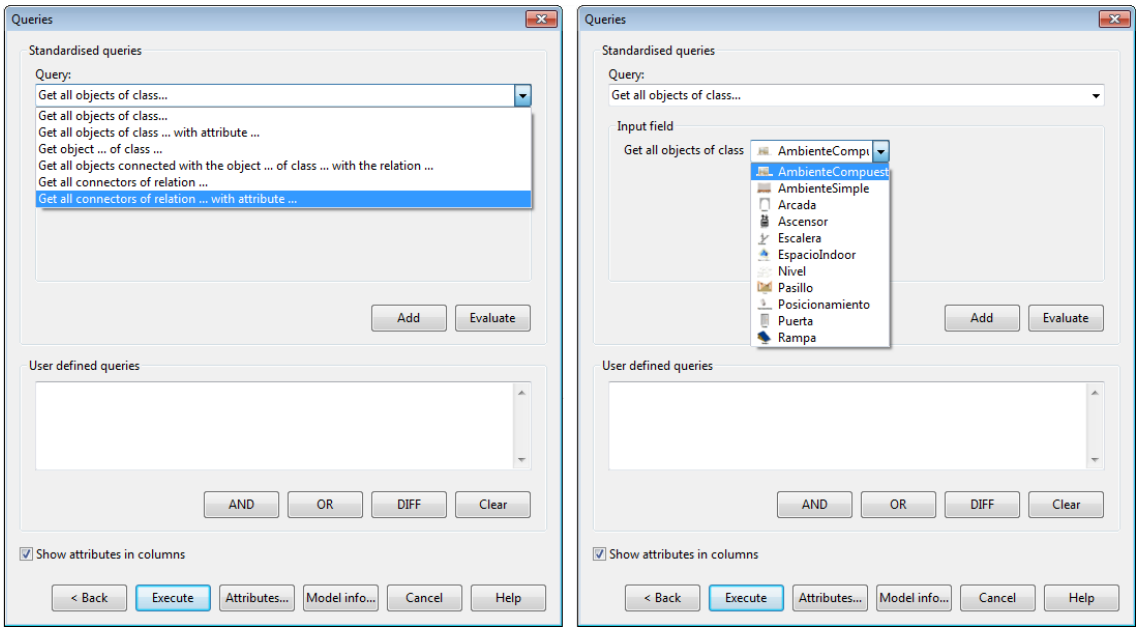


Figura 6.7 - *Interface para la generación de consultas AQL.*

 **Import/Export:** Provee la funcionalidad necesaria para exportar los modelos en un formato genérico XML. Dicho formato comprende constructores como *instances* que corresponde a instancias de clases del metamodelo, *connectors* que corresponden a instancias de relaciones de clases del metamodelo y *attributes* que corresponden a los atributos de clases y relaciones.

La definición de los metamodelos desarrollados en *"Development toolkit"* son almacenados como librerías (archivos con extensión .abl) que deben importarse dentro de cualquier entorno ADOxx para su utilización en *"Modelling toolkit"*.

6.4.1 Creación de metaclases, relaciones y atributos

Como se mencionó en la sección anterior, ADOxx provee un meta metamodelo con metaclases y constructores predefinidos desde el cual extienden los metamodelos definidos por el usuario, por lo tanto es necesario importar la librería *"Experimentation Library"* provista por ADOxx y renombrarla para comenzar con la definición del metamodelo *"Espacio Indoor"*. Para ello es necesario, dentro de *"Development toolkit >> Library Management"*, acceder a la pestaña *"Management"* y desde allí importar la librería mencionada anteriormente. Luego de realizar todos los pasos de la importación

se puede comenzar a definir las metaclases y relaciones propias del metamodelo a desarrollar.

Para crear una nueva clase o relación se debe acceder a la pestaña *"Settings"*, seleccionar la librería correspondiente y hacer click en el botón *"Class hierarchy"*. Se abrirá una ventana con dos carpetas *"Classes"* y *"Relation classes"*, en cada una de ellas se despliegan las clases y relaciones, respectivamente, definidas para el metamodelo.

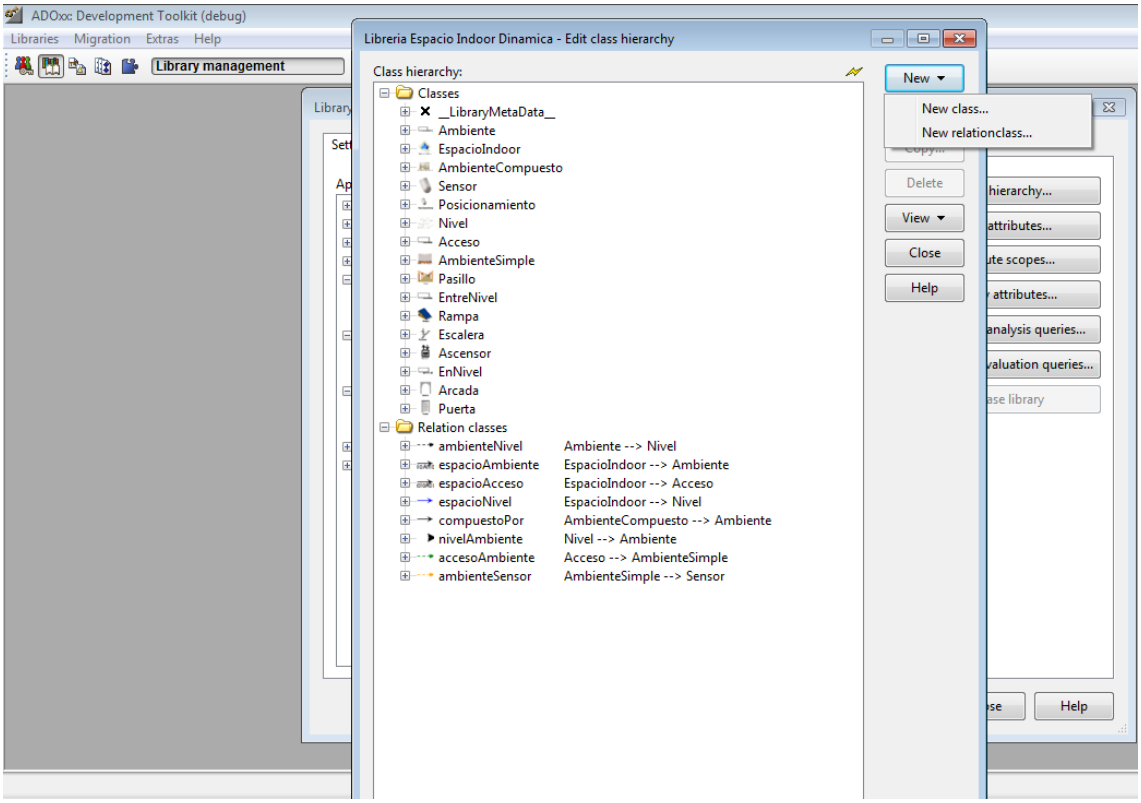


Figura 6.8 - *Creación de clases y relaciones.*

- **Nueva clase:** Para crear una nueva clase se debe seleccionar, de las clases disponibles dentro de la carpeta *"Classes"*, la superclase de la cual debe heredar la clase que se desee crear. En el caso que se quiera heredar del meta metamodelo ADOxx se debe seleccionar la clase *"_LibraryMetaData_"*, en caso contrario se debe seleccionar alguna clase creada previamente por el usuario; una vez seleccionada la superclase debe hacerse click en el botón *"New"* y se debe seleccionar del menú que se despliega la opción *"New class…"*. A continuación se mostrará la ventana expuesta en la figura 6.9, en la que se debe escribir el nombre de la nueva clase. Una vez completada la edición se debe hacer click en el botón *"OK"* para finalizar con la creación de la clase.

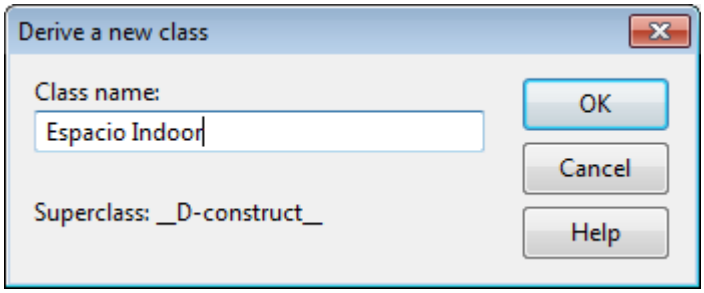


Figura 6.9 - *Configuración nueva clase.*

Una vez creada la clase se debe definir la representación visual de la misma. Para ello se debe desplegar, haciendo click en el botón *"+"*, la clase que se quiere configurar, y hacer doble click en el atributo *"GraphRep (Metamodel)"*. Tal como se explicó en la sección anterior, y como se muestra en la Figura 6.5, se debe definir en lenguaje ADOscript la visualización de la misma.

- **Nuevo atributo:** Para crear un nuevo atributo se debe seleccionar, de las clases disponibles dentro de la carpeta *"Classes"* o *"Relation classes"*, la clase a la cual se le quiere agregar un atributo, luego hacer click en el botón *"New"* y se debe seleccionar del menú que se despliega la opción *"New attribute…"*. A continuación se mostrará la ventana expuesta en la Figura 6.10, en la que se debe escribir el nombre del atributo y se debe seleccionar el tipo del mismo. Una vez completada la edición se debe hacer click en el botón *"OK"* para finalizar con la creación del atributo.

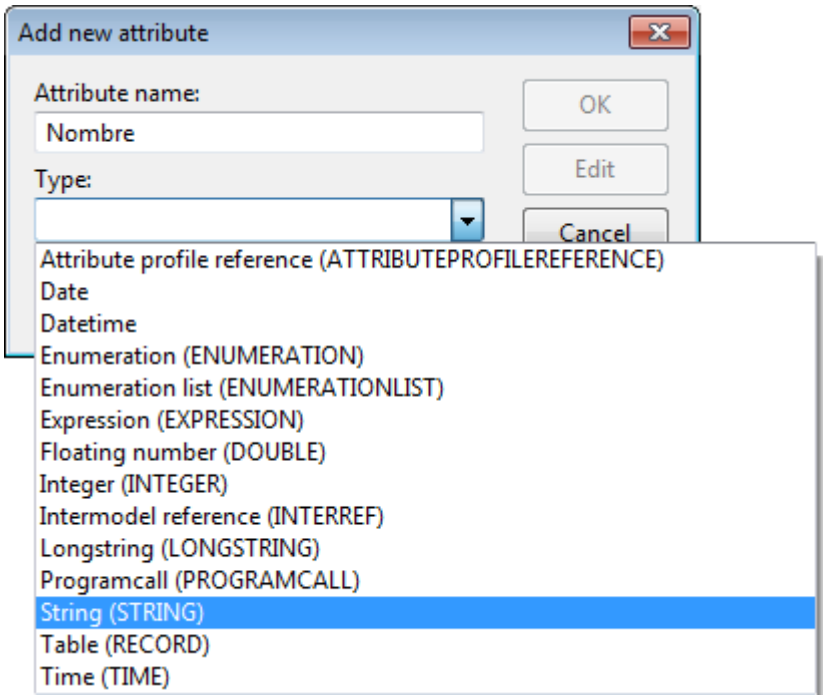


Figura 6.10 - *Configuración nuevo atributo.*

Si el tipo seleccionado es *"Enumeration (ENUMERATION)"* o *"Enumeration list (ENUMERATIONLIST)"*, previamente a la creación del atributo, se mostrará la ventana de la Figura 6.11, en la cual se deben agregar los valores posibles que puede tomar el atributo definido.

Una vez creados todos los atributos de una clase se debe definir cuáles son visibles para el usuario y cuál es la manera de visualizarlos. Para ello se debe desplegar, haciendo click en el botón *"+"*, la clase que se quiere configurar, y hacer doble click en el atributo *"AttrRep (Metamodel)"*. Es aquí en donde se define qué atributos son obligatorios; para ello se debe indicar la palabra clave "`MANDATORY`" a aquellos atributos que deben poseer un valor. Tal como se explicó en la sección anterior, y como se muestra en la Figura 6.4, se debe definir en lenguaje ADOscript la visualización de los mismos.

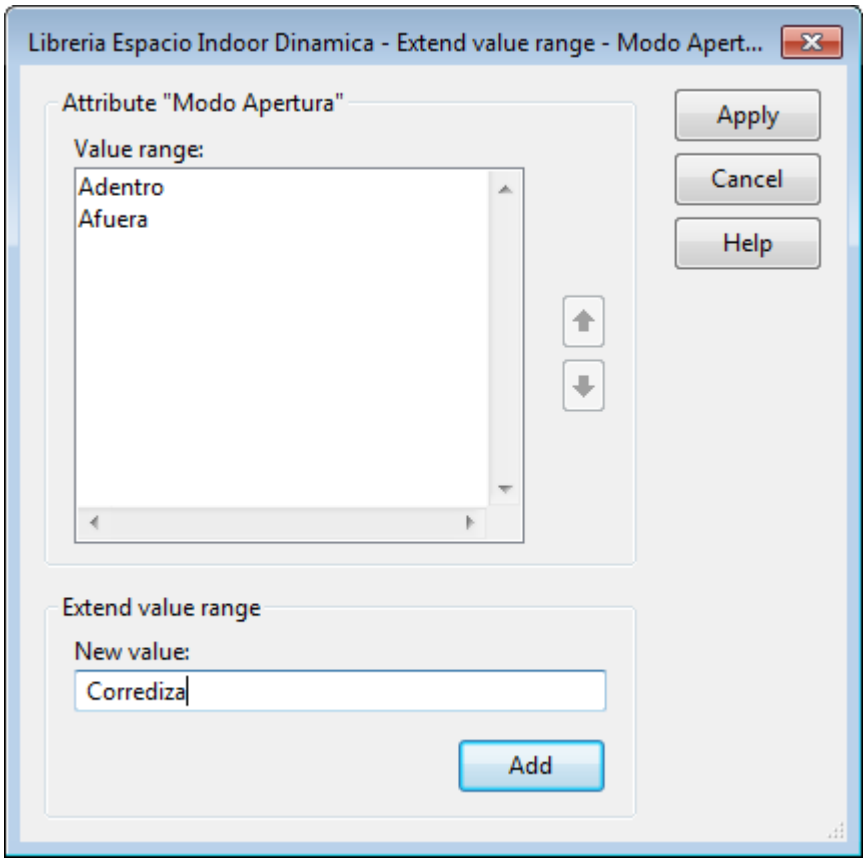


Figura 6.11 - *Configuración nuevo atributo Enumeration/Enumeration list.*

- **Nueva relación:** Para crear una nueva relación de clases se debe seleccionar la carpeta *"Relation clases"*, hacer click en el botón *"New"* y luego se debe seleccionar del menú que se despliega la opción *"New relationclass…"*. A continuación se mostrará la ventana expuesta en la Figura 6.12, en la que se debe escribir el nombre de la nueva relación y se deben seleccionar las clases que conecta la misma (origen y destino). Una vez completada la edición se debe hacer click en el botón *"OK"* para finalizar con la creación de la relación.

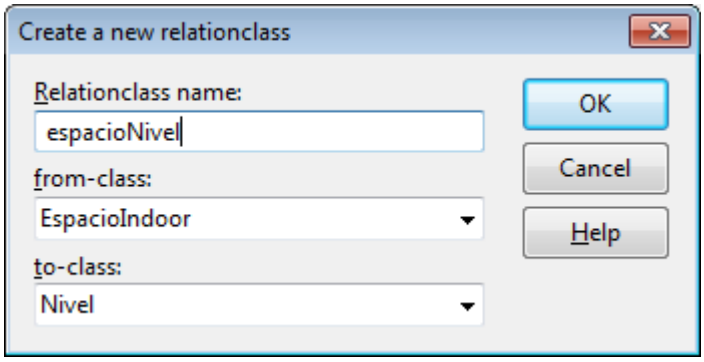


Figura 6.12 - *Configuración nueva relación.*

Luego de crear la relación se debe definir la representación visual de la misma. Al igual que para la visualización de clases, se debe desplegar, haciendo click en el botón *"+"*, la relación que se quiere configurar, y hacer doble click en el atributo *"GraphRep (Metamodel)"*. En la Figura 6.13 se muestra un ejemplo de representación visual de la relación definida anteriormente.

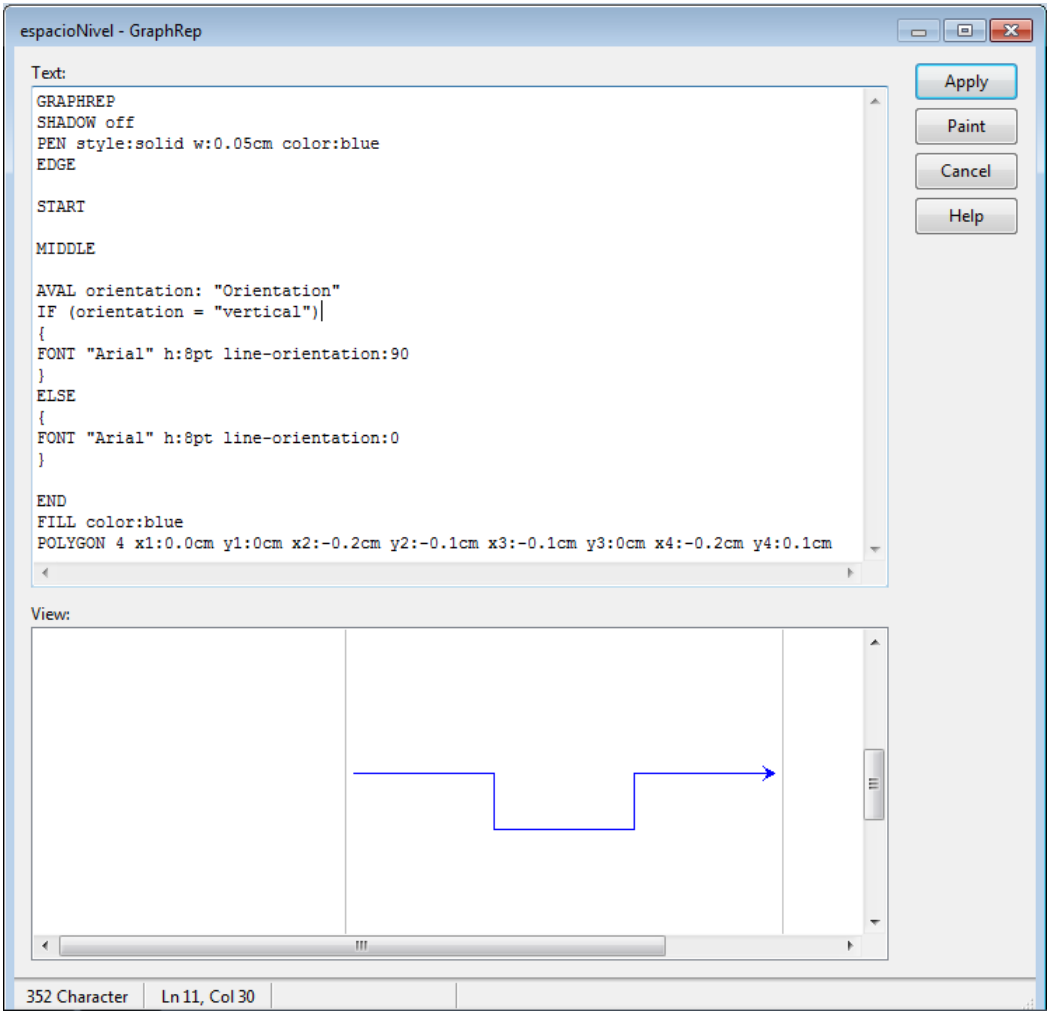


Figura 6.13 - *Definición en ADOscript de la relación "espacioNivel" en el atributo ATTRREP del meta metamodelo ADOxx y su representación visual.*

Luego de generar las clases y relaciones es necesario definir, para los casos en que sea necesario, las cardinalidades de cada una. Para ello se debe desplegar, haciendo click en el botón *"+"*, la clase que se quiere configurar, y hacer doble click en el atributo *"Class cardinality (Metamodel)"*. En dicho atributo se puede definir cuántas relaciones pueden conducir desde o hacia la clase particular y el número de instancias permitido en el modelo. Si no se especifica ninguna cardinalidad se supone que el número de relaciones no está restringido.

En la Figura 6.14 se muestra la definición, en lenguaje ADOscript, de las cardinalidades para la clase *"AmbienteSimple"* del metamodelo *"Espacio Indoor"*. Ellas son:

- En el modelo debe existir como mínimo una instancia de la clase *"AmbienteSimple"* y como máximo pueden existir N instancias.
- De cada instancia de la clase debe salir como mínimo una relación *"ambienteSensor"* a una instancia de la clase *"SensorDePosicionamiento"* y puede recibir como máximo N relaciones.
- Cada instancia de la clase debe recibir sólo una relación *"nivelAmbiente"* desde una instancia de la clase *"Nivel"*.
- Cada instancia de la clase debe recibir como mínimo una relación *"accesoAmbiente"* desde una instancia de las subclases de la clase *"Acceso"* y como máximo puede recibir N relaciones.

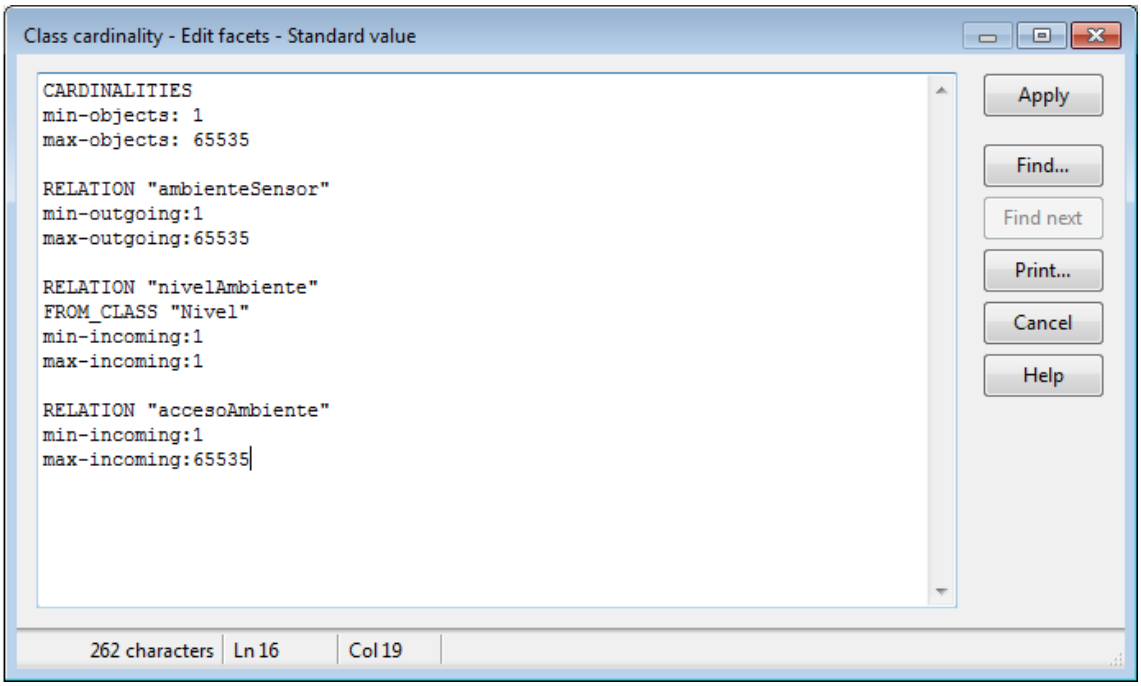


Figura 6.14 - *Ejemplo de cardinalidades definidas para la clase "AmbienteSimple" del metamodelo "Espacio Indoor".*

Una vez generadas todas las metaclases y relaciones junto con sus atributos y su representación visual, se debe indicar qué clases y relaciones serán incluidas en la

paleta *"Modelling"* de la herramienta *"Modelling toolkit"*. Para ello dentro de *"Development toolkit >> Library Management"* se debe acceder a la pestaña *"Settings"* y hacer click en el botón *"Library attributes"*; y en el atributo *"Modi"* dentro de la pestaña *"Adds-on"* se deben definir, en lenguaje ADOscript, todas las clases que se incluirán. En la Figura 6.15 se muestra la configuración respectiva al metamodelo *"Espacio Indoor"*.

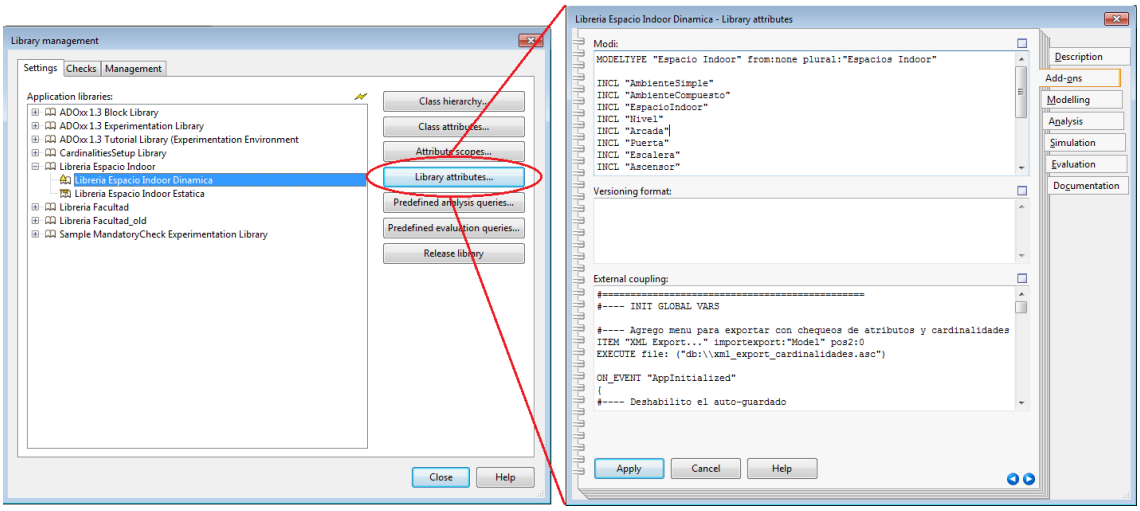


Figura 6.15 - *Configuración de clases y relaciones incluidas en la herramienta "Modelling toolkit" del metamodelo "Espacio Indoor".*

6.4.2 XML Export

Como se mencionó al comienzo de esta sección, una de las características que posee ADOxx es que provee funcionalidades predefinidas, entre ellas la exportación en formato XML de los modelos instanciados. Esta funcionalidad toma cada elemento y propiedad del modelo instanciado, los transforma en tags XML y genera un archivo con extensión .xml en el cual se encuentra toda la información del modelo.

De acuerdo a la definición final del metamodelo *"Espacio Indoor"* expuesta en el capítulo 4 y luego del estudio realizado, concluimos que es necesario realizar una validación previa a la exportación del modelo instanciado para verificar que el mismo cumpla con las restricciones OCL definidas. Es por ello que redefinimos la funcionalidad existente en ADOxx para que, antes de generar el archivo .xml, se realice una validación de cardinalidades y atributos obligatorios de todas las clases instanciadas. Durante el proceso de exportación se validan las siguientes restricciones:

- Para todas las clases el atributo nombre debe ser obligatorio
- Cada Ambiente debe tener al menos un Acceso
- Cada Ambiente debe tener al menos un SensorDePosicionamiento
- El TipoDeAcceso Ascensor no puede tener seteado el atributo esDeEmergencia en true.

- El atributo ubicación de EspacioIndoor debe ser obligatorio
- El atributo alto de Ambiente debe ser mayor que 0
- El atributo ancho de Ambiente debe ser mayor que 0
- El atributo profundidad de Ambiente debe ser mayor que 0
- El atributo posición de SensorDePosicionamiento debe ser obligatorio
- El atributo posición de Acceso debe ser obligatorio
- El atributo longitud de Rampa debe ser mayor que 0
- El atributo pendiente de Rampa debe ser mayor que 0
- El atributo pendiente de Escalera debe ser mayor que 0

Si la validación falla no es posible generar el archivo; es decir que es condición necesaria que el modelo instanciado cumpla con las restricciones listadas anteriormente para poder crear el XML correspondiente.

6.5 Extensión del metamodelo para un dominio particular

Luego de desarrollar la librería para la implementación del metamodelo abstracto *"Espacio Indoor"*, tenemos como objetivo mostrar su utilización para implementar dominios específicos más concretos. Como se mencionó a lo largo de la tesina, seleccionamos como espacio indoor particular a las *"Facultades"* y tomamos una sección de la Facultad de Informática de la Universidad de La Plata para mostrar la instanciación utilizando dicha extensión.

En esta sección se detallan los pasos necesarios para extender la librería *"Espacio Indoor"* con el fin de desarrollar la herramienta "Modelador de facultades"; a su vez se detalla el mecanismo que utiliza ADOxx para la extensión de metamodelos, y finalmente se muestra la utilización de "Modelador de facultades" mediante la instanciación mencionada anteriormente.

6.5.1 Desarrollo en ADOxx Development Toolkit

Como se mencionó en secciones anteriores, ADOxx maneja la implementación de metamodelos a través de librerías por lo tanto es necesario crear una nueva para la definición del metamodelo *"Facultad"*. Como primera instancia debemos exportar la librería *"Espacio Indoor"*, para ello dentro de la herramienta de ADOxx *"Development toolkit >> Library Management"* accedemos a la pestaña *"Management"* y desde allí, seleccionando la librería que deseamos exportar, hacemos click en el botón *"Export…"*. A continuación aparecerá una ventana en la cual se debe elegir el directorio destino y luego de hacer click en el botón *"Export"* se creará un archivo .abl en el directorio seleccionado. El siguiente paso es importar la librería previamente

exportada, para ello dentro de la herramienta de ADOxx *"Development toolkit >> Library Management"* accedemos a la pestaña *"Management"* y desde allí importamos la librería, de la misma manera que se explicó en la sección anterior, renombrándola a *"Facultad"*. En la Figura 6.16 se muestra la ventana que se utiliza para realizar la importación.

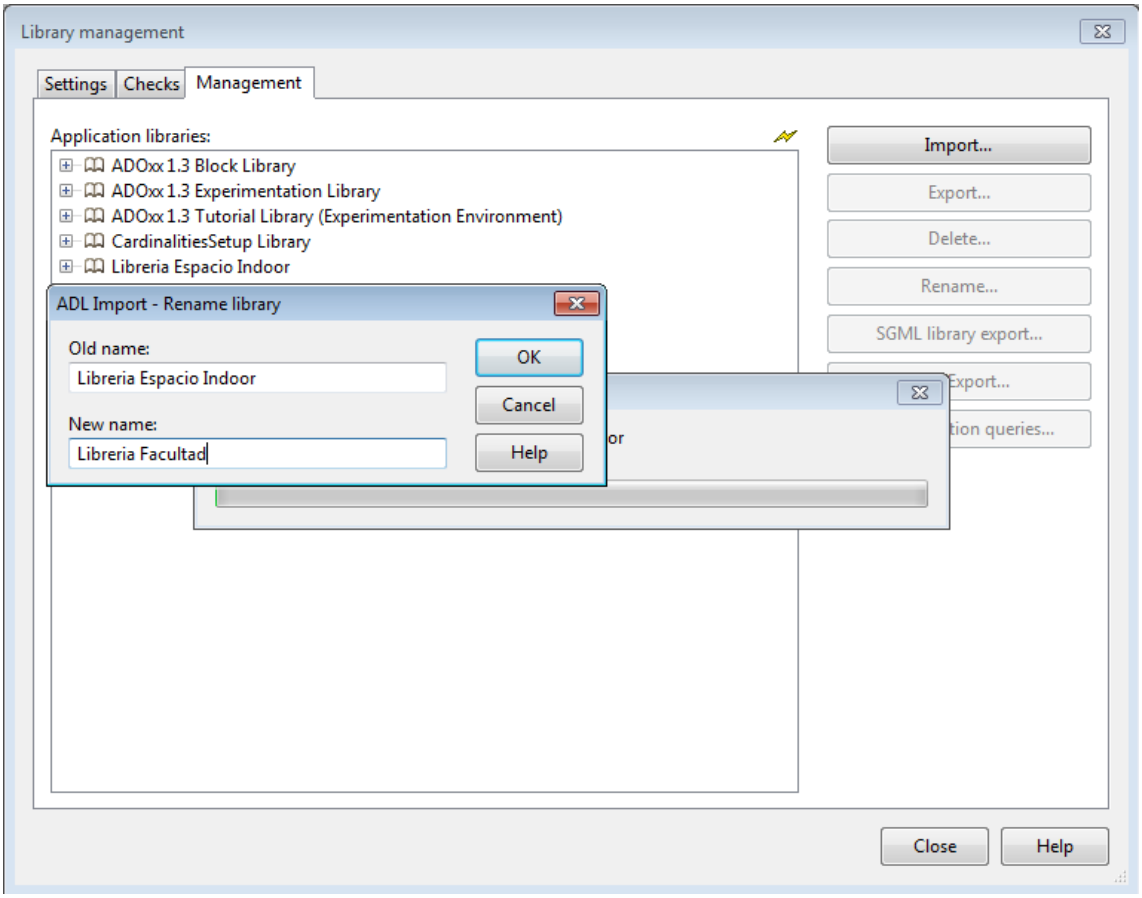


Figura 6.16 - *Importación librería "Espacio Indoor".*

Luego de realizar los pasos mencionados ya estamos en condiciones de comenzar a definir las particularidades del metamodelo *"Facultad"* debido a que, como se muestra en la Figura 6.17, ambas librerías son independientes, cada una con sus propias metaclases y relaciones.

Como se explicó en la sección 4.5 del capítulo 4 existen dos métodos para realizar la extensión de metamodelos. Por un lado el método *conservativo,* que no permite la modificación de las metaclases del metamodelo padre, sino que la manera de generar nuevas metaclases a las que se le pueden agregar nuevos atributos y comportamiento, es a través de la herencia y; por otro lado se encuentra el método *libre,* el cual permite la modificación de las metaclases (atributos y comportamiento) del metamodelo padre.

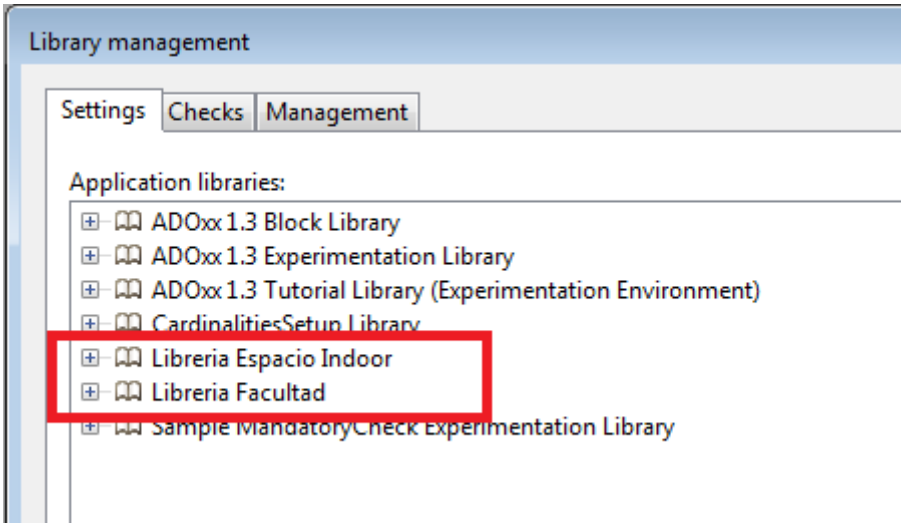


Figura 6.17 - *"Library management".*

Para la extensión de la librería, ADOxx utiliza el método libre, es decir que es posible modificar los atributos y comportamiento de las metaclases padre. Esto se debe a que ADOxx se rige bajo una licencia open source, con lo cual todo el código incluido en el repositorio es propiedad de la comunidad y cualquiera que tenga acceso al mismo puede modificar localmente las librerías descargadas.

El paso siguiente luego de realizar la importación, fue crear las metaclases particulares en la librería *"Facultad"*. Debido a que el metamodelo *"Facultad"* se definió basándose en el método de extensión conservativo, en la creación de la herramienta *"Modelador de Facultades"* las metaclases existentes en la librería *"Espacio Indoor"* no fueron modificadas. De la misma manera que se explicó en la sección anterior se crearon las metaclases que heredan de *"AmbienteSimple"*. Seleccionamos la metaclase *"AmbienteSimple"*, hacemos click en el botón *"New"* y seleccionamos la opción *"New class…"*.En la ventana que aparece escribimos el nombre de la nueva metaclase y luego de hacer click en *"Ok"* la metaclase nueva ya se encuentra creada. De esta manera se crearon las siguientes metaclases:

- Estacionamiento
- Aula
- Oficina
- Servicio
- Biblioteca
- Baño
- Espacio Abierto
- Hall

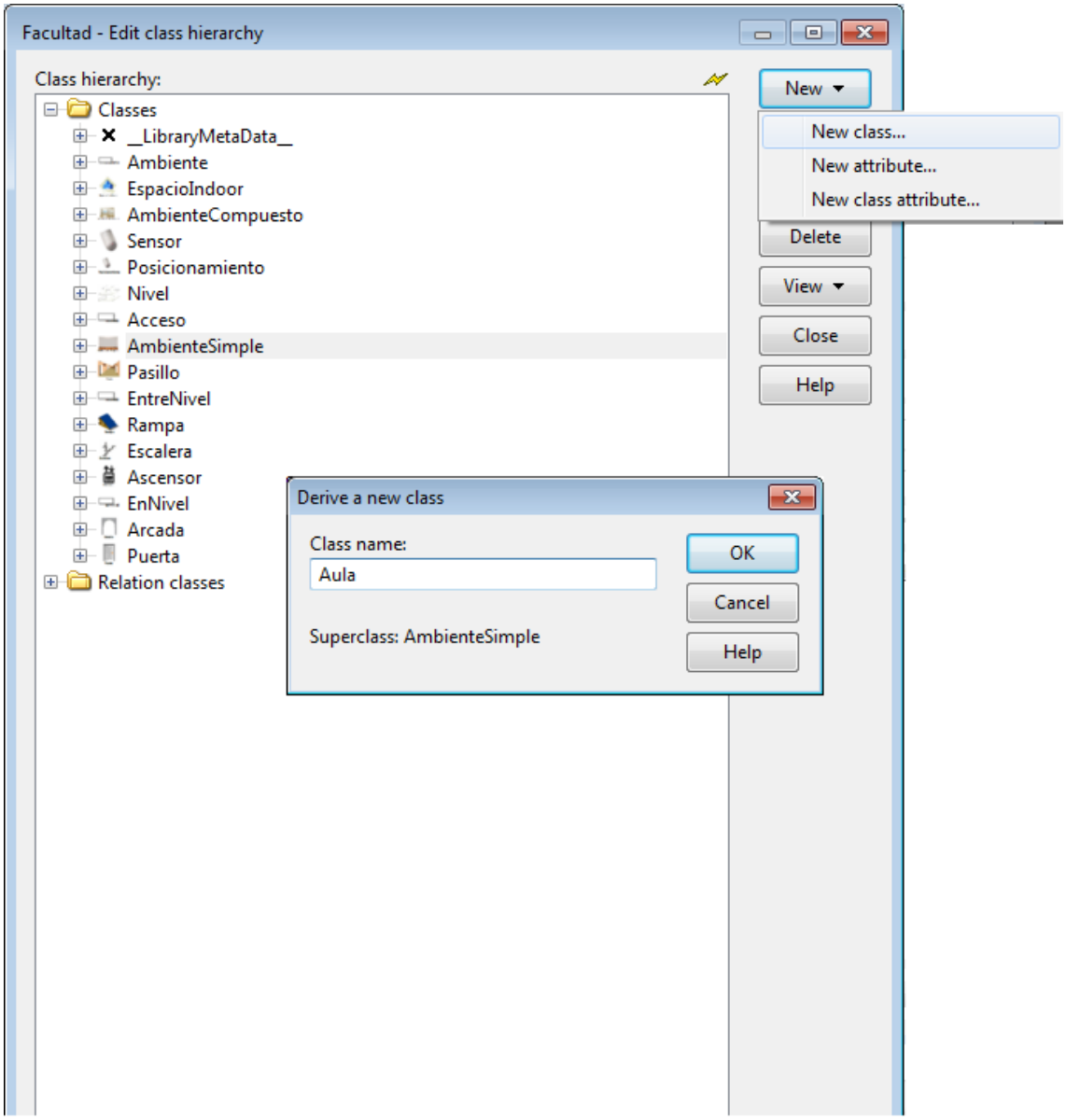


Figura 6.18 - *Creación de metaclase.*

Para la creación del resto de las metaclases se realizó el mismo procedimiento explicado anteriormente, pero seleccionando la metaclase padre correspondiente:

Metaclase hija >> Metaclase padre

- *"Facultad" >> "EspacioIndoor"*
- *"AulaEspecial" >> "Aula"*
- *"Fotocopiadora" >> "Servicio"*
- *"Buffet" >> "Servicio"*
- *"PatioInterno" >> "EspacioAbierto"*

Luego de la creación de las metaclases particulares de *"Facultad"* se definió su representación visual y se crearon los atributos correspondientes a cada una de ellas junto con su visualización, de la misma manera que se explicó en la sección anterior. Además, para la clase *"Baño"* se definió como restricción que al menos deben existir 2

(dos) instancias de la misma (ya que es necesario un baño para caballeros y otro para damas); y para las clases *"Aula"* y *"Biblioteca"* se definió que debe existir al menos 1 (una) instancia de cada una en el modelo.

Una vez finalizados los pasos anteriores ya se tienen definidos todos los conceptos y características de la librería *"Facultad"*. Para completar la definición se debió modificar el atributo *"Modi"* de la librería para indicar qué clases y relaciones son visibles para el usuario en la herramienta *"Modelador de facultades"*; debido a que no son las mismas que utiliza la librería *"Espacio Indoor"*. Para ello dentro de *"Development toolkit >> Library Management"* se debe acceder a la pestaña *"Settings"* y hacer click en el botón *"Library attributes"*; dentro de la pestaña *"Adds-on"* se encuentra el atributo mencionado, y es allí donde se realizaron las modificaciones necesarias. En la Figura 6.19 se muestran las configuraciones de dicho atributo para la librería *"Espacio Indoor"* y *"Facultad"* respectivamente.

```
MODELTYPE "Espacio Indoor"
from:none plural:"Espacios Indoor"

INCL "AmbienteSimple"
INCL "AmbienteCompuesto"
INCL "EspacioIndoor"
INCL "Nivel"
INCL "Arcada"
INCL "Puerta"
INCL "Escalera"
INCL "Ascensor"
INCL "Rampa"
INCL "Pasillo"
INCL "Posicionamiento"
INCL "espacioNivel"
INCL "compuestoPor"
INCL "accesoAmbiente"
INCL "ambienteSensor"
INCL "nivelAmbiente"
```

```
MODELTYPE "Facultad" from:none
plural:"Facultades"

INCL "AmbienteCompuesto"
INCL "Aula"
INCL "Aula Especial"
INCL "Oficina"
INCL "Fotocopiadora"
INCL "Buffet"
INCL "Biblioteca"
INCL "Baño"
INCL "Patio interno"
INCL "Hall"
INCL "Estacionamiento"
INCL "Facultad"
INCL "Nivel"
INCL "Arcada"
INCL "Puerta"
INCL "Escalera"
INCL "Ascensor"
INCL "Rampa"
INCL "Pasillo"
INCL "Posicionamiento"
INCL "espacioNivel"
INCL "compuestoPor"
INCL "accesoAmbiente"
INCL "ambienteSensor"
INCL "nivelAmbiente"
```

Figura 6.19 - *Atributo "Modi" librería "Espacio Indoor" y "Facultad".*

Luego de realizar todos estos pasos nos encontramos en condiciones de comenzar a instanciar la sección de la Facultad de Informática seleccionada para mostrar la utilización de la herramienta *"Modelador de facultades".* A continuación, en la Figura 6.20 se muestra la interface de la herramienta creada.

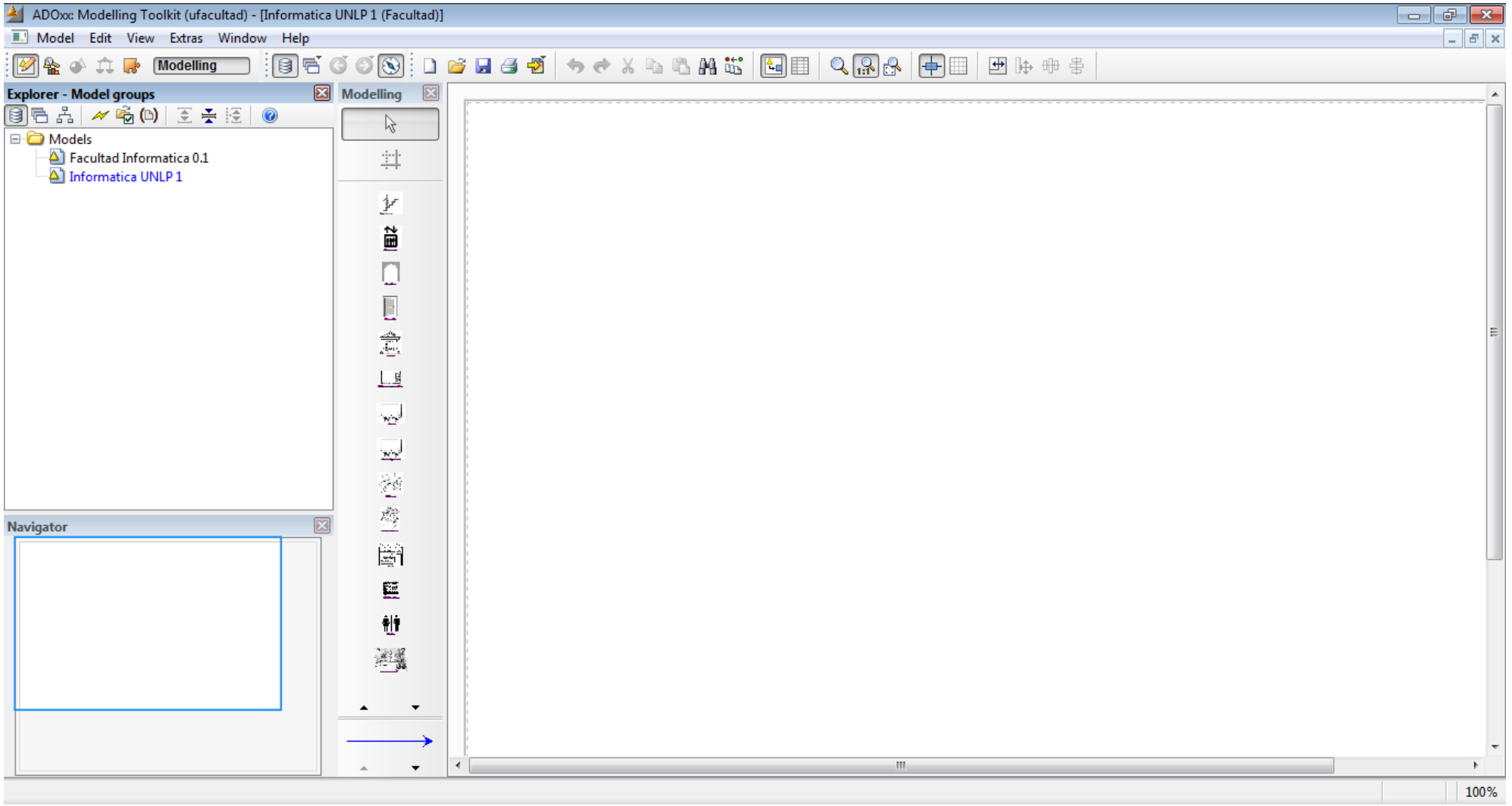


Figura 6.20 - *Interface herramienta "Modelador de Facultades".*

6.5.2 Instanciación en ADOxx Modelling Toolkit

Para comenzar con la instanciación de un nuevo modelo, como primera instancia es necesario crear un modelo vacío. Esta tarea se realiza desde el menú *"Model >> New"* donde se abrirá una ventana como la que se muestra en la Figura 6.21 en la que se debe ingresar el nombre del modelo, se debe seleccionar el grupo y tipo de modelo al que pertenece y, opcionalmente, se puede definir la versión del modelo a crear.

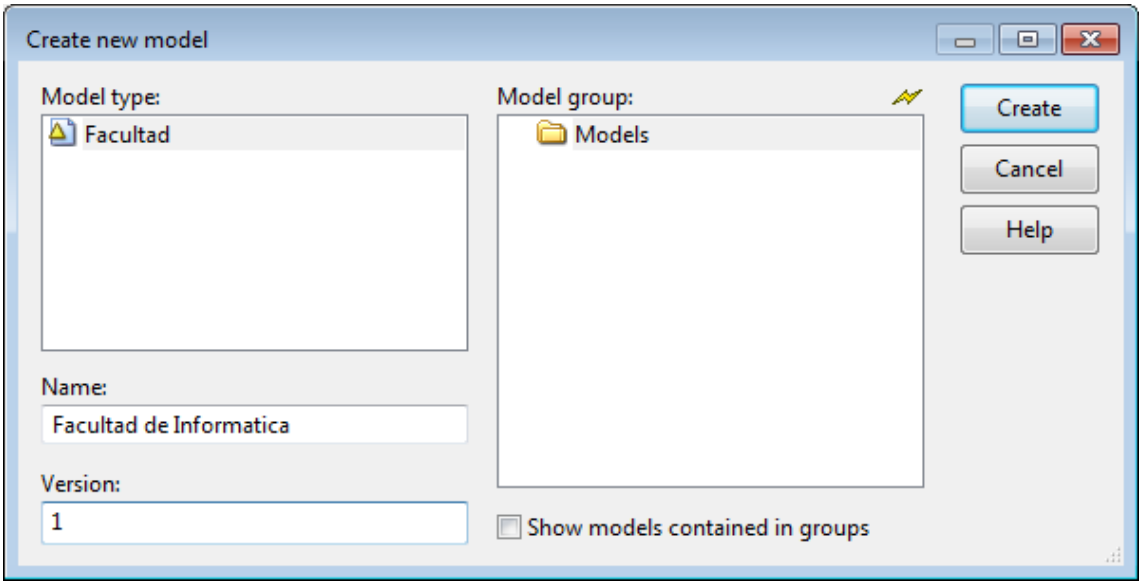


Figura 6.21 - *Creación de modelo vacío.*

Luego de hacer click en el botón *"Create"* hemos generado nuestro modelo vacío denominado "Facultad de Informática", en el que realizaremos la instanciación de la sección de la Facultad de Informática de la UNLP seleccionada.

6.5.2.1 Creación de clases y relaciones

Una vez creado el modelo vacío nos encontramos en el "modo edición", y ya estamos en condiciones de comenzar a incluir las clases y relaciones pertenecientes al modelo.

- **Nueva clase:** Desde el panel lateral izquierdo denominado *"Modelling"* se debe hacer click sobre el icono de la clase que se desea agregar. El mismo queda seleccionado y el puntero del mouse se convierte en una lapicera indicando que está permitido ubicar nuevas clases y relaciones dentro del área de modelado, es decir que nos encontramos en el "modo dibujo". Luego de seleccionar la clase deseada se debe hacer click dentro del área de modelado en la posición en donde se quiere insertar, de esta manera la clase queda creada como se muestra en la Figura 6.22. Una característica importante a destacar, es que la herramienta permite crear el número permitido de instancias posibles de cada clase de manera consecutiva mientras nos encontremos en el "modo dibujo".

Otra característica importante que posee la herramienta es que a medida que se dibujan instancias de la misma clase dentro del área de modelado, se realiza la validación de número máximo de instancias permitido de cada clase. En el metamodelo *"Facultad"* existe la restricción OCL que indica que a lo sumo puede existir una sola instancia de la clase "*Facultad*" (restricción heredada del metamodelo *"Espacio Indoor"*), por lo tanto si se intenta agregar dos clases *"Facultad"* al modelo se mostrará un error como el que aparece en la Figura 6.23, alertando al usuario que el modelo contiene 2 objetos de la clase *"Facultad"* y el máximo permitido es 1. Luego de hacer click en el botón "*OK*" automáticamente se cerrará la ventana de alerta y se eliminará la instancia de la clase que arrojó el error.

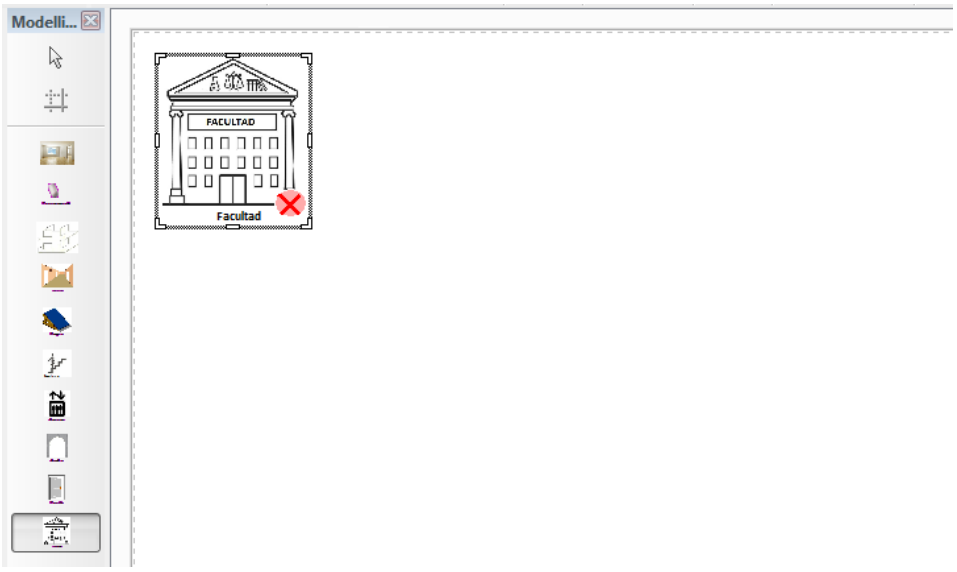


Figura 6.22 - *Instanciación de nueva clase.*

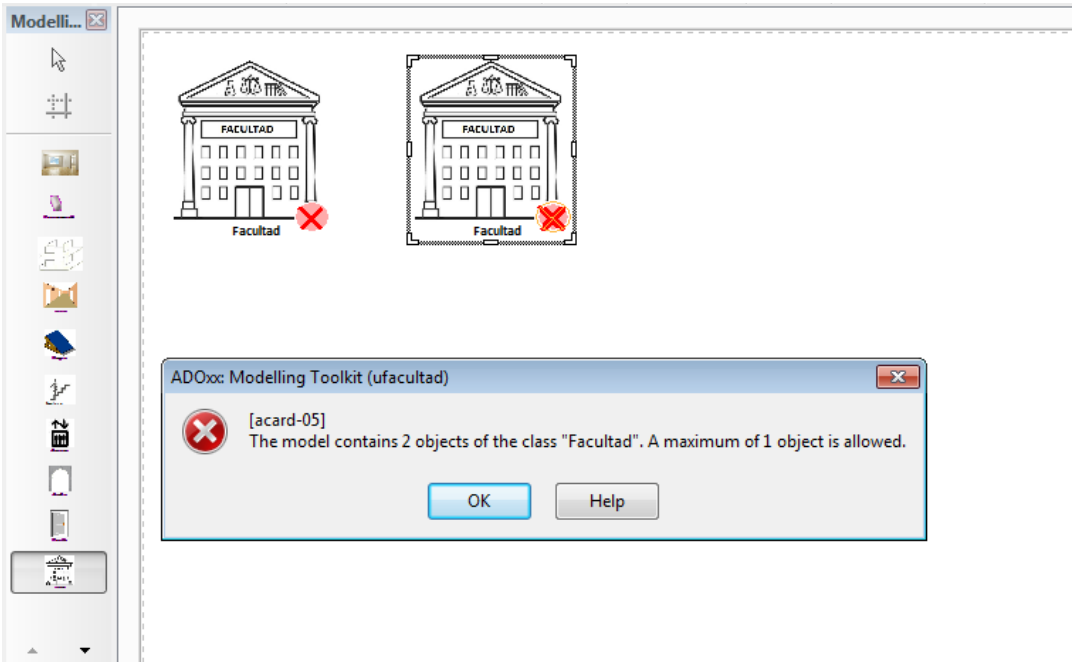


Figura 6.23 - *Error al validar número máximo de instancias.*

Si se desea agregar una clase diferente a la seleccionada dentro del modelo, simplemente se debe seleccionar desde el panel *"Modelling"* el icono correspondiente a la clase deseada y seguir los pasos mencionados anteriormente.

Existen 3 (tres) maneras para pasar del "modo dibujo" al "modo edición":

- Haciendo click derecho sobre un espacio libre en el área de modelado
- Haciendo click en el icono *"Edit"*  dentro del panel *"Modelling"*
- Presionando la tecla "Esc"

Como se puede ver en las Figuras 6.22 y 6.23, la instancia de la clase *"Facultad"* aparece con el símbolo  . Esto indica que la instancia de la clase contiene errores, como puede ser atributos obligatorios que no poseen valor o atributos que poseen un valor incorrecto (de acuerdo a las restricciones OCL definidas para cada clase). Una vez que se hayan definido correctamente todos los atributos a la clase, el símbolo desaparecerá indicando que la instancia cumple con todas las restricciones definidas para la clase correspondiente.

- **Nueva relación:** El mecanismo para crear nuevas relaciones dentro del modelo, es similar al procedimiento para crear nuevas clases. De la misma manera, desde el panel izquierdo denominado *"Modelling"* se debe hacer click sobre el ícono de la relación que se desea agregar. El mismo queda seleccionado y el puntero del mouse se convierte en una lapicera indicando que nos encontramos en el "modo dibujo". Es importante destacar que antes de crear la relación deben haberse creado previamente las clases que se desean conectar.

  Continuando, una vez seleccionada la relación a agregar, se debe posicionar el puntero del mouse sobre la clase origen del conector. Si la clase recibe el foco, significa que la misma es válida para la relación de acuerdo a las definiciones descriptas en la librería. Luego debe hacerse click sobre dicha clase y se debe mover el puntero del mouse hacia la clase destino del conector (al igual que en la selección de la clase origen, si el objeto recibe el foco implica que la clase destino es válida para la relación). Finalmente se debe hacer click sobre la clase destino para concretar la creación del conector. En la Figura 6.24 se muestra un ejemplo de la relación *"espacioNivel"* que conecta la clase *"Facultad"* con la clase *"Nivel".* De la misma manera que se pueden crear clases consecutivas, la herramienta brinda la funcionalidad equivalente para las relaciones. Para ello se debe mantener presionada la tecla CTRL mientras se selecciona la clase origen y, mientras la misma se mantenga con el foco, se pueden agregar relaciones hacia las clases destino.

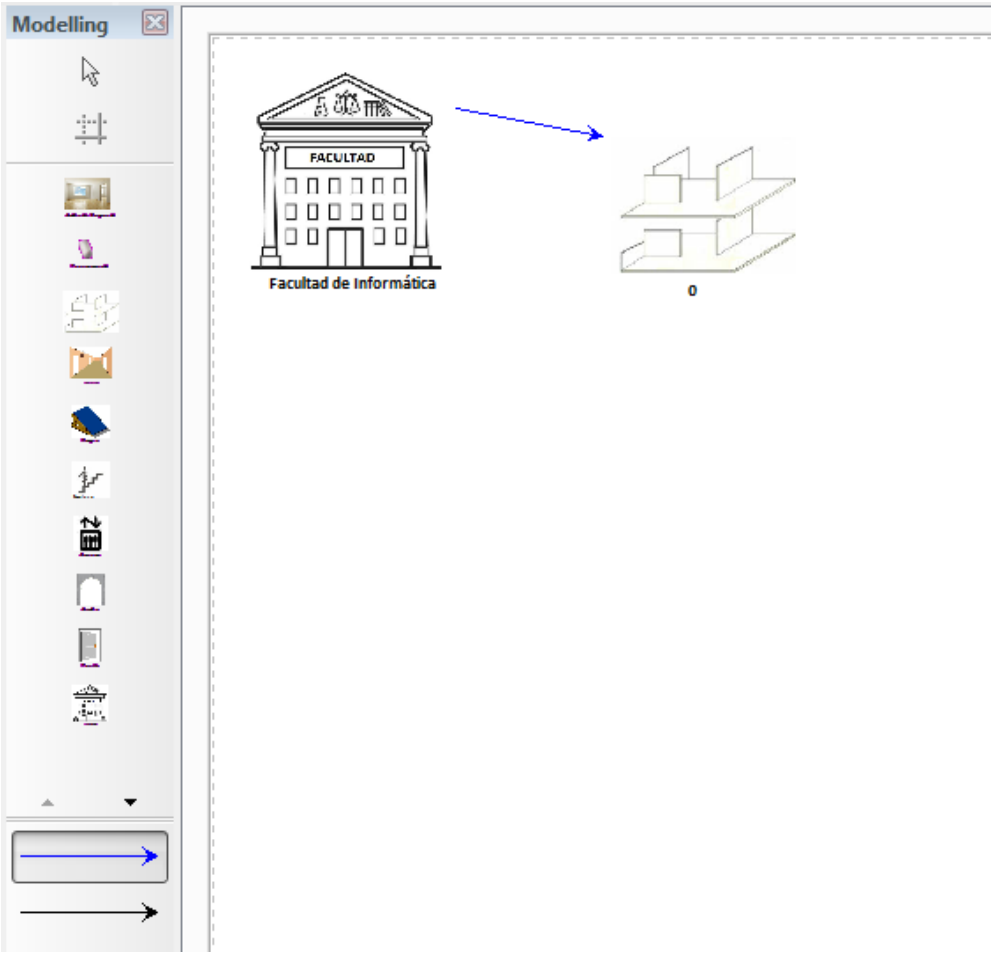


Figura 6.24 - *Instanciación de nueva relación.*

Otra manera de crear una nueva relación es a través del menú contextual de las clases. Para ello se debe hacer click derecho sobre la instancia de clase a la que se le desea agregar un conector y seleccionar del menú que se despliega la opción *"New relation"*. A continuación se listarán todas las relaciones posibles (de entrada y salida) sobre la clase seleccionada. Luego de seleccionar el conector deseado aparecerá una ventana en la que se debe elegir la clase origen o destino, dependiendo de la relación seleccionada, y luego de hacer click en el botón *"Create"* la relación queda efectivamente creada y dibujada sobre el modelo. La Figura 6.25 muestra el escenario planteado.

Durante la creación de relaciones se realizan ciertas validaciones que permiten la detección de errores durante el proceso de modelado, como así también se verifican algunas de las restricciones definidas en el metamodelo. Estas funcionalidades tienen como ventaja que sea más fácil para el usuario llegar a construir un modelo que sea válido para el metamodelo propuesto.

- Una de las validaciones que se realiza a medida que se agregan relaciones al modelo, es verificar que una clase no contenga más conectores (de entrada o de salida, dependiendo de la relación) que el

máximo permitido. Por ejemplo, en el metamodelo *"Facultad"* existe como restricción que un *"Acceso"* puede tener a lo sumo 2 (dos) relaciones de salida del tipo *"accesoAmbiente"*, por lo tanto si durante la generación del modelo se desea agregar una tercera relación a un *"Acceso"* que ya contiene dos, se generará un error como el que se muestra en la Figura 6.26 alertando al usuario que la clase ya contiene el máximo número de relaciones permitidas, por lo que no es posible agregar la nueva. Luego de hacer click en el botón *"OK"*, se cerrará la ventana y se eliminará la relación que provocó el error.

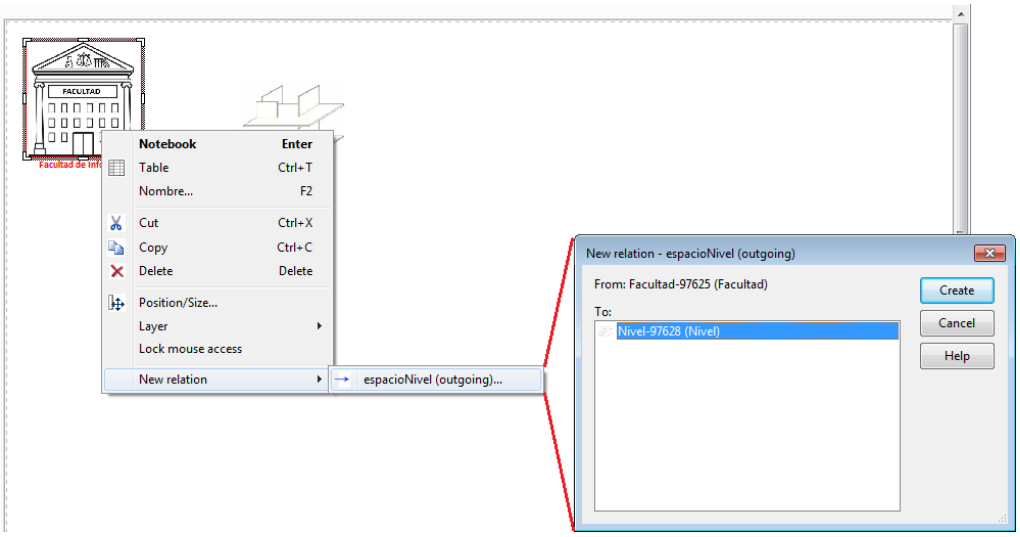


Figura 6.25 - *Instanciación de nueva relación a través del menú contextual.*

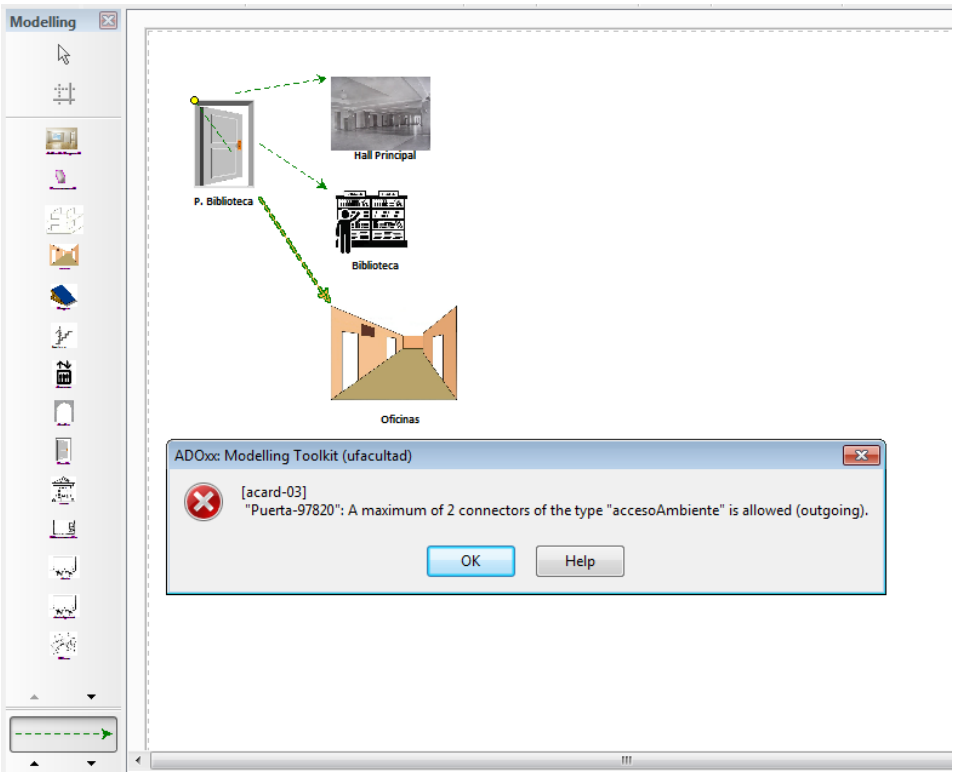


Figura 6.26 - *Error generado al agregar más relaciones que el máximo permitido.*

- Otra de las validaciones que se realiza en el proceso de modelado de las relaciones, es verificar que las clases origen y destino de un conector correspondan a las definidas en la librería. Por ejemplo, si el usuario quiere agregar una relación del tipo *"accesoAmbiente"* entre las clases *"Facultad"* y *"Aula"* la herramienta arrojará un error como se muestra en la Figura 6.27 indicando que no es posible concretar la instanciación de la relación, debido a que, de acuerdo al metamodelo *"Facultad"*, el tipo de relación mencionado debe conectar una instancia de la clase *"Acceso"* con una instancia de la clase *"Ambiente"*; e indica cuáles clases pueden utilizarse como origen y cuáles como destino. Luego de presionar el botón *"OK"*, se cierra la ventana y se elimina automáticamente la relación que provocó el error.

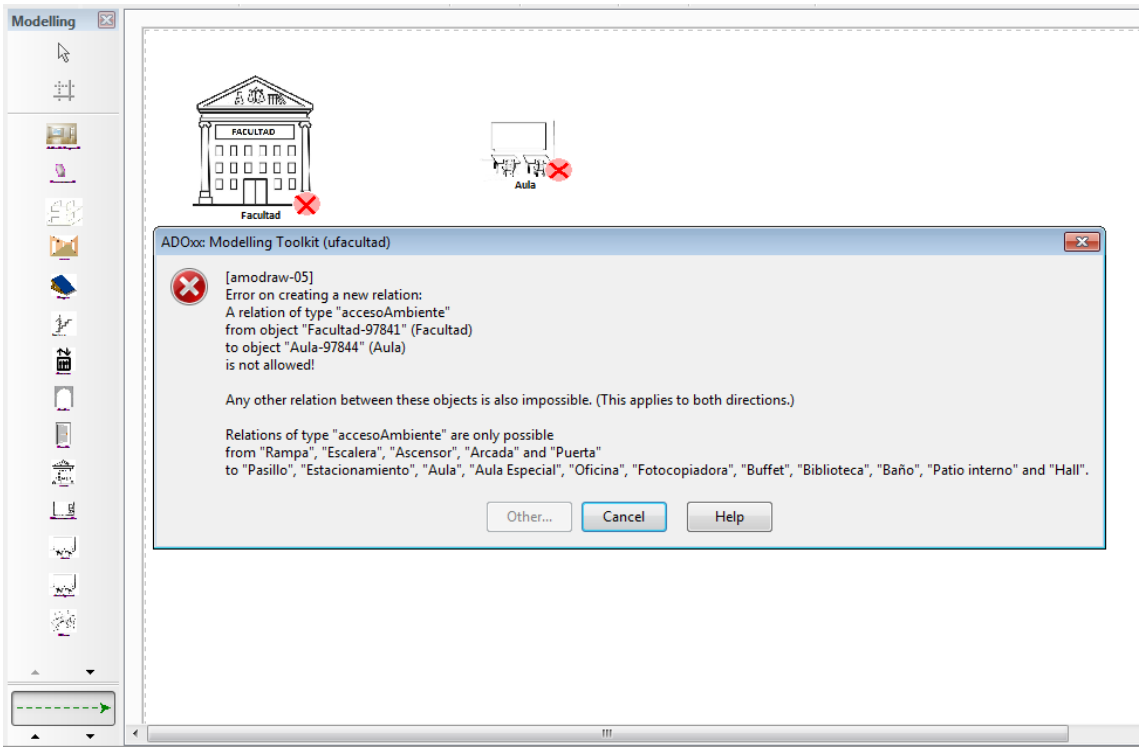


Figura 6.27 - *Error generado al instanciar una relación entre clases incorrectas.*

- Como última validación en el proceso de creación de relaciones, la herramienta generará un error como el de la Figura 6.28 si el usuario desea agregar dos relaciones del mismo tipo desde la misma clase origen hacia una misma clase destino, indicando que la relación que se quiere agregar ya existe. Esta validación provoca que el modelo que se está definiendo cumpla con la siguiente restricción OCL definida en el metamodelo *"Facultad":*

  - Cada Acceso debe conectar dos Ambientes distintos

  Luego de presionar el botón *"OK"*, la ventana se cerrará y se eliminará de manera automática la relación que provocó el error.

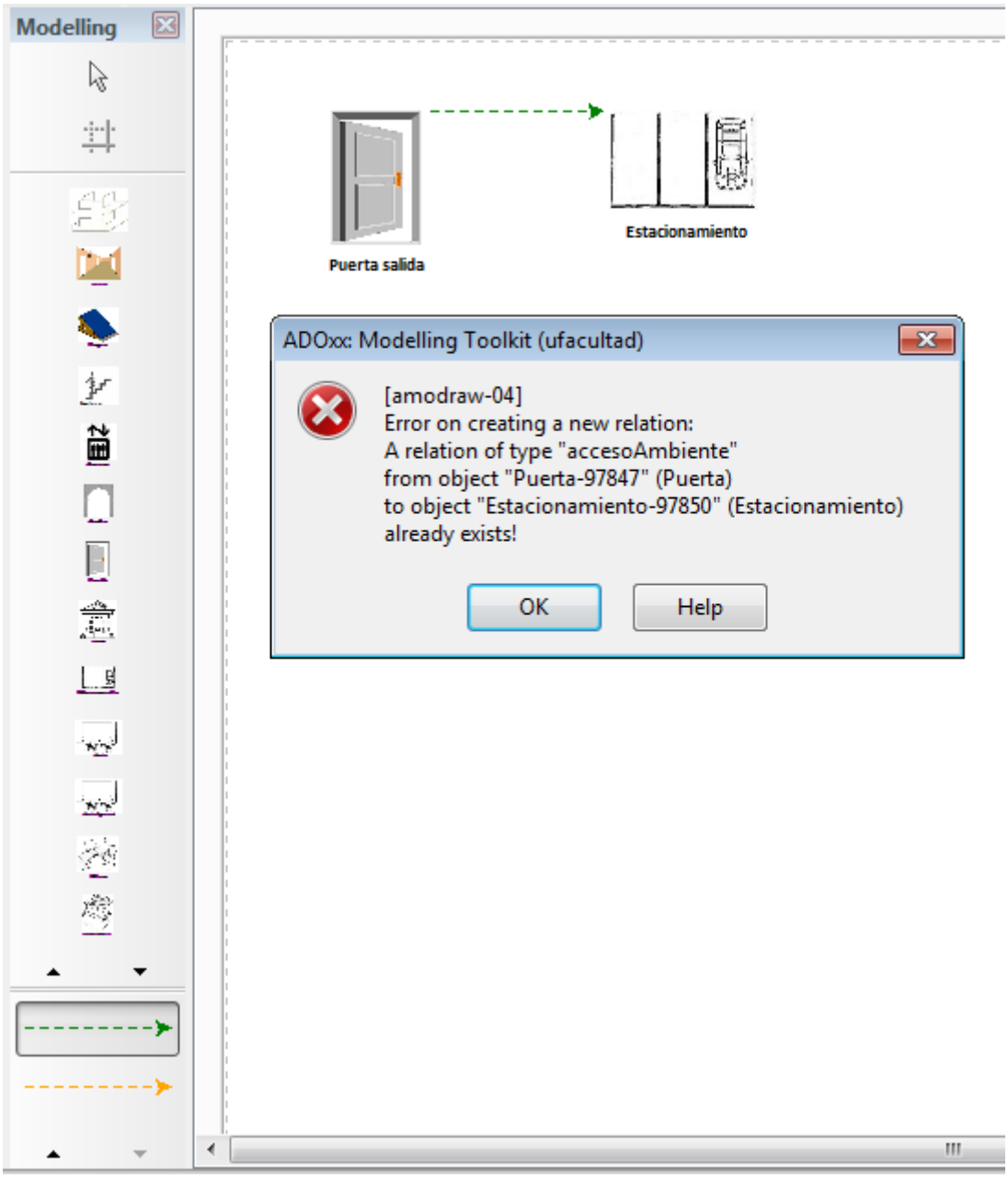


Figura 6.28 - *Error generado al crear una relación ya existente en el modelo.*

6.5.2.2 Definición de atributos

Para definir el valor de los atributos propios de cada clase es necesario haber creado previamente la instancia de la clase a la que se le quieren configurar los atributos. De la misma manera en que durante la creación de clases y relaciones se realizan validaciones y se verifica que el modelo cumpla con algunas restricciones del metamodelo *"Facultad"*, en el momento de configurar los valores de los atributos durante el proceso de modelado, también se realizan validaciones para verificar que el modelo cumpla con otras de las restricciones necesarias para que el modelo sea válido.

Para configurar el valor de los atributos de una determinada instancia de clase, se debe hacer doble click sobre dicha clase o hacer click derecho sobre la clase y elegir de la lista que se despliega la opción "***Notebook***". Luego de ejecutar cualquiera de las opciones mencionadas se abrirá una ventana que muestra los atributos propios de la clase a los cuales se les puede asignar un valor determinado. En la Figura 6.29 se presenta como ejemplo la ventana de atributos para la clase *"Aula"*.

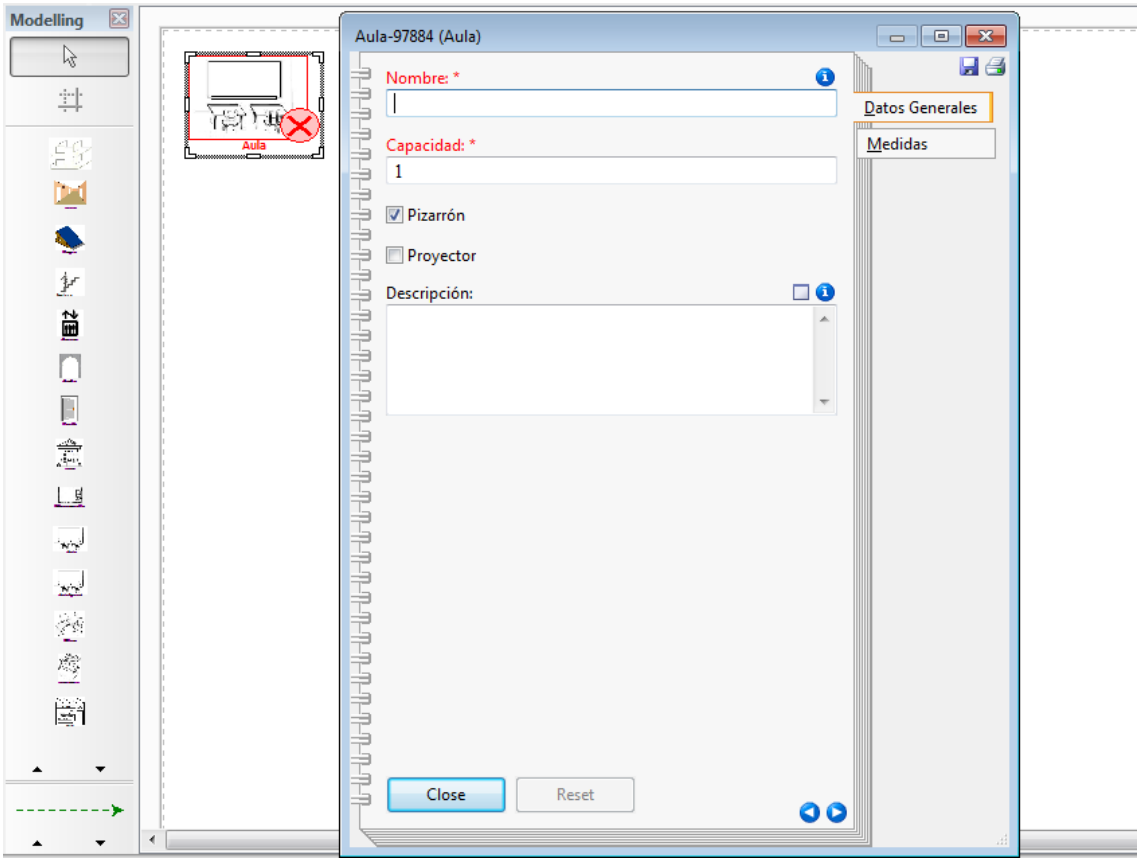


Figura 6.29 - *Definición de atributos.*

Es importante destacar que la ventana de atributos se mostrará sólo para aquellas clases o relaciones que definieron el atributo *"AttrRep (Metamodel)"* en la librería *"Facultad"*.

Como se puede ver en la Figura 6.29 la ventana está estructurada como un fichero, con un conjunto de pestañas (en el ejemplo se denominan *"Datos Generales"* y *"Medidas"*) ubicadas del lado derecho. Al hacer click sobre cada una se mostrarán los atributos respectivos definidos para cada sección.

Como hemos mencionado a lo largo de la tesina, para enriquecer el metamodelo propuesto se definieron ciertas restricciones que un modelo debe cumplir para que sea válido. Entre las restricciones definidas existen varias que indican que un atributo debe ser obligatorio; por lo tanto para facilitar al usuario la detección de dichos atributos de cada clase, en la ventana se presentan en color rojo y con el símbolo "*" indicando que debe introducirse un valor obligatoriamente.

Para editar los atributos de una clase el usuario debe introducir el valor deseado en el campo del atributo. Dichos valores serán validados cuando el usuario retire el foco del campo editado. La herramienta brinda las siguientes validaciones:

- Se verifica que el tipo de dato ingresado sea válido. En el ejemplo se arrojará un error como se muestra en la Figura 6.30 si al campo *"Capacidad"* de la pestaña *"Datos Generales"* o a alguno de los campos *"Alto"*, *"Ancho"* y *"Profundidad"* de la pestaña *"Medidas"* se le asigna un string en lugar de un entero o decimal, según corresponda.

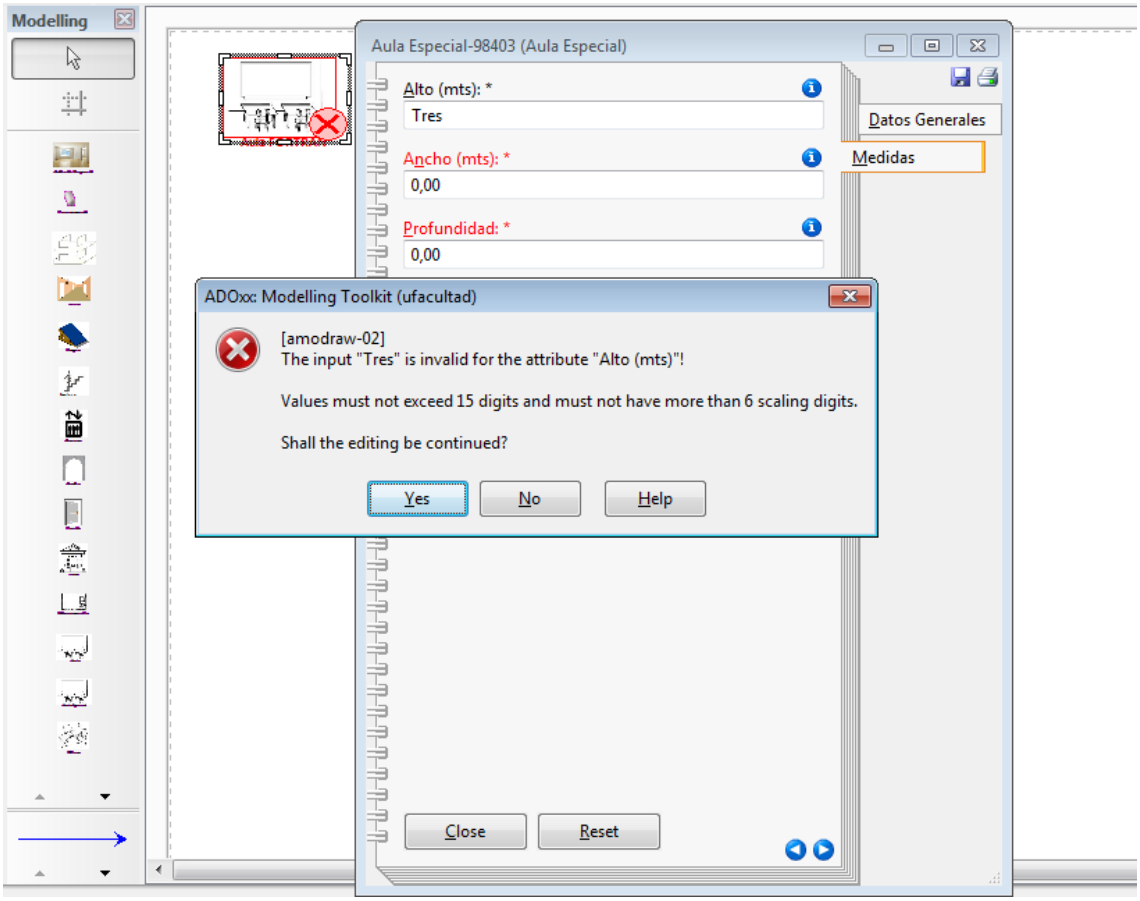


Figura 6.30 - *Definición de atributos: Error tipo de dato incorrecto.*

- Una vez ingresado el valor para un atributo se valida que el mismo se encuentre dentro del rango permitido, de esta manera se verifica que el modelo cumpla con las restricciones OCL, listadas a continuación, definidas para el metamodelo *"Facultad"*:
    - El atributo ancho de un TipoDeAcceso EnNivel debe ser >= 0.80.
    - El atributo alto de un TipoDeAcceso EnNivel debe ser >= 2.
    - El atributo cantEscalones de un TipoDeAcceso Escalera debe ser > 0.
    - El atributo cantAccesos de un TipoDeAcceso Ascensor debe ser > 0.

- El atributo capacidadMaxima de un TipoDeAcceso Ascensor debe ser > 0.
- El atributo alto de Ambiente debe ser > 0.
- El atributo ancho de Ambiente debe ser > 0.
- El atributo profundidad de Ambiente debe ser > 0.
- El atributo longitud de Rampa debe ser > 0.
- El atributo pendiente de Rampa debe ser > 0.
- El atributo pendiente de Escalera debe ser > 0.
- El atributo capacidad de Aula debe ser > 0.
- El atributo capacidad de Estacionamiento debe ser > 0.
- El atributo capacidad de Baño debe ser > 0.

En el caso de que no cumpla con la restricción, se mostrará un error como el de la Figura 6.31 en el que indica que el valor ingresado es incorrecto e informa cuál es el rango permitido. En el ejemplo se muestra la validación del campo *"cantAccesos"* de la clase *"Ascensor"*, el cual tiene que ser mayor que 0 (cero).

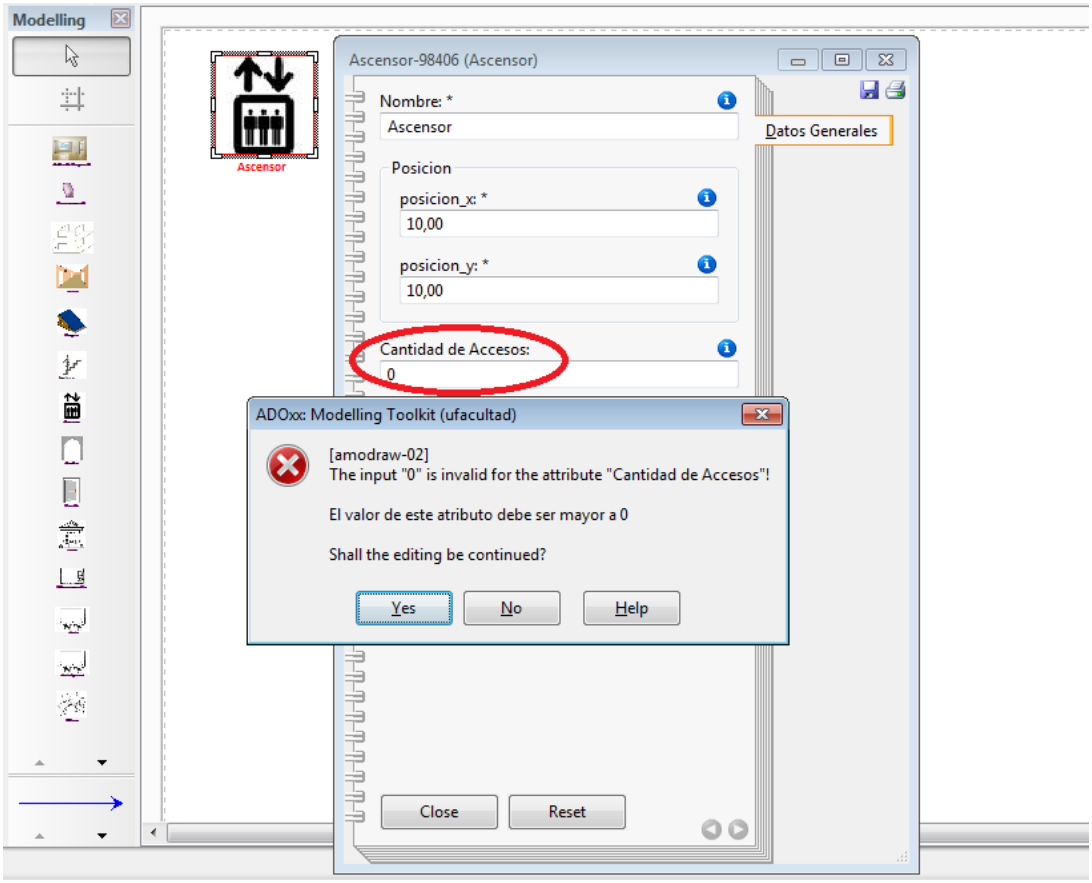


Figura 6.31 - *Definición de atributos: Error al ingresar un valor fuera del rango permitido.*

Como se explicó en la sección 6.5.2.1, al crear una nueva clase la misma aparece con el símbolo  indicando que contiene errores en la definición de sus atributos; ya sean atributos obligatorios sin valor o atributos con un valor incorrecto. Luego de lo expuesto en esta sección estamos en condiciones de definir correctamente los atributos para, por ejemplo, la clase *"Facultad"* que mostrábamos con error en la sección anterior. En la Figura 6.32 se muestra la clase junto con la correcta definición de los atributos. Se puede observar que, además de eliminar el símbolo de error de la clase, en la ventana de los atributos se puede ver que aquellos que aparecían en rojo indicando que son obligatorios, al completarlos con un valor válido, se elimina el color que indica error.

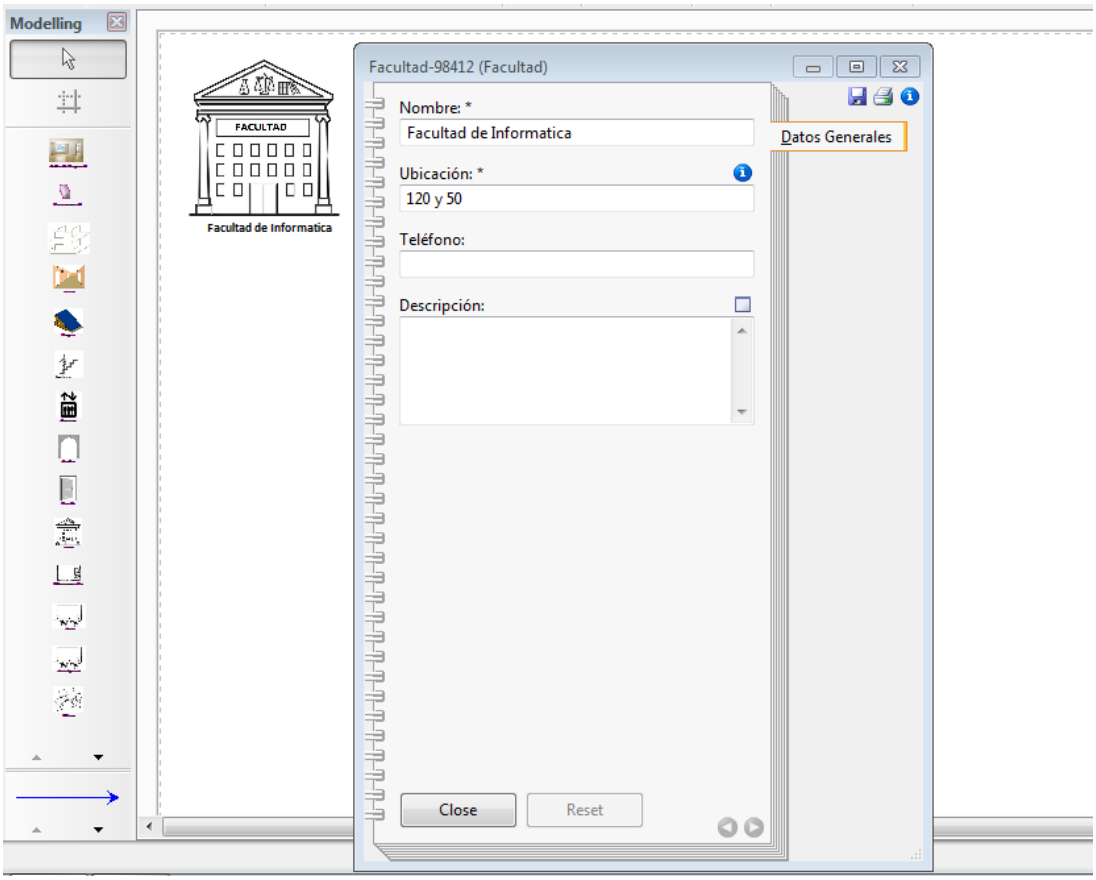


Figura 6.32 - *Definición válida de una clase con sus atributos.*

Existe otra forma de cargar los valores de atributos de una manera más dinámica para el usuario. Luego de agregar las clases que se desean configurar, se puede acceder al "modo tabular" haciendo click en el botón  de la barra de herramientas ó desde el menú *"View >> Table"*.

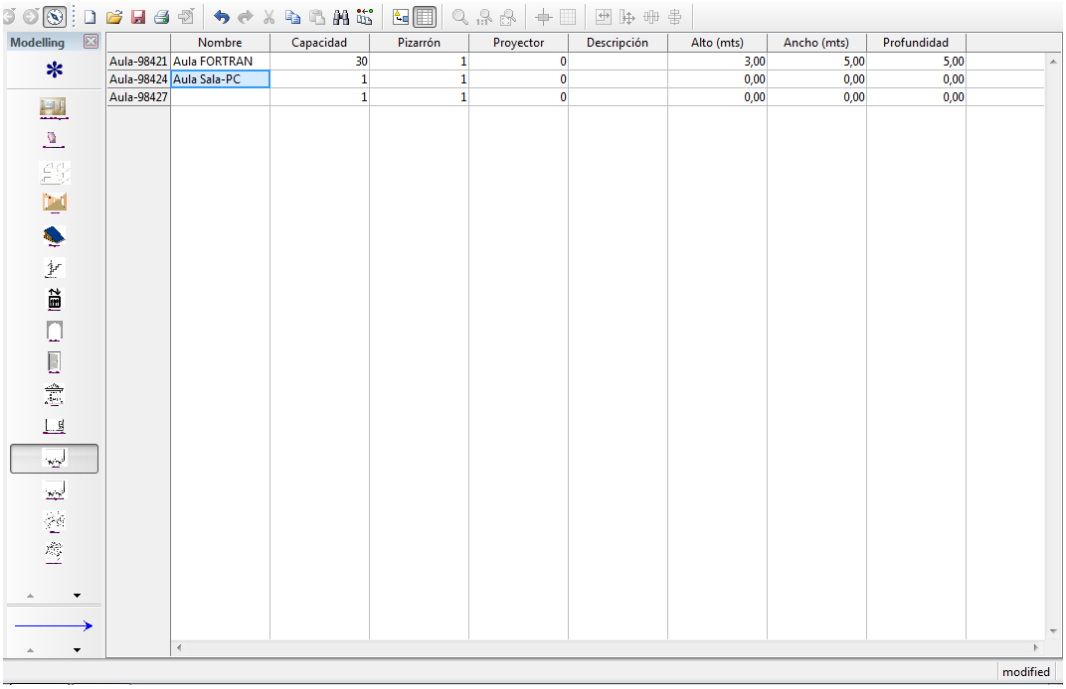


Figura 6.33 - *Definición de atributos - Modo tabular.*

Desde esta vista se debe seleccionar en el panel izquierdo *"Modelling"* la clase que se quiere configurar, en el ejemplo de la Figura 6.33 se seleccionó la clase *"Aula"*. Una vez seleccionada, en la grilla de la derecha aparecerán listadas de a una por línea todas las instancias definidas para dicha clase, y las columnas representan los atributos correspondientes a la clase.

Para editar los valores de los atributos desde esta vista, o bien se debe hacer click sobre la celda correspondiente o se puede navegar por la tabla usando las flechas del teclado hasta llegar a la celda correspondiente y presionando la tecla <Enter> luego de setear el valor deseado. Este mecanismo permite al usuario agilizar la carga de los valores de los atributos de las clases. Es importante destacar que se respetan las mismas validaciones que se ejecutan durante el modelado en el "modo dibujo".

6.5.2.3 Instancia generada

A partir del conocimiento obtenido a lo largo de la tesina acerca de los conceptos, la manera en que ellos se relacionan, y las características de las facultades (espacio indoor seleccionado para mostrar la validez del metamodelo abstracto *"Espacio Indoor"*), y en complemento con lo expuesto en las secciones anteriores acerca de cómo modelar en la herramienta "Modelador de facultades", estamos en condiciones de instanciar la sección seleccionada de la Facultad de Informática de la UNLP para mostrar la validez del metamodelo concreto *"Facultad"*.

En la Figura 6.34 se muestra el modelo implementado con la herramienta desarrollada. Se puede ver que el mismo posee todos los conceptos estudiados para una facultad y que cumple con todas las restricciones definidas para el metamodelo concreto.

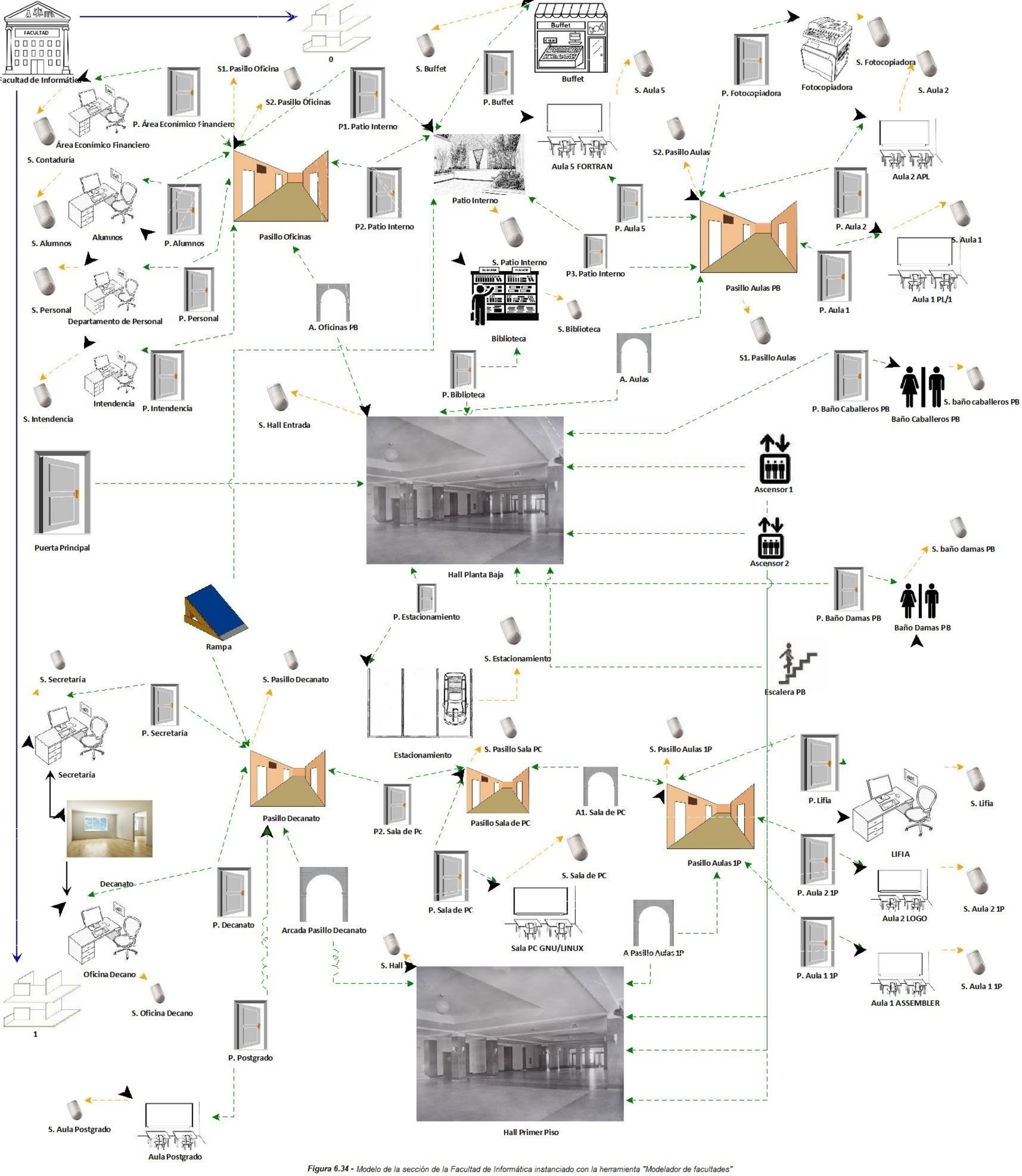


Figura 6.34 - Modelo de la sección de la Facultad de Informática instanciado con la herramienta "Modelador de facultades"

6.5.2.4 Guardado y exportación del modelo

Como se mencionó durante este capítulo, otras de las funcionalidades que ofrece la herramienta son el guardado y la exportación del modelo instanciado. Ambas son similares debido que realizan los mismos chequeos al momento de ejecutarlas; sin embargo se diferencian en que un modelo que contiene errores, tales como ausencia de relaciones obligatorias ó de valores en atributos obligatorios, entre otros, puede ser guardado para en un futuro ser corregido, pero no es posible exportarlo.

Para guardar un modelo se debe hacer click sobre el botón  o desde el menú *"Model >> Save"*, en ese momento se validan atributos obligatorios, relaciones obligatorias y que la cantidad de instancias de cada clase sea igual o mayor al mínimo permitido. Si el modelo contiene errores los mismos se mostrarán como aparece en la Figura 6.35. Haciendo click en el botón *"Yes"* se cerrará dicha ventana y a continuación se mostrará de la misma manera el siguiente error, si es que el modelo contiene otros, en caso contrario, si se presiona el botón *"No"* o el modelo no contiene más errores, se mostrará una ventana como la de la Figura 6.36, indicando al usuario que el modelo contiene errores, si de todas maneras desea continuar con el guardado del mismo.

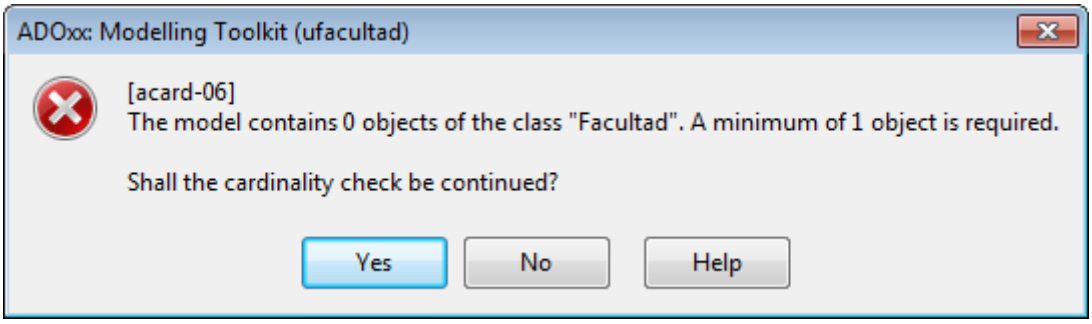


Figura 6.35 - *Ejemplo de error generado al guardar un modelo incorrecto.*

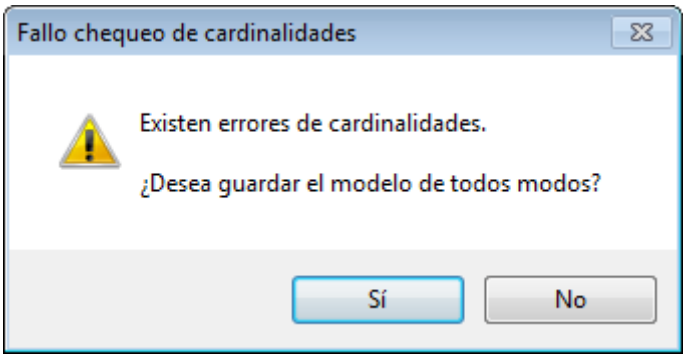


Figura 6.36 - *Ventana para finalizar el guardado del modelo.*

A continuación se listarán las restricciones OCL definidas para el metamodelo *"Facultad"* que se validan durante el proceso de guardado y exportación del modelo junto con el error que arrojan, a modo de presentar un ejemplo de cada uno de ellos. Es importante destacar que además se validan las restricciones implícitas en el metamodelo dadas por las cardinalidades de las asociaciones.

- **Atributos obligatorios sin valor**
  - Para todas las clases el atributo nombre debe ser obligatorio.
  - El atributo ubicación de un EspacioIndoor debe ser obligatorio
  - El atributo posición de SensorDePosicionamiento debe ser obligatorio
  - El atributo posición de Acceso debe ser obligatorio

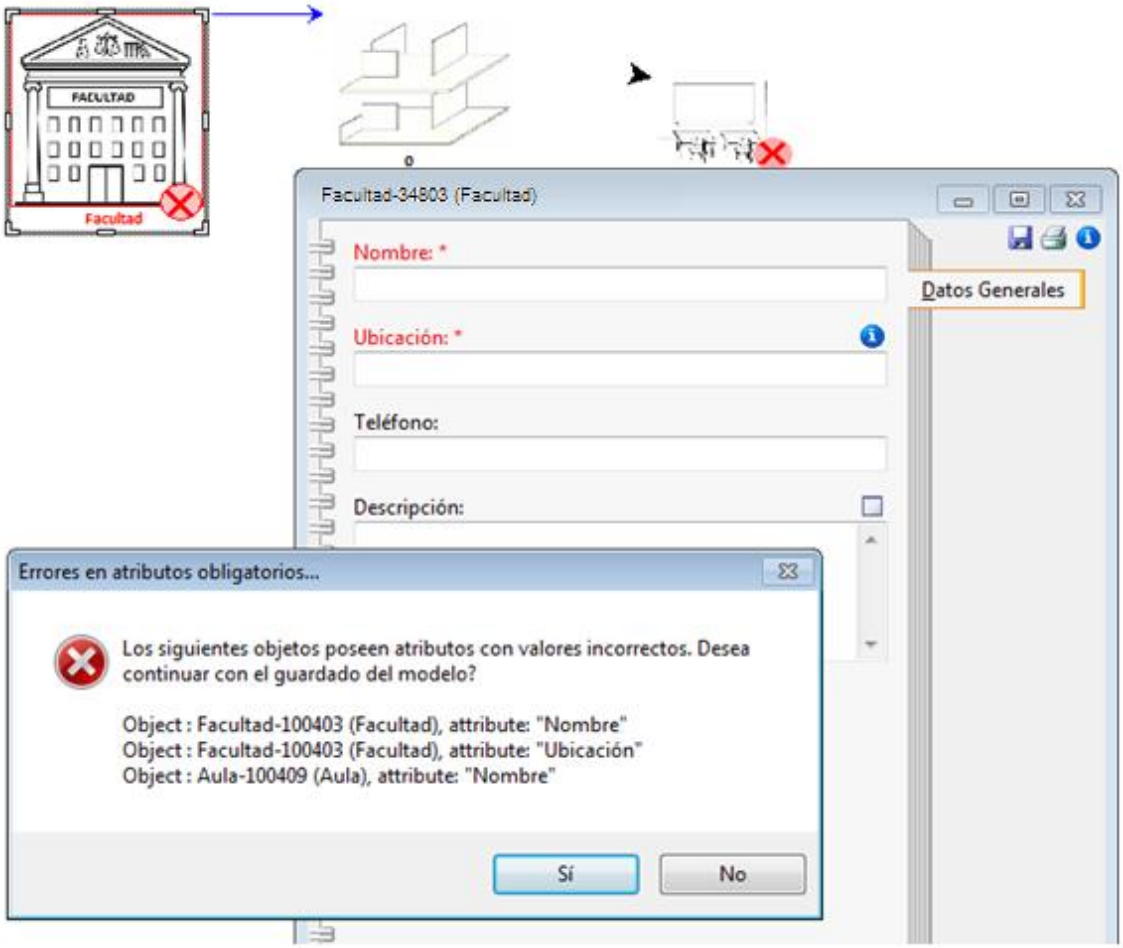


Figura 6.37 - *Guardar modelo: Error en atributos obligatorios.*

- **Relaciones obligatorias**
  - Cada Ambiente debe tener al menos un Acceso
  - Cada Ambiente debe tener al menos un SensorDePosicionamiento

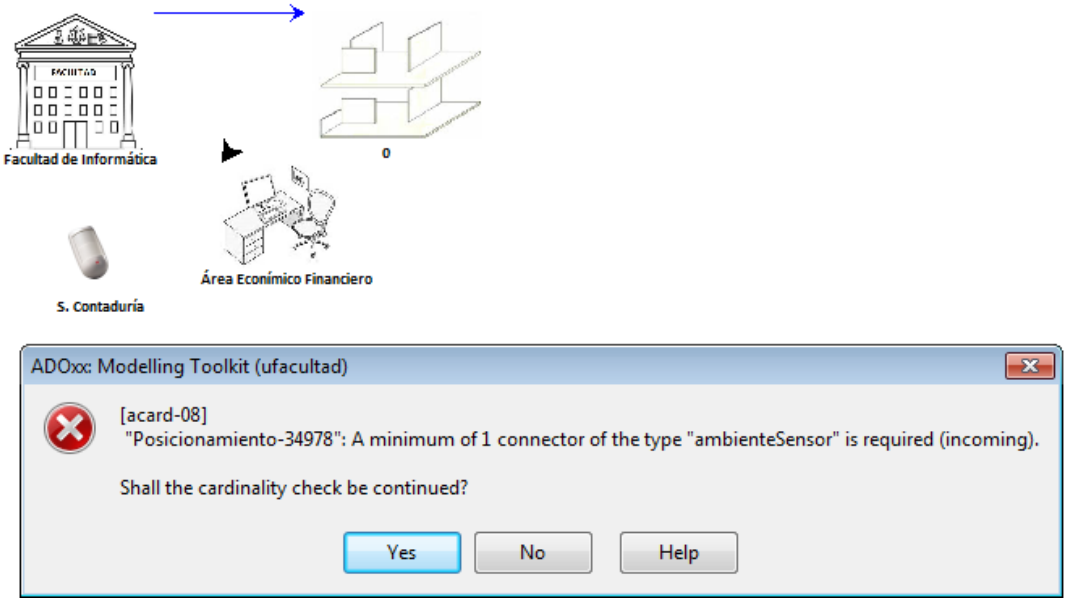


Figura 6.38 - *Guardar modelo: Error en relaciones obligatorias.*

- **Número mínimo de instancias**
  - La Facultad debe tener al menos un Aula
  - La Facultad debe tener al menos una Biblioteca

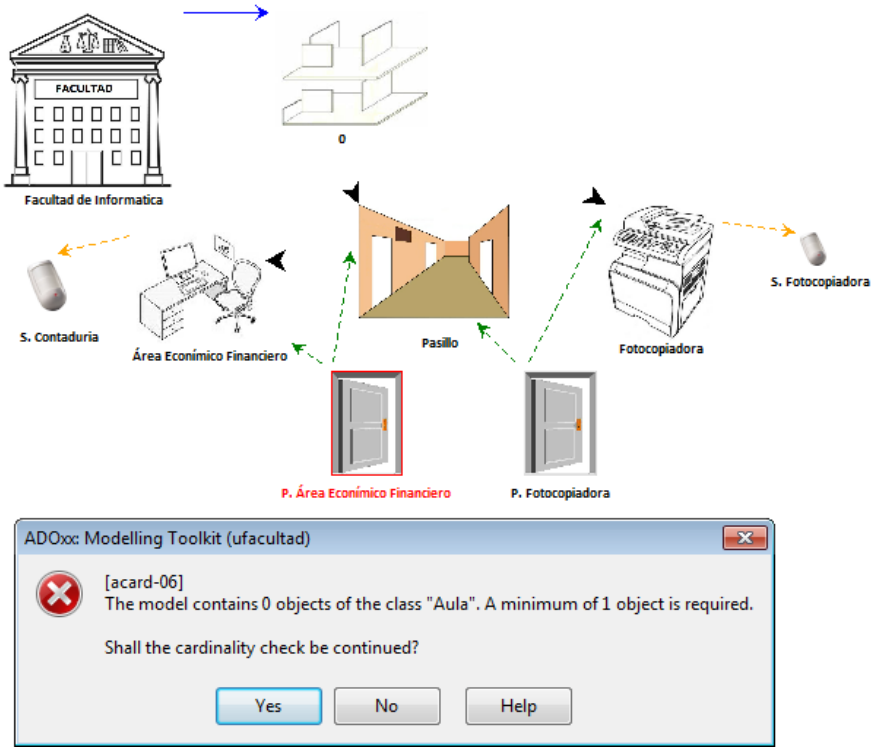


Figura 6.39 - *Guardar modelo: Error en número mínimo de instancias.*

Si bien la herramienta "Modelador de facultades" abarca un gran porcentaje del comportamiento del metamodelo concreto *"Facultad"*, la misma posee ciertas limitaciones en cuanto a la validación de algunas restricciones definidas en OCL; es por ello que definimos un prototipo de herramienta (ver sección 6.6) que, además de definir algoritmos para recorrer el espacio instanciado, valida las restricciones que no pudieron ser verificadas en el momento de exportar un modelo definido en "Modelador de facultades".

Para exportar un modelo debe hacerse click en el botón  y acceder desde el menú a la opción *"Model >> XML Export..."*, a continuación se mostrará una ventana como la de la Figura 6.40 en la cual se debe seleccionar el modelo que se va a exportar y el directorio en el cuál se desea guardar el archivo generado.

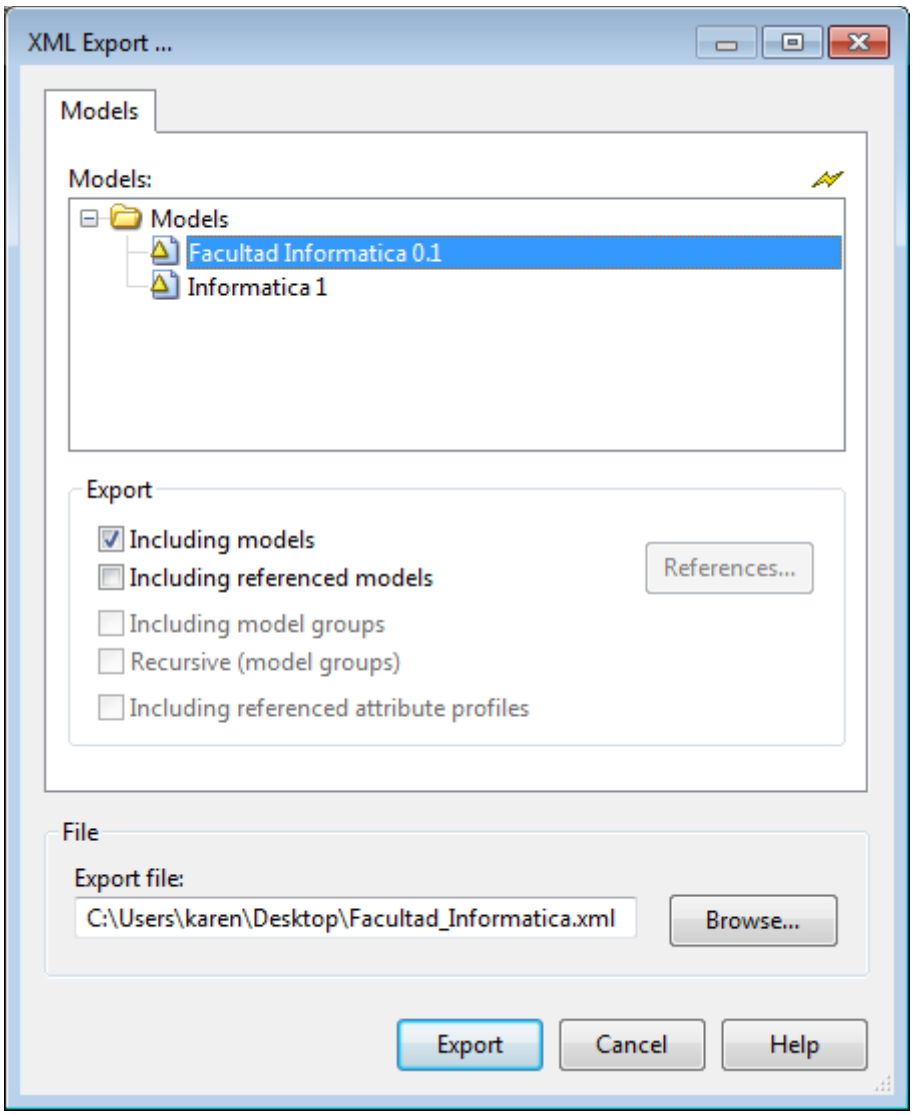


Figura 6.40 - *Exportar modelo.*

Como se ha mencionado anteriormente, durante el proceso de exportación se realizan las mismas validaciones listadas para el proceso de guardado. De la misma forma, si el modelo contiene errores se mostrará una ventana como la de la Figura

6.35 y se procederá de la misma manera. Al finalizar el chequeo de restricciones o al cancelarlo, se mostrará al usuario una notificación de alerta como el de la Figura 6.41 indicando que el modelo no ha sido exportado; en caso contrario la alerta será como la que se muestra en la Figura 6.42 y el archivo XML será generado en el directorio indicado.

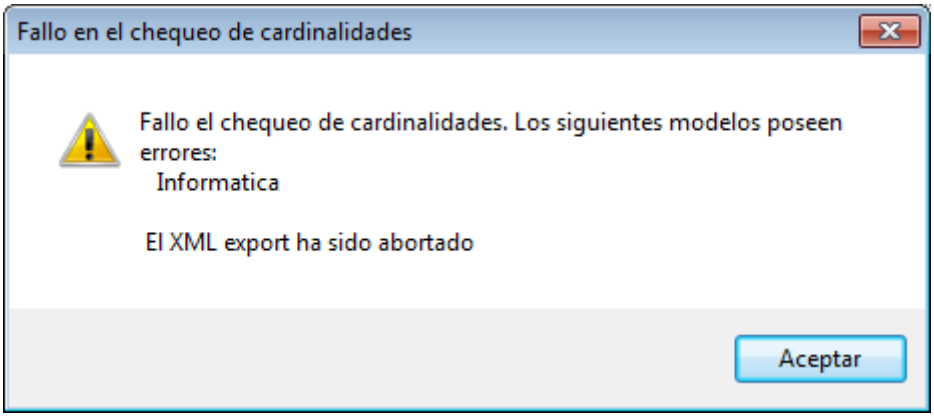


Figura 6.41 - *Error al exportar modelo.*

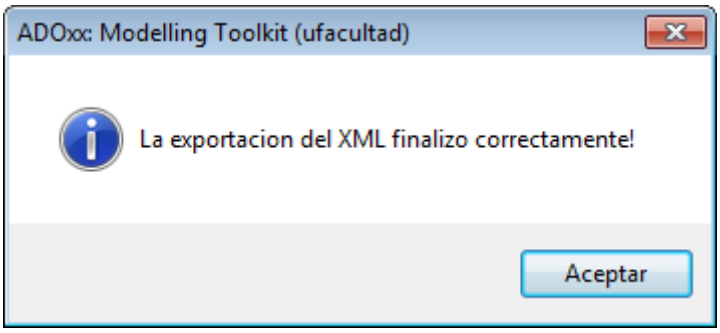


Figura 6.42 - *Modelo exportado correctamente.*

Haciendo un análisis del archivo XML generado, podemos describir tres secciones en el que lo podemos desglosar: por un lado, entre los tags `<modelattributes> </modelattributes>` se encuentran todos los atributos referentes al modelo tales como el *"nombre"*, *"autor"*, *"versión"*, *"fecha de creación"*, entre otros; por otro lado entre los tags `<instance> </instance>`, se encuentra definida cada instancia de clase agregada al modelo tales como *"Facultad"*, *"Aula"*, *"Puerta"*; y finalmente bajo los tags `<connector> </connector>` se define cada instancia de relación perteneciente al modelo, como por ejemplo *"accesoAmbiente"*, *"espacioNivel"*, etc. Estas tres secciones mencionadas se encuentras incluidas entre los tags `<adoxml> </adoxml>`. A continuación se muestra el esqueleto del modelo de la sección de la Facultad de Informática instanciado.

```
<?xml version="1.1" encoding="UTF-8"?>
<adoxml xmlns="@boc-eu.com/boc-is/adonis.model.document;1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="@boc-eu.com/boc-is/adonis.model.document;1 adoxmlmodel.xsd" version="" date="31.08.2015" time="15:59:53" database="test" username="pau" adoversion="4.0">
    <models>
        <model id="34988" name="Facultad Informatica" version="0.1" modeltype="Facultad" libtype="bp" applib="Libreria Facultad Dinamica">
            <modelattributes>
                <attribute name="Version number" type="STRING">0.1</attribute>
                <attribute name="Author" type="STRING">paula</attribute>
                <attribute name="Creation date" type="STRING">17.03.2015, 00:43</attribute>
                ...
            </modelattributes>
            <instance id="obj.34989" class="Facultad" name="Facultad-34803">
                ...
                <attribute name="Nombre" type="STRING">Facultad de Informática</attribute>
                <attribute name="Ubicación" type="STRING">50 y 120</attribute>
                ...
            </instance>
            <instance id="obj.34992" class="Nivel" name="Nivel-34806">
                ...
                <attribute name="Número" type="INTEGER">0</attribute>
                ...
            </instance>
            ...
            <connector id="con.35133" class="espacioNivel">
                <from instance="Facultad-34803" class="Facultad"></from>
                <to instance="Nivel-34806" class="Nivel"></to>
                ...
            </connector>
            ...
        </model>
    </models>
</adoxml>
```

Como ejemplo tomamos un conjunto de clases con sus respectivas relaciones del modelo instanciado para mostrar su representación en XML y analizar cómo se relacionan estas clases entre sí.

Ejemplo 1

En el primer ejemplo se muestra la instancia de la clase *"Facultad"* relacionada a la clase *"Nivel"* por medio de la relación *"nivelAmbiente"*. A continuación se muestran los xml respectivos a cada instancia:

Clases

```
<instance id="obj.34989" class="Facultad" name="Facultad-34803">
     <attribute name="Position" type="STRING">NODE x:0cm y:0cm w:2.25cm
        h:2.25cm index:1
     </attribute>
     <attribute name="External tool coupling" type="STRING"></attribute>
     <attribute name="Nombre" type="STRING">Facultad de Informática
     </attribute>
     <attribute name="Teléfono" type="STRING"> 0221 427-7270 / 427-7271
        Fax: 423-0124
     </attribute>
     <attribute name="Ubicación" type="STRING">50 y 120</attribute>
     <attribute name="Descripción" type="LONGSTRING">La Facultad de
        Informática de la Universidad Nacional de la Plata cuenta con 15
        años de prestigio académico que se visualiza en la formación de
        excelencia de sus docentes, investigadores y graduados.
     </attribute>
</instance>
```

```
<instance id="obj.34992" class="Nivel" name="Nivel-34806">
     <attribute name="Position" type="STRING">NODE x:9cm y:0cm w:2.1cm
        h:1.56cm index:2
     </attribute>
     <attribute name="External tool coupling" type="STRING"></attribute>
     <attribute name="Número" type="INTEGER">0</attribute>
     <attribute name="Descripción" type="LONGSTRING">Planta Baja</attribute>
</instance>
```

Como se mencionó anteriormente las instancias de clase son definidas entre los tags `<instance> </instance>` a los cuales se les agrega tres propiedades: `id` que es un valor único interno utilizado por ADOxx, `class` que hace referencia a la clase a la cual pertenece la instancia, y `name` que es la propiedad que se utiliza como valor clave de la instancia; el mismo es generado en el momento de instanciar cada clase, como puede verse en la Figura 6.37, dicho valor aparece en la parte superior de la ventana de atributos.

Entre los tags `<attribute> </attribute>` se define cada atributo que contiene la clase, con las propiedades `name`, que indica el nombre del atributo y `type` que indica de qué tipo es. Los primeros dos son definidos y utilizados por ADOxx; el primero indica la ubicación de la clase dentro del área de dibujo, y el segundo se utiliza para indicar el acoplamiento con herramientas externas. A continuación se lista cada atributo de la clase junto con el valor, si es que posee.

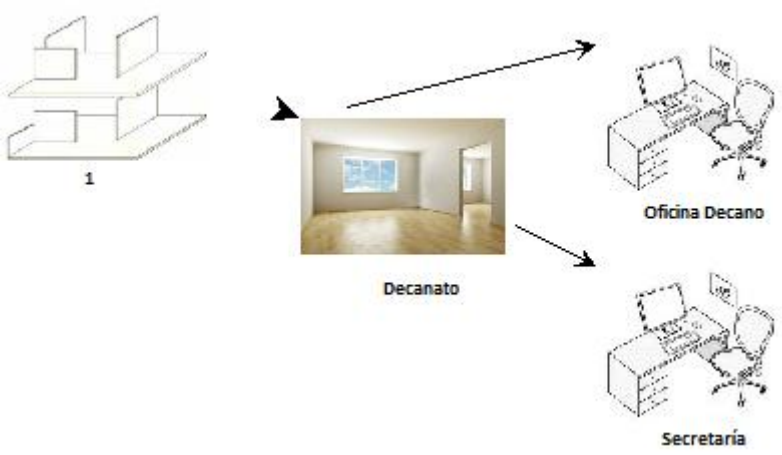
Relación

```
<connector id="con.35133" class="espacioNivel">
     <from instance="Facultad-34803" class="Facultad"></from>
     <to instance="Nivel-34806" class="Nivel"></to>
     <attribute name="Positions" type="STING">EDGE 1 x1:8cm y1:.5cm index:3
     </attribute>
     <attribute name="GraphRep" type="STRING">GRAPHREP&#10;SHADOW off&#10;
       PEN style:solid w:0.05cm&#10;EDGE&#10;&#10;START&#10;&#10;MIDDLE&#10;
       &#10;AVAL orientation: &quot;Orientation&quot;&#10;IF (orientation =
       &quot;vertical&quot;)&#10;{&#10;FONT &quot;Arial&quot; h:8pt line-
       orientation:90&#10;TEXT &quot;espacioNivel&quot; x:0.00cm y:0.00cm
       w:c:2.00cm h:t&#10;}&#10;ELSE&#10;{&#10;FONT &quot;Arial&quot; h:8pt
       line-orientation:0&#10;TEXT &quot;espacioNivel&quot; x:0.00cm
       y:0.00cm w:c:2.00cm h:t&#10;}&#10;&#10;END&#10;FILL
       color:black&#10;POLYGON 4 x1:0.0cm y1:0cm x2:-0.2cm y2:-0.1cm x3:-
       0.1cm y3:0cm x4:-0.2cm y4:0.1cm
     </attribute>
     <attribute name="AttrRep" type="STRING"></attribute>
</connector>
```

Las relaciones incluidas en el modelo son representadas entre los tags `<connector> </connector>` a los cuales se le agregan dos propiedades: `id`, al igual que para las clases es un valor único utilizado por ADOxx, y `class` que hace referencia a la relación a la cual pertenece la instancia. Dentro de esta definición se incluyen los tags: `<from> </from>` que hace referencia a la clase origen de la relación indicando en la propiedad `instance` el `name` de la clase origen y en la propiedad `class` la clase de la instancia origen; el tag `<to> </to>` que hace

referencia a la clase destino de la relación, con las mismas propiedades que el tag <from> </from>, y finalmente entre los tags <attribute> </attribute> se define cada atributo del conector, todos ellos son utilizados por ADOxx.

Ejemplo 2



Clases

```
<instance id="obj.37812" class="AmbienteCompuesto" name="AmbienteCompuesto-
37812">
     <attribute name="Position" type="STRING">NODE x:2cm y:24cm w:2.55cm
       h:1.7cm index:194
     </attribute>
     <attribute name="External tool coupling" type="STRING"></attribute>
     <attribute name="Nombre" type="STRING">Decanato</attribute>
     <attribute name="Profundidad" type="DOUBLE">6</attribute>
     <attribute name="Descripción" type="LONGSTRING"></attribute>
     <attribute name="Alto (mts)" type="DOUBLE">3</attribute>
     <attribute name="Ancho (mts)" type="DOUBLE">8</attribute>
</instance>
```

```
<instance id="obj.37821" class="Oficina" name="Oficina-37821">
     <attribute name="Position" type="STRING">NODE x:2cm y:27cm w:2.29cm
       h:1.98cm index:196
     </attribute>
     <attribute name="External tool coupling" type="STRING"></attribute>
     <attribute name="Nombre" type="STRING">Oficina Decano</attribute>
     <attribute name="Profundidad" type="DOUBLE">6</attribute>
     <attribute name="Descripción" type="LONGSTRING"></attribute>
     <attribute name="Alto (mts)" type="DOUBLE">3</attribute>
     <attribute name="Ancho (mts)" type="DOUBLE">4</attribute>
     <attribute name="Atención por ventanilla" type="INTEGER">1</attribute>
     <attribute name="Horario de atención" type="STRING">Lunes a Viernes de
       8hs a 14hs
     </attribute>
</instance>
```

```
<instance id="obj.37824" class="Oficina" name="Oficina-37824">
     <attribute name="Position" type="STRING">NODE x:1cm y:21cm w:2.29cm
       h:1.98cm index:197
     </attribute>
     <attribute name="External tool coupling" type="STRING"></attribute>
     <attribute name="Nombre" type="STRING">Secretaría</attribute>
     <attribute name="Profundidad" type="DOUBLE">6</attribute>
     <attribute name="Descripción" type="LONGSTRING"></attribute>
     <attribute name="Alto (mts)" type="DOUBLE">3</attribute>
     <attribute name="Ancho (mts)" type="DOUBLE">4</attribute>
     <attribute name="Atención por ventanilla" type="INTEGER">1</attribute>
     <attribute name="Horario de atención" type="STRING">Lunes a Viernes de
       8hs a 14hs
     </attribute>
</instance>
```

```
<instance id="obj.35207" class="Nivel" name="Nivel-35207">
     <attribute name="Position" type="STRING">NODE x:0cm y:29cm w:2.4cm
       h:1.79cm index:102
     </attribute>
     <attribute name="External tool coupling" type="STRING"></attribute>
     <attribute name="Número" type="INTEGER">1</attribute>
     <attribute name="Descripción" type="LONGSTRING">Nivel Uno</attribute>
</instance>
```

Relaciones

```
<connector id="con.37827" class="compuestoPor ">
     <from instance="AmbienteCompuesto-37812" class="AmbienteCompuesto "></from>
     <to instance="Oficina-37821" class="Oficina"></to>
     <attribute name="Positions" type="STING">EDGE 0 index:198</attribute>
     <attribute name="GraphRep" type="STRING">GRAPHREP&#10;SHADOW off&#10;
       PEN style:solid w:0.05cm&#10;EDGE&#10;&#10;START&#10;&#10;MIDDLE&#10;&#10;
       AVAL orientation: &quot;Orientation&quot;&#10;IF (orientation = &quot;
       vertical&quot;)&#10;{&#10;FONT &quot;Arial&quot; h:8pt line-
       orientation:90&#10;}&#10;ELSE&#10;{&#10;FONT &quot;Arial&quot;
       h:8pt line-orientation:0&#10;}&#10;&#10;END&#10;FILL color:black&#10;
       POLYGON 4 x1:0.0cm y1:0cm x2:-0.2cm y2:-0.1cm x3:-0.1cm y3:0cm x4:-0.2cm
       y4:0.1cm
     </attribute>
     <attribute name="AttrRep" type="STRING"></attribute>
</connector>
```

```
<connector id="con.37828" class="compuestoPor">
     <from instance="AmbienteCompuesto-37812" class="AmbienteCompuesto"></from>
     <to instance="Oficina-37824" class="Oficina"></to>
     <attribute name="Positions" type="STING">EDGE 1 x1:1.5cm y1:24.5cm index:199
     </attribute>
     <attribute name="GraphRep" type="STRING">GRAPHREP&#10;SHADOW off&#10;
       PEN style:solid w:0.05cm&#10;EDGE&#10;&#10;START&#10;&#10;MIDDLE&#10;&#10;
       AVAL orientation: &quot;Orientation&quot;&#10;IF (orientation = &quot;
       vertical&quot;)&#10;{&#10;FONT &quot;Arial&quot; h:8pt line-
       orientation:90&#10;}&#10;ELSE&#10;{&#10;FONT &quot;Arial&quot;
       h:8pt line-orientation:0&#10;}&#10;&#10;END&#10;FILL color:black&#10;
       POLYGON 4 x1:0.0cm y1:0cm x2:-0.2cm y2:-0.1cm x3:-0.1cm y3:0cm x4:-0.2cm
       y4:0.1cm
     </attribute>
     <attribute name="AttrRep" type="STRING"></attribute>
</connector>
```

```
<connector id="con. 37831" class="nivelAmbiente">
     <from instance="Nivel-35207" class="Nivel"></from>
     <to instance="AmbienteCompuesto-37812" class="AmbienteCompuesto"></to>
     <attribute name="Positions" type="STING">EDGE 0 index:202</attribute>
     <attribute name="GraphRep" type="STRING">GRAPHREP no-edge&#10;SHADOW
        off&#10;&#10;PEN style:dash w:0cm color:white&#10;EDGE&#10;&#10;END
        &#10;PEN w:0.1cm&#10;FILL color:black&#10;POLYGON 4 x1:0.05cm y1:0cm
        x2:-0.15cm y2:-0.1cm x3:-0.1cm y3:0cm x4:-0.15cm y4:0.1cm
     </attribute>
     <attribute name="AttrRep" type="STRING"></attribute>
</connector>
```

6.5.2.5 Consultas sobre el modelo

Como se ha mencionado al inicio del capítulo, la herramienta ofrece una funcionalidad para realizar consultas sobre el modelo instanciado. Para ello se debe hacer click en el botón  y luego en  , ó desde el menú *"Analysis >> Queries/Reports…"*, y seleccionar de la pantalla mostrada en la Figura 6.43 el modelo sobre el cual se desean realizar las consultas.

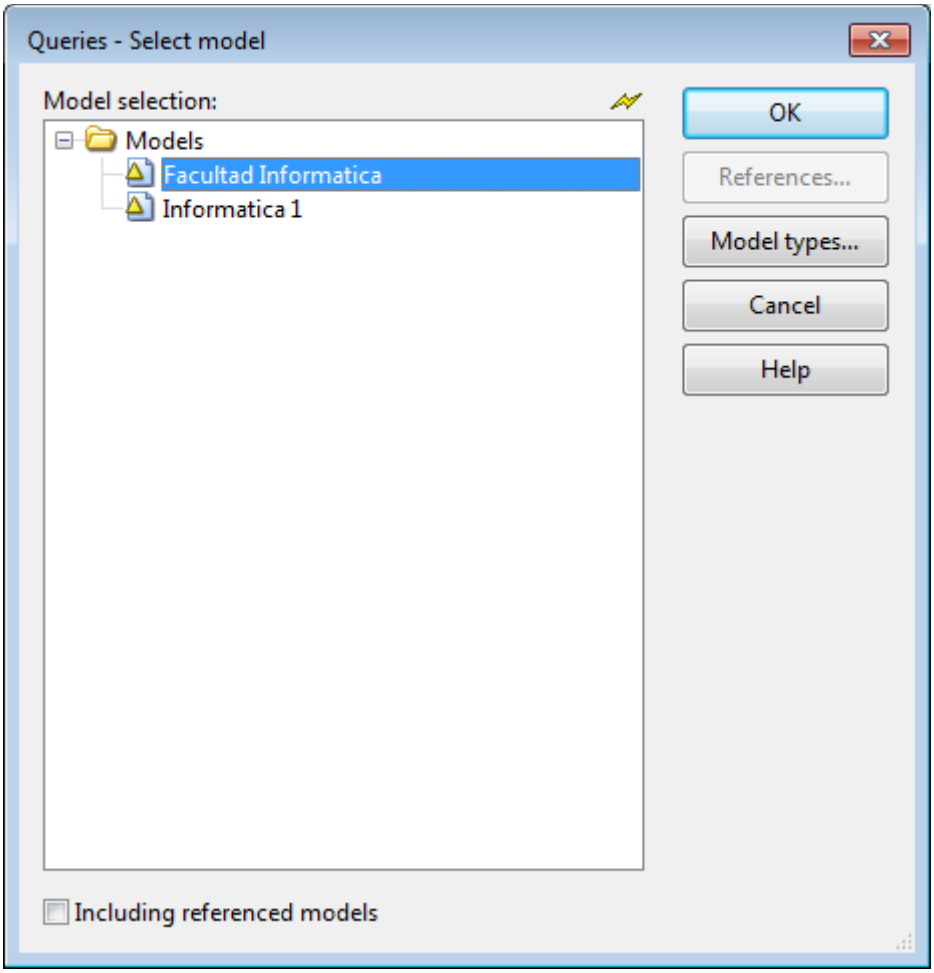


Figura 6.43 - *Selección de modelo para realizar consultas.*

A continuación aparecerá la pantalla que se muestra en la Figura 6.7, en la cual se debe definir la consulta a realizar.

El metamodelo *"Facultad"* posee como comportamiento de la clase *"Facultad"* el método "*poseeSalidasDeEmergencia()"* (heredado de la clase *"EspacioIndoor"*) el cual es resuelto mediante esta funcionalidad. En la Figura 6.44 se muestra la manera de configurar la consulta que responde al método mencionado.

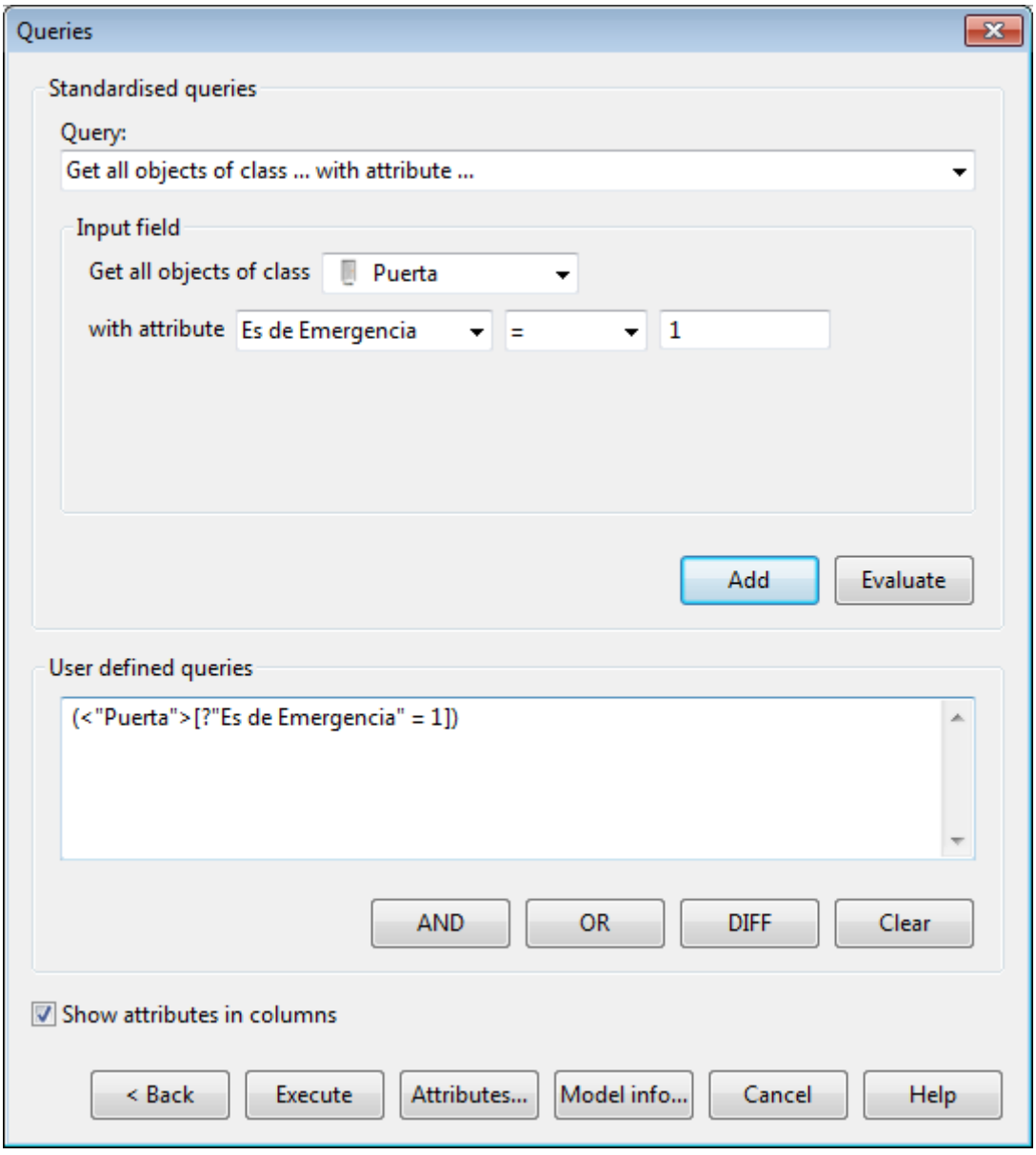


Figura 6.44 - *"Facultad::poseeSalidasDeEmergencia()".*

Luego de hacer click en el botón *"Execute"* se cerrará la ventana mostrada y, en el caso de que existan resultados, los mismos se mostrarán en la parte inferior de la pantalla principal, tal como en la Figura 6.45. Para el método seleccionado como resultado se listarán todos los *"Accesos"* *"Puerta"* que tengan seteado en true el valor del atributo *"esDeEmergencia"*.

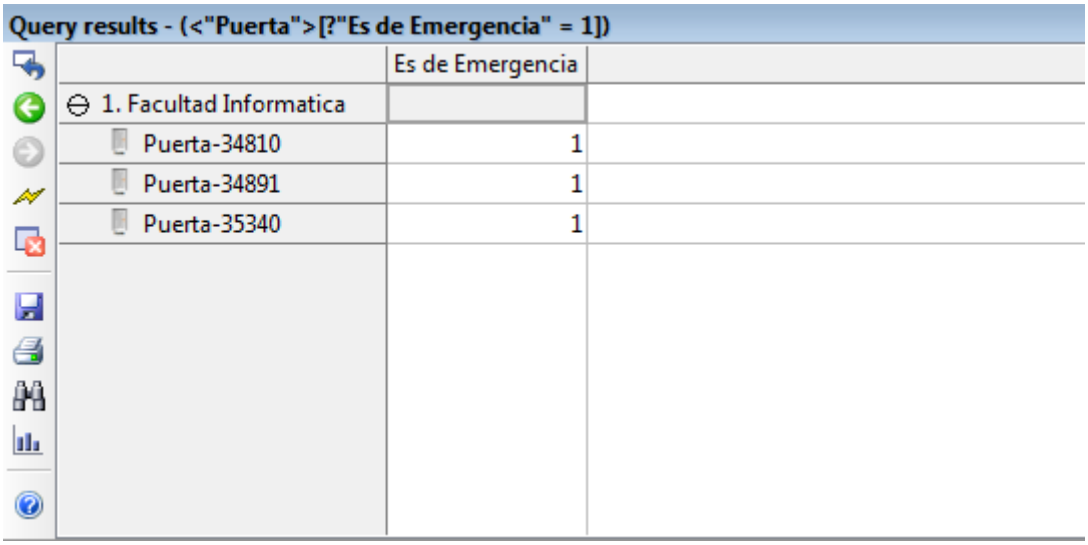


Figura 6.45 - *Resultado de ejecutar el método "poseeSalidasDeEmergencia()"*

6.6 Prototipo PHP: “Buscador de Caminos”

Como se explicó durante el desarrollo de la tesina, a partir de la utilización de la herramienta de metamodelado ADOxx, logramos diseñar dos librerías de metamodelado;

- la primera que representa al metamodelo abstracto *"Espacio Indoor",* que incluye todos los conceptos comunes de los espacio indoor en general y las relaciones que existen entre ellos y,

- la segunda que representa al metamodelo concreto *"Facultad"*, la cual hereda todo el comportamiento del metamodelo padre y lo refina agregando las propiedades particulares que poseen todas las facultades en general.

Una vez diseñado y corregido el metamodelo concreto, desarrollamos una instancia, creando un modelo exclusivo que representa una sección de la Facultad de Informática de la Universidad Nacional de La Plata; el cual posee todas las propiedades, conceptos y atributos de ella. Luego mostramos que "Modelador de Facultades" permite exportarlo creando un archivo .xml que contiene toda la información del modelo creado.

Con el objetivo de demostrar la utilidad y la importancia del archivo .xml generado y la validez del metamodelo concreto propuesto, creamos un prototipo PHP al que llamamos “Buscador de Caminos", que permite capturar la información contenida en cualquier xml generado por la herramienta, y así poder manipularla según nuestras necesidades.

Otro motivo que nos impulsó al desarrollo de este prototipo, fue las limitaciones que la herramienta ADOxx nos presentó al momento de querer implementar las restricciones OCL. Varias de ellas no pudieron ser resueltas en “Modelador de Facultades” debido a que ADOxx, al momento de la realización de la tesina, no proveía el soporte necesario para, por ejemplo, acceder a los valores de los atributos

de las clases o transitar entre las relaciones de las mismas en el momento de la exportación, funcionalidades requeridas para poder validar algunas de las restricciones definidas para el metamodelo “Facultad”. Es por ello que decidimos incluir las validaciones que no pudieron ser resueltas en la herramienta desarrollada en ADOxx dentro del prototipo para, efectivamente, poder validar que el modelo instanciado cumpla con el metamodelo “Facultad”.

A continuación se presenta la funcionalidad del prototipo:

1) En la página de inicio del prototipo se solicita al usuario que adjunte el archivo .xml generado con la herramienta "Modelador de facultades".

2) Una vez adjunto el archivo, se valida que el modelo instanciado cumpla con las restricciones OCL que no se pudieron verificar en "Modelador de facultades". Ellas son:

   - Un EspacioIndoor debe contener al menos un Acceso cuyo atributo conectaExterior este seteado con el valor true
   - Un Acceso de tipo EntreNivel debe conectar dos Ambientes de distinto Nivel.
   - Un Acceso de tipo EntreNivel debe conectar dos Ambientes con Niveles sucesivos.
   - Un Acceso de tipo EnNivel debe conectar dos Ambientes que se encuentren en el mismo Nivel.
   - En un mismo Nivel no pueden existir Ambientes con el mismo nombre.
   - El TipoDeAcceso Escalera no puede tener seteado el atributo esDeEmergencia en true si su atributo mecánica esta seteado en true.
   - La Facultad debe tener al menos un Baño con el atributo sexo seteado en "F".
   - La Facultad debe tener al menos un Baño con el atributo sexo seteado en "M".

   Si todo es correcto se guarda la información contenida en el archivo en una base de datos MySQL creada para este fin. Si alguna validación falla, o si la transacción no finalizó correctamente se informa al usuario con un aviso que aparece en pantalla donde especifica el tipo de error cometido.

3) Cuando toda la información se encuentra guardada en la base de datos, se informa al usuario que se pudo procesar correctamente el archivo e, internamente se genera un grafo que contiene toda la información; donde los nodos del grafo son los accesos del modelo y las aristas los ambientes.

4) Con los datos almacenados en la base de datos, y el grafo generado en memoria podemos avanzar a la siguiente pantalla en donde se le solicita al usuario que seleccione el acceso origen y el acceso destino al que quiere llegar, y le ofrecen dos opciones para buscar un camino posible: un camino “sin condiciones”, o un camino que sea “apto para discapacitados”, o bien puede

elegir la opción de encontrar un camino que le permita llegar al exterior desde cualquier acceso.

5) Una vez seleccionados los datos del camino que se quiere encontrar, se ejecuta un algoritmo basado en grafos, necesario para obtener un camino que cumpla con las condiciones. Si se encuentra un camino con esas características, se muestra al usuario el camino obtenido en forma de grafo, especificando las distancias que existen entre cada nodo; estos datos son obtenidos de la base de datos. Si no se encuentra un camino, se informa al usuario con un aviso, y se le permite volver a la pantalla anterior para realizar nuevamente la selección de otro camino.

6.6.1 Ejemplos

En esta sección se presentan algunos ejemplos de las consultas que pueden realizarse con "Buscador de caminos", mostrando los resultados obtenidos luego de cargar el archivo generado con "Modelador de facultades" para una sección de la Facultad de Informática.

- *Facultad::llegarAlExterior()*
  En este ejemplo se muestra cuál es el camino que debe realizarse para llegar al exterior desde la puerta del aula 1 del primer piso. Para ello, en el paso 4 mencionado en la sección anterior, se eligió la opción *"Llegar al exterior"* y se seleccionó el acceso *"P. Aula 1 1P"*, y el resultado es el siguiente:

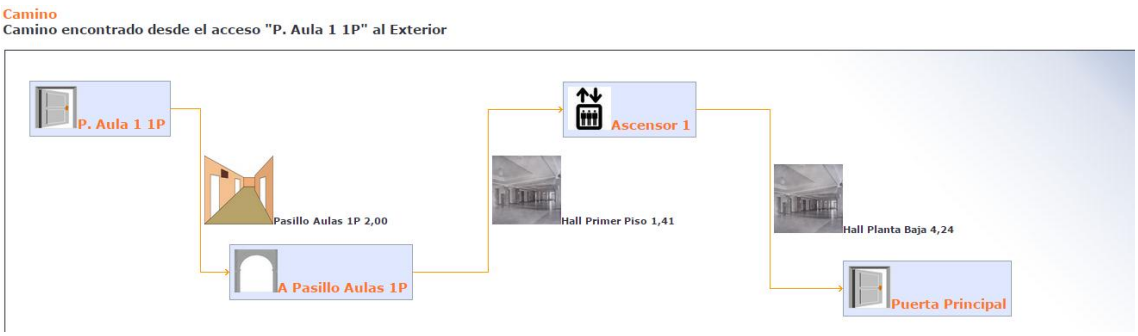


Figura 6.46 - *Prototipo PHP - "llegarAlExterior()"*

- *Facultad::caminoDesdeHastaAptoDiscapacitados()*
  Aquí se muestra un camino apto para discapacitados que nos lleva desde la puerta del buffet, ubicado en la planta baja, hasta la puerta de la sala de PC ubicada en el primer piso. Es importante destacar que este método retorna un camino en donde todos los accesos a los ambientes son aptos para discapacitados; por ejemplo se indica que hay que utilizar la rampa en lugar de la escalera.

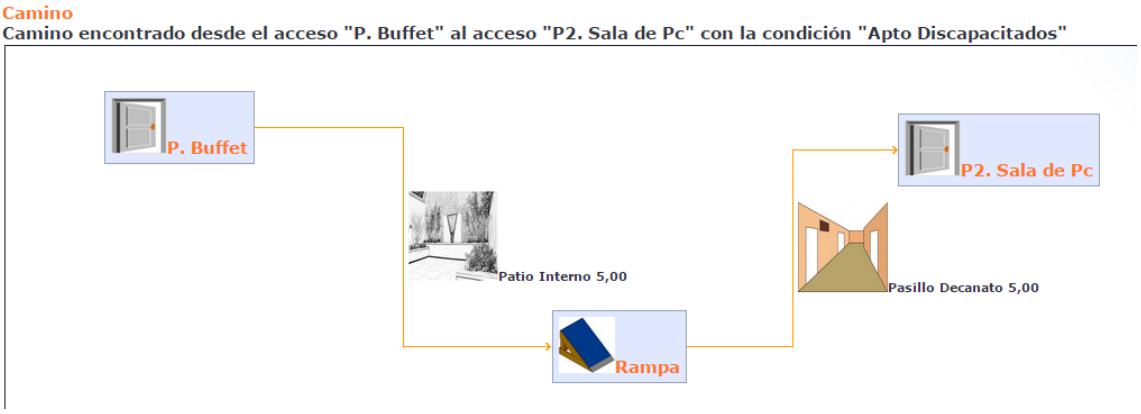


Figura 6.47 - *Prototipo PHP - " caminoDesdeHastaAptoDiscapacitados()"*

- *Facultad::caminoDesdeHastaSinCondicion()*
  Como último ejemplo se muestra un camino que nos lleva desde la puerta de entrada, ubicada en la planta baja, hasta la puerta del laboratorio LIFIA ubicada en el primer piso.

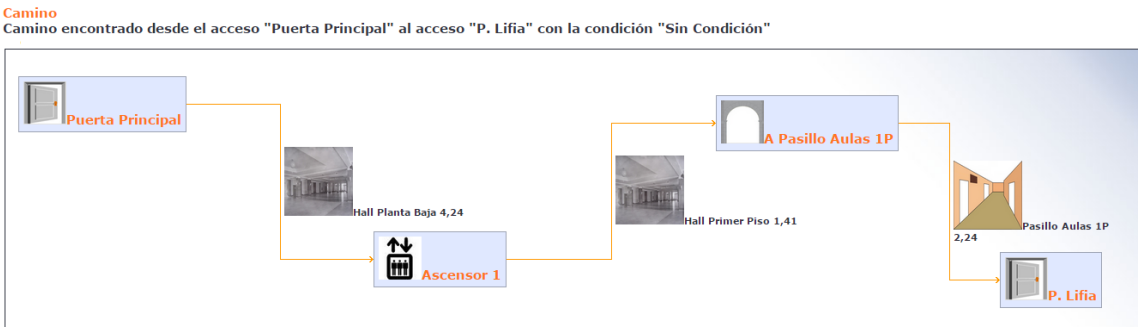


Figura 6.48 - *Prototipo PHP - " caminoDesdeHastaAptoSinCondicion()"*

Nota: *El código fuente del prototipo "Buscador de Caminos", el script que permite crear la base de datos y las librerías de los metamodelos "Espacio Indoor" y "Facultad" se encuentran en el CD entregado con la tesina. A su vez se incluye el archivo .xml generado con la herramienta "Modelador de Facultades" de la sección de la Facultad de Informática instanciada.*

6.7 Resumen

En este capítulo se realizó un estudio acerca de la tecnología seleccionada para realizar las implementaciones de las librerías *"espacioIndoor"* y *"facultad"*, que se corresponden con el metamodelo abstracto *"Espacio Indoor"* y concreto *"Facultad"* respectivamente, definidos en el capítulo 4. Se describen detalladamente las funcionalidades que brinda "Modelador de facultades", haciendo hincapié principalmente en las siguientes que son fundamentales para construir cualquier modelo de facultad que se desee:

- Instanciación de clases
- Instanciación de relaciones
- Definición de atributos de clases
- Generación de archivo XML para utilizar en el prototipo "Buscador de caminos"

Además se tomó como ejemplo de modelo una sección de la Facultad de Informática de la UNLP con el objetivo de validar que la herramienta desarrollada cumple con las restricciones definidas en el metamodelo *"Facultad"*, para de esta manera mostrar la validez de la misma y la capacidad que brinda para modelar cualquier espacio universitario.

Una limitación que presenta "Modelador de facultades" es que existen ciertas restricciones definidas para el metamodelo concreto que no pueden ser validadas en dicha herramienta, es por eso que no es posible garantizar con certeza que el modelo resultante cumpla exactamente con el metamodelo *"Facultad"*. Por este motivo es que se presenta el prototipo "Buscador de caminos" que complementa el desarrollo realizado, en el cual se validan las restricciones que no pudieron ser resueltas durante la exportación de un modelo instanciado en "Modelador de facultades". Además, este prototipo rectifica la validez del metamodelo concreto propuesto debido a que se muestra que es posible recorrer el espacio instanciado a través de los diferentes ambientes que lo componen.

Capítulo 7

Conclusiones

Capítulo 7 – Conclusiones

7.1 Trabajos Relacionados

En esta sección se presenta una pequeña descripción de algunos trabajos que estudiamos a lo largo del desarrollo de la tesina, relacionados a las temáticas expuestas en este trabajo, las cuales demuestran el crecimiento del interés de la comunidad por el estudio de los espacios indoor. Entre ellos podemos mencionar los siguientes:

- **Situm Indoor Maps:** Es un software creado en el año 2015 por la empresa *SITUM: Indoor Positioning*, ubicada en España. Esta aplicación promete una localización multisensorial para interiores, y es capaz de determinar la posición de cualquier dispositivo sensorizado, como smartphones, con una alta precisión y consistencia. Permitirá al usuario conocer su posición, saber qué tiene a su alrededor, cómo puede llegar a los lugares que le interesan e, incluso, planificar rutas y ser guiado hasta un punto. El guiado de personas se puede realizar mediante la propia interfaz visual del smartphone o a través de instrucciones auditivas. Al ser una aplicación comercial, creada por una empresa privada, el diseño e implementación no es accesible al mundo de la informática, por lo tanto no pudimos acceder a ellos. Seguramente el metamodelo abstracto “Espacio Indoor” expuesto en esta tesina se podría utilizar como base del diseño de esta aplicación, para poder crear los mapas de los distintos espacios que se pueden descargar en los smartphones.

  Para más información sobre esta herramienta se puede visitar la página oficial [33].

- **Representaciones enriquecidas para la navegación indoor-outdoor en aplicaciones móviles** [4]: En este trabajo, se expone un modelo orientado a objetos que permite representar espacios enriquecidos. Definen espacio enriquecido como una representación espacial, el cual posee al menos un punto de acceso y puede tener definida una red de circulación, sobre la que se calculan caminos, y puntos de interés. En la arquitectura de cuatro niveles expuesta en el capítulo 3, podemos ver que en el nivel M1 se encuentran los modelos a los que se define como una instancia del metamodelo. Este trabajo relacionado presenta una posible solución al diseño de los espacios indoor en el nivel M1, en esta tesina se presenta una solución con un mayor nivel de abstracción, el nivel M2: metamodelo.

- **IndoorGML:** Es un estándar que especifica un modelo abstracto de datos y un esquema XML para información espacial en espacios interiores, que se centra específicamente en modelación de este tipo de espacios desde un punto de vista de la navegación. IndoorGML proporciona una descripción tecnológica neutra de los espacios navegables de un edificio que pueden ser utilizados por cualquier desarrollador de aplicaciones y proveedores de plataformas ya que es un esquema aplicación del estándar GML. Los desarrolladores de

IndoorGML anticipan que esto beneficiará a respuestas de emergencia, compradores, visitantes de aeropuertos y muchos otros ámbitos.

En lugar de centrarse en componentes de construcción como techos, paredes, IndoorGML proporciona un marco que representa espacios cerrados como celdas. La conexión entre espacios y restricciones de movimiento entre espacios (como puertas, arcadas) son representados como relaciones entre celdas.

Para más información sobre el estándar se puede visitar la página oficial [34].

7.2 Contribuciones al mundo de la Informática

- Diseñamos un metamodelo abstracto que permite representar cualquier espacio indoor que se quiera modelar. Dicho metamodelo puede ser utilizado para cualquier aplicación que se considere necesaria; el paso a paso de su generación ha sido desarrollado a lo largo de toda la tesina.

- Con el objetivo de ejemplificar una forma de utilizar el metamodelo *"Espacio Indoor*", describimos cómo generar un metamodelo concreto que hereda todo el comportamiento, atributos y restricciones del espacio indoor, al que denominamos metamodelo *"Facultad"*.

- Desarrollamos una librería utilizando la Plataforma de Desarrollo de Metamodelos ADOxx, con el fin de diseñar cualquier espacio indoor que se desee. Esta librería va a estar disponible para su uso, con código open source, con el fin de poder extenderla para algún espacio indoor en particular.

- Adicionalmente creamos la herramienta “Modelador de Facultades”, que extiende y hereda el comportamiento que brinda la librería de Espacio Indoor nombrada en el párrafo anterior que permite realizar el modelo de cualquier facultad.

- Instanciamos el metamodelo *"Facultad"*, diseñando una sección de la Facultad de Informática de la UNLP para lograr una mayor comprensión de la robustez de los metamodelos generados en el transcurso de esta tesina.

- Incorporamos la herramienta desarrollada al repositorio OMI; en dicho repositorio existen todas las herramientas que se generaron utilizando la Plataforma de Desarrollo de Metamodelos ADOxx. Para lograr esta incorporación se deben cumplimentar con ciertos requisitos, los cuales ya fueron entregados. Al momento de presentar esta tesina, nuestra herramienta se encuentra en proceso de testing por parte de los administradores del repositorio.

- Realizamos un prototipo llamado “Buscador de Caminos”, desarrollado en lenguaje de programación PHP, que permite importar un archivo XML (generado por la herramienta “Modelador de Facultades”), y realizar consultas sobre el modelo creado en ADOxx.

7.3 Conclusiones Generales

Las personas pasan la mayor parte de su tiempo en espacios indoor, trabajan, viven, van de compras o se divierten dentro de ellos. Por este motivo, estos espacios son cada vez más grandes y más complejos, logrando complicar la capacidad de las personas de recorrerlos sin perderse.

Hoy en día consultando, por ejemplo, los sitios web de algunos espacios indoor, seguramente vamos a poder obtener una imagen que nos muestre el mapa del mismo. Si logramos interpretar esa imagen vamos a poder localizar por nuestros propios medios los lugares que nos interesen visitar dentro del espacio.

Actualmente, con la tecnología existente en el mercado, se podría brindar al usuario una asistencia más completa acerca de cómo movilizarse dentro de un espacio particular. Por ejemplo, ofrecerle información sobre los puntos de interés existentes e indicarle el camino más corto para llegar a ellos, en base a su posición actual. Para poder lograr una aplicación que provea este tipo de funcionalidades, es necesario obtener información del espacio que la imagen mencionada anteriormente no provee. Es necesario definir un modelo que logre representar de alguna manera el espacio para poder obtener esta información. La finalidad de dicho modelo es permitir seleccionar la ubicación de puntos de acceso en el interior del espacio, los cuales pueden tener varios niveles y varios ambientes en cada nivel.

Por estas razones, centramos nuestro estudio en realizar un análisis detallado sobre los espacios indoor, los conceptos comunes que conforman este tipo de espacios, junto con las características que representan a cada uno y cómo estos conceptos se relacionan entre sí; con el objetivo de obtener un metamodelo que logre representarlos.

Consideramos que esta tesina ofrece un aporte empírico a la disciplina denominada "Ingeniería de Metamodelos" [16], ya que ofrece un caso de estudio de un proceso de construcción de metamodelos de manera iterativa e incremental. Este experimento muestra cómo el metamodelo puede ir mejorándose progresivamente a través de la aplicación de patrones de diseño y otras buenas prácticas de modelado.

También muestra cómo un metamodelo puede estructurarse de manera que los conceptos más abstractos puedan ser reutilizados en varios dominios concretos diferentes. Esto se logró mediante la definición de un metamodelo abstracto ("Espacio Indoor") que luego fue refinado mediante un metamodelo más concreto ("Facultad") el cual reutiliza todos los conceptos comunes. Para este fin se analizaron las formas de relacionar metamodelos, en particular el método de herencia conservativa vs. el método de extensión libre, seleccionándose el primero de ellos para su aplicación en este trabajo. Adicionalmente, se mostró cómo la incorporación de restricciones OCL a las metaclases permite la definición de reglas de buena formación de los modelos (por ejemplo, una escalera debe conectar ambientes de distinto nivel) y permite agregar conceptos semánticos propios del dominio (por ejemplo: una Facultad debe tener al menos una Biblioteca).

Además, se realizó un estudio avanzado de la *"Plataforma de desarrollo de metamodelos ADOxx"*, con el fin de poder plasmar en ella el metamodelo concreto. Con el uso de esta plataforma, creamos la herramienta “Modelador de Facultades”, la cual incorporamos al repositorio OMI para que sea accesible al mundo de la informática. Para ejemplificar y demostrar la validez del metamodelo creado, instanciamos una sección de la Facultad de Informática de la Universidad Nacional de La Plata.

Adicionalmente, se creó un prototipo desarrollado en el lenguaje de programación PHP, al que llamamos “Buscador de Caminos” que permite a importar un archivo con extensión xml generado por la herramienta desarrollada y se encarga de almacenar los datos, verificar restricciones adicionales y permite realizar ciertas consultas sobre el modelo, por ejemplo, obtener caminos, poder alcanzar la salida, etc.

7.4 Trabajos Futuros

- Realizar una investigación que permita descubrir cómo se puede llegar a obtener las coordenadas geográficas reales de los ambientes; con esta información podríamos por ejemplo, informarle al usuario la distancia real que separa un ambiente de otro, o también, la distancia total que tendría que recorrer para llegar al destino deseado.

- Investigar la manera de lograr que los sensores capten (a través de dispositivos móviles, por ejemplo) y retornen las coordenadas que indiquen la posición real del usuario; con esta información se podría obtener la ubicación específica del usuario dentro del espacio indoor, y así poder refinar los algoritmos de búsqueda de caminos.

- En esta tesina tanto los algoritmos de búsqueda de caminos como la validación de algunas restricciones OCL se encuentran implementados en el Prototipo PHP creado para éste fin (ver capitulo 6 sección 6.6). En un futuro, se deberá investigar la manera de incorporar el código de estos algoritmos y restricciones a la herramienta “Modelador de Facultades”.

- Investigar una manera de incorporar funcionalidad y accesibilidad a la búsqueda de caminos dentro de un espacio indoor, una propuesta puede ser agregar audio al resultado y logrando así una mejor interpretación del mismo.

- Realizar un desarrollo móvil de la herramienta "Modelador de Facultades", que permita a los usuarios cargar el modelo y obtener on-line los caminos que desea recorrer.

Bibliografía

Referencias Bibliográficas

1. Sultan Alamri, David Taniar, Maytham Safar - Indexing Moving Objects in Indoor Cellular Space. 2015

2. Christian S. Jensen, Hua Lu, Bin Yang - Indexing the Trajectories of Moving Objects in Symbolic Indoor Space. 2009

3. Laia Descamps-Vila, A. Pérez Navarro, Jordi Conesa - Integración de un Sistema de posicionamiento Indoor en aplicaciones SIG para dispositivo móvil. 2013 (Enlace disponible al 27/08/2015). http://www.sigte.udg.edu/jornadassiglibre2013/uploads/articulos_13/a29.pdf

4. Alejandra Lliteras, Cecilia Challiol, Catalina Mostaccio, Silvia Gordillo - Representaciones enriquecidas para la navegación indoor-outdoor en aplicaciones móviles. CACIC 2011 (Enlace disponible al 27/08/2015). http://sedici.unlp.edu.ar/handle/10915/18743

5. Miguel A. Niño, Carlos A. Cobos, Martha E. Mendoza - Unicauca Virtual: Metamodelos de Universidad Virtual y Herramientas de Soporte (2004) (Enlace disponible al 27/08/2015). http://www.ufrgs.br/niee/eventos/RIBIE/2004/comunicacao/com1041-1050.pdf

6. Gómez Palarea, Pablo, Sanchez Ramon Oscar - Herramientas de Metamodelado Microsoft DSL Tools vs MetaEdit+ (Enlace disponible al 27/08/2015). http://dis.um.es/~jmolina/Pfc/DSLvsMetaedit.pdf

7. Sitio web oficial de OMI - Open Model Initiative http://www.omilab.org/web/guest

8. Sitio web oficial de ADOxx https://www.adoxx.org/live/home

9. Ki Joune Li - Indoor space: A new notion of space. Personal and Ubiquitous Computing. Páginas: 1-3. 2008.

10. Wenjie Yuan, Markus Schneider - Supporting Continuous Range Queries in Indoor Space. 2010. (Enlace disponible 27/08/2015). http://www.cise.ufl.edu/~mschneid/Research/papers/YS10MDM.pdf

11. Lisa Walton, Michael Worboys - Indoor Spatial Theory. 2009. (Enlace disponible 27/08/2015). http://www.spatial.maine.edu/ISAmodel/documents/IST_ISA09.pdf

12. Tecnología 2.0 - IPS: Indoor Positioning System. 2011 (Enlace disponible al 23/09/2015). http://tecnologiadospuntocero.wordpress.com/2011/04/30/ips-indoor-positioning-system/

13. Christian Schmitt, Oliver Kaufmann - Indoor Navigation with SVG. 2005. (Enlace disponible 27/08/2015) http://www.svgopen.org/2005/papers/IndoorNavigationWithSVG/

14. Gonzalo Génova - What is a metamodel: the OMG's metamodeling infrastructure. 2009 (Enlace disponible al 23/09/2015)
http://www.ie.inf.uc3m.es/ggenova/Warsaw/Part3.pdf

15. EcuRed: Conocimiento con todos y para todos - Metamodelado (Enlace disponible 16/09/2015)
http://www.ecured.cu/index.php/Metamodelado

16. Anneke, Kleppe. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Capítulo 5. Diciembre de 2008.

17. Claudia Pons, Roxana Giandini y Gabriela Pérez: Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica. Octubre de 2009 (Enlace disponible al 27/08/2015)
http://sedici.unlp.edu.ar/bitstream/handle/10915/26667/INFORMATICA+Pons-Giandini-Perez.pdf;jsessionid=14E7745A0DFF08E7338B4CF60AA6736F?sequence=1

18. Romero Guillen Paola - Análisis y diseño orientado a objetos. Instituto tecnológico de la Laguna.
http://www.itlalaguna.edu.mx/Academico/Carreras/sistemas/Analisis%20y%20dise%C3%B1o%20orientado%20a%20objetos/semant.pdf

19. Carlos Mario Zapata, Gloria Lucía Giraldo, Byron Portilla, Duvan Gómez, Marcela Naranjo, Patricia Carmona - Aproximación a una ontología para lenguajes de modelado gráfico. Marzo de 2009. (Enlace disponible al 17/08/2015)
http://www.scielo.org.co/pdf/ring/n29/n29a3.pdf

20. Sitio web oficial de UML. (Enlace disponible al 27/08/2015).
http://www.uml.org/

21. Introduction to OMG's Unified Modeling Language - What is UML? (Enlace disponible al 27/08/2015).
http://www.omg.org/gettingstarted/what_is_uml.htm

22. UNET Virtual – Capítulo 12: Notaciones Formales. Lenguaje OCL. (Enlace disponible al 27/08/2015).
https://uvirtual.unet.edu.ve/pluginfile.php/256574/mod_resource/content/0/capitulo12_1_.pdf

23. Dan Massey, Y&L Consulting Inc - Introduction to OCL in Together. (Enlace disponible al 27/08/2015).
http://conferences.embarcadero.com/article/32200#SubtopiconeA

24. OMG Unified Modeling Language TM (OMG UML). Version 2.5 FTF – Beta 1. Capitulo 14. (Enlace disponible al 16/09/2015)
http://www.omg.org/spec/UML/2.5/Beta1/PDF/

25. Koch Nora and, Andreas Kraus - Toward a Common Metamodel for the Development of Web Applications. Julio de 2003. (Enlace disponible al 16/09/2015).
http://www.pst.informatik.uni-muenchen.de/~kochn/presentations/icwe03-pres.pdf

26. Carlos M. Zapata, Carlos A. Alvarez, Fernando Arango - Propuesta para el manejo de restricciones en modelos de clases usando atom3. 2005. (Enlace disponible al 23/09/2015)
http://www.researchgate.net/publication/28200509_Propuesta_para_el_manejo_de_restricciones_en_modelos_de_clases_usando_AToM3

27. Leandro Alegsa - Herramientas de Metamodelado. Mayo de 2012. (Enlace disponible al 17/08/2015)
http://www.alegsa.com.ar/Dic/herramienta%20de%20modelado.php

28. EcuRed: Conocimiento con todos y para todos - Herramientas de Metamodelado. (Enlace disponible al 23/09/2015)
http://www.ecured.cu/index.php/Herramientas_de_metamodelado

29. Sitio web de Eclipse. Eclipse Modeling Framework (EMF)
https://eclipse.org/modeling/emf/

30. Hans-Georg Fill, Dimitris Karagianni - On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. Marzo de 2013 (Enlace disponible 27/08/2015).
http://homepage.dke.univie.ac.at/fill/papers/Fill_Karagiannis_EMISA_2013.pdf

31. Constantinos Giannoulis - Model-driven Alignment. Apéndice A (Enlace disponible al 23/03/2015)
http://www.diva-portal.org/smash/get/diva2:708684/FULLTEXT02.pdf

32. "Adoxx Meta-Model Compiler. Modelling Methods for Robotic Systems" Programa de Cooperación Científico-Tecnológica entre el Ministerio de Ciencia, Tecnología e Innovación Productiva de la República Argentina (MINCyT) y el Ministerio de Ciencia e Investigación de la República de Austria (BMWF).
http://www.mincyt.gob.ar/cooperacion-bilateral-pais/austria-12

33. Sitio web oficial de Situm.
https://situm.es/index.php

34. Sitio web oficial de IndoorGML.
http://indoorgml.net/

35. Eich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Patrones de Diseño. 2002.

36. Ezequiel Postan - Patrones de Diseño. (Enlace disponible al 23/09/2015).
http://www.fceia.unr.edu.ar/ingsoft/Contribuciones/Patrones.pdf

37. Marco de Desarrollo de la Junta de Andalucía - Patrones de Diseño: Composite. (Enlace disponible al 23/09/2015)
http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/184

38. Patrones de Diseño: Patrón Composite. (Enlace disponible al 23/09/2015)
http://patronesdediseno.blogspot.com.ar/2009/05/patron-composite

Anexo I

Patrón de Diseño Composite

Anexo I

¿Qué es un patrón de diseño?

Un patrón de diseño representa una solución a un problema en términos de objetos e interfaces. En general consta de cuatro partes [35]:

- **Nombre**: Permite generar un vocabulario de diseño, esto trae ventajas de comunicación dentro de un grupo de trabajo y acorta la documentación refiriendo a los conceptos representados por el patrón.
- **Problema**: Describe las características y contexto en que aplicar el patrón puede traer buenos resultados.
- **Solución**: Describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. No describe una implementación concreta. Es una descripción abstracta de la solución a un conjunto de problemas con características comunes.
- **Consecuencias**: Resultados, ventajas e inconvenientes al aplicar el patrón. Impacto sobre la flexibilidad, extensibilidad y portabilidad de un sistema.

¿Por qué usar patrones de diseño?

Algunas breves ventajas de utilizar patrones de diseño son [36]:

- Los patrones de diseño son soluciones estudiadas que se han aplicado con éxito en diversos sistemas y que han respondido con buenos resultados a las exigencias que plantearon resolver.
- Permiten reutilizar con mayor facilidad buenos diseños y arquitecturas.
- Facilitan la documentación de los sistemas.

I.1 Patrón de Diseño: Composite

- **Descripción**

  El patrón Composite se utiliza para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que, al poseer todos ellos una interfaz común, se tratan todos de la misma manera [37].

- **Nombre**

  También es denominado Compositor.

- **Clasificación**

  Patrón estructural.

- **Motivación**

  Pensemos en una aplicación gráfica de componentes sencillos (líneas, texto, etc.). Imaginemos que necesitamos crear una serie de clases para guardar información acerca de figuras geométricas. Se crearán las clases *Círculo*, *Cuadrado* y *Triángulo*, que heredarán todas de la clase *Figura* y que implementarán el método *pintar()*. Además, se necesita poder tratar también grupos de imágenes, ya que nuestro programa permite seleccionar varias de estas figuras a la vez para moverlas por la pantalla. Para la implementación de estos grupos de *Figuras* podríamos caer en la tentación de crear una clase particular separada de las anteriores llamada *GrupoDeImágenes*, también con un método *pintar()*.

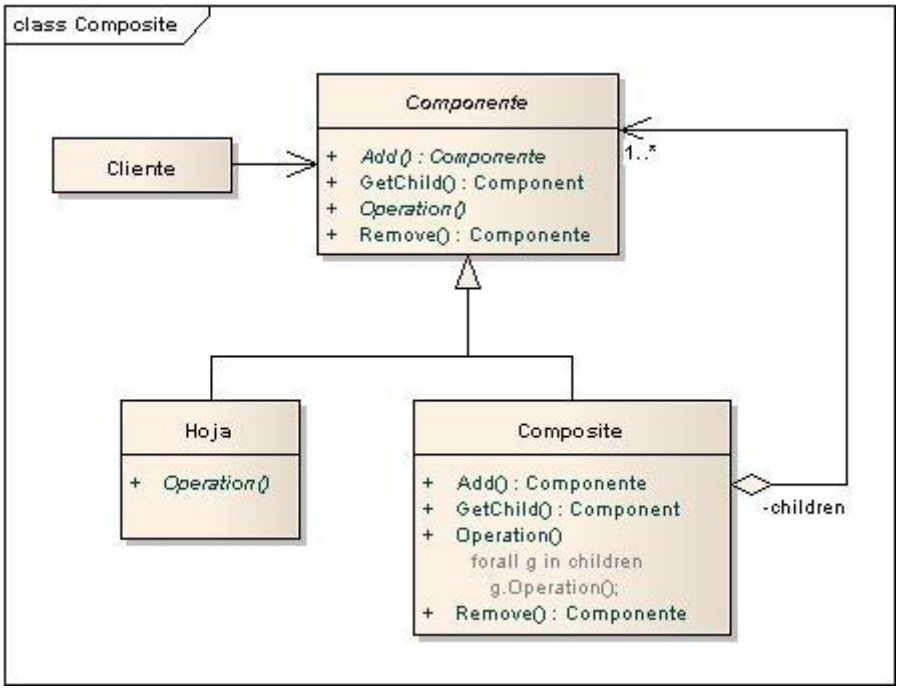
  Podríamos pensar la idea de separar en clases privadas componentes (figuras) y contenedores (grupos) pero tiene el problema de que, para cada uno de los dos tipo de objeto, el método *pintar()* tendrá una implementación diferente, aumentando la complejidad del sistema. Lo lógico es crear una clase abstracta que represente componentes y contenedores de la cual todas heredan, y que define sus operaciones. Por ejemplo, en este caso, se podría crear un clase *Gráfico* de la que hereden tanto la clase *Figura* como la clase *GrupoDeImágenes*, y que incluya el método *pintar()*. Además, ésta última tendría una relación "todo-parte" de multiplicidad con *Gráfico*: un *GrupoDeImágenes* contendría varios *Gráficos*, ya fuesen éstos *Cuadrados*, *Triángulos*, u otras clases *GrupoDeImágenes.*

  De esta forma, el patrón Composite da una solución elegante a este problema, de la que además resulta en una implementación más sencilla.

- **Aplicabilidad**
  - Se quiere realizar jerarquías de objetos del tipo "todo-parte"
  - Se quiere ser capaz de ignorar la diferencia entre los objetos individuales y composiciones de objetos. Los clientes tratarán a todos los objetos de la estructura compuesta uniformemente.

- **Estructura**

  El patrón se representa gráficamente de la siguiente manera [38]:



- **Participantes**

  - ***Componente***: Define la interfaz común para los objetos de la composición. Además, define la interfaz para acceder y gestionar los hijos. Implementa un comportamiento por defecto común a las subclases

  - ***Hoja***: Representa los objetos sin hijos de la composición. Define el comportamiento de los mismos.

  - ***Composite***: Define el comportamiento para los componentes que tienen hijos. Almacena los componentes con hijos e implementa las operaciones para la gestión de hijos

  - ***Cliente***: Manipula los objetos de la composición a través de la interfaz

- **Colaboraciones**

  Cliente usa el interfaz de *Componente* para interactuar con objetos en la estructura de *Composite*. Si el receptor es una *Hoja*, entonces la petición es manejada directamente. Si el receptor es un *Composite*, lanza la petición a sus hijos.

- **Consecuencias**

  - Define jerarquías de clases hechas de objetos primitivos y compuestos. Si el código cliente espera un objeto simple, puede recibir también uno compuesto.

- Simplifica el cliente, ya que los objetos simples y compuestos se tratan homogéneamente.
- Facilita la incorporación de nuevos tipos de componentes.
- Puede hacer el diseño demasiado general.

- **Implementación**

  Algunas recomendaciones para la implementación del patrón son las siguientes.

  - **Referencias explicitas a los padres.** Con ello se simplifica algunas operaciones de la estructura compuesta. Lo mejor es definirlas en la clase *Componente*. Gestionarlas al añadir/eliminar elementos de un *Composite*.
  - **Compartir Componentes:** Es muy útil por el ahorro de memoria que supone. La gestión del componente con varios padres puede llegar a ser compleja
  - **Maximizar la interfaz del componente:** Dar un comportamiento por defecto que sobrescribirán las subclases.
  - **Orden de los hijos:** En ocasiones los hijos presentan un orden, y hay que tenerlo en cuenta para la implementación
  - **Declaración de las operaciones de manejo de hijos:** Definirlas en la raíz *Componente* y darle una implementación por defecto permite aumentar la transparencia, pero se pierde seguridad (¿Cómo evitar que añada/elimine objetos a una hoja?). Si las definimos en la clase *Composite* se obtiene más seguridad pero se pierde transparencia al no existir una interfaz uniforme.

Acrónimos

Acrónimos

API – *Application Program Interface*

AQL – *Analytics Query language*

BNF – *Backus Naur Form*

CASE – *Computer Aided Software Engineering*

CMOF – *Complete MOF*

EBNF – *Extended Backus Naur Form*

EMOF – *Essential MOF*

GIS – *Sistema de Información Geográfica*

GML – *Geography Markup Language*

GPS – *Global Positioning System*

INS – *Sistema de navegación inercial*

LHS – *Left Hand Side*

MERL – *Metaedit* Reporting Language

MOF – *Model Object Facility*

OCL – *Object Constraint Language*

OMG – *Object Management Group*

OMI – *Open Model Initiative*

PHP – *Hypertext Preprocessor*

QR – *Quick Response*

RFID – *Radio Frequency IDentification*

RHS – *Right Hand Side*

SMOF – *Semantic MOF*

SOO – *Single sign-on*

UI – *User Interface*

UML – *Unified Modeling Language*

UNLP – *Universidad Nacional de La Plata*

UWB – *Ultra-wide-band*

XMI – *XML Metadata Interchange*

XML – *Extensible Markup Language*

Agradecimientos

Esta tesina no hubiera sido posible sin la valiosa colaboración de muchas personas e instituciones, a quienes deseamos expresar nuestro profundo agradecimiento.

En primer lugar a nuestra querida Facultad de Informática de la Universidad Nacional de La Plata, institución pública y gratuita, por su excelencia académica que posibilitó nuestro desarrollo, tanto profesional como personal, durante estos maravillosos años en los que nos desempeñamos como estudiantes de esta casa de altos estudios.

A las Doctoras, Catalina Mostaccio, directora de esta Tesina, y Claudia Pons, co-directora, por guiarnos y acompañarnos, por compartir sus conocimientos y experiencia con tanta generosidad y calidez humana. Gracias por estar siempre dispuestas a responder nuestras dudas y proporcionarnos su infinita colaboración, en este arduo pero hermoso camino.

Y, por supuesto, a nuestras familias, amigos y colegas que nos ayudaron para llegar a la recta final.

Agradecimientos de Karen Poch

Quisiera agradecer:

A mi mamá Cristina, por haberme apoyado en todo momento, por sus consejos y valores, por la motivación constante a que siga luchando por mis sueños, por alentarme cada día a no bajar los brazos y por todo el amor que me brinda.

A mi hermana Brenda, por cuidarme, alentarme, apoyarme, acompañarme en este camino, por sacarme una sonrisa en los momentos de tristeza y, por ser un ejemplo a seguir.

A mi papá Jorge, por apoyarme, acompañarme e incentivarme a la culminación de este trabajo.

A Lucía, Paula, María Julia, Victoria, Julieta, Viviana y Laura, amigas que me regaló la vida. Gracias por su amistad incondicional, por estar en cada momento, por darme fuerzas y aliento para "seguir adelante", por comprender ausencias y sobre todo por apoyarme en todos mis proyectos. Son amigas que valen oro.

A Paula, amiga y autora de esta Tesina. Gracias por acompañarme en este camino, por brindarme tu confianza y apoyo, por ser incondicional, y por hacer que este camino haya sido más ameno. Fue un placer trabajar con vos.

A Cristian y a la familia Rispoli, por haberme abierto las puertas de sus casas y por el apoyo constante que me brindaron durante la realización de esta Tesina.

A Santiago, mi sobrino postizo, por hacer más divertidas y especiales las largas tardes de trabajo.

A mis compañeros de trabajo y de cátedra de la Facultad de Informática, que hicieron posible la realización de esta Tesina.

Agradecimientos de Paula Rispoli

Quisiera agradecer:

A Cristian, mi marido, por haberme apoyado y motivado durante toda mi estadía en la Facultad. Por acompañarme y cuidarme día a día. Te amo.

A Santiago, mi hijo, mi sol, gracias por sacarme una sonrisa todas las mañanas. Sos la luz que ilumina mi vida.

A mi mamá Roxana, por ser mi ejemplo y mi sostén. Gracias por los valores que me inculcaste, por cuidarme y apoyarme siempre, por amarme y demostrármelo. Gracias por ser mi mamá.

A mi hermana Rocío, por cuidarme, por escucharme, por darme los mejores consejos, por demostrarme que pase lo que pase siempre hay un lado positivo, por hacerme reír.

A Sofía, Laura y Viviana. Gracias por estar siempre a mi lado, por interesarse en este trabajo, por alentarme a realizar el "último esfuerzo", por entenderme, y quererme. Las adoro.

A Karen, amiga y autora de esta Tesina. Gracias por ser mi compañera de este trabajo, por tu paciencia, por tu alegría, por tu buena predisposición y, sobre todo, gracias por tu amistad.

A Cristina y a Brenda por su hospitalidad y su permanente colaboración a lo largo de este proceso.

A mis compañeros de trabajo que colaboraron en la realización de esta Tesina.