Capítulo IV. Diseño e implementación de software

En este capítulo se presenta un reporte de las etapas de diseño e implementación del software desarrollado para este proyecto de tesis. Esta aplicación se denominará en adelante VELOAT, por ser las siglas de "*Voice Enhanced Learning Object Authoring Tool*".

Se presenta en primer lugar una sección que trata sobre el modelado de la aplicación, que incluye la identificación de casos de uso, el diseño de la interfaz gráfica de usuario, los patrones de diseño e ideas principales en las que está basada la estructura del sistema y la arquitectura del mismo. Después se presenta otra sección que aborda los detalles más importantes de la etapa de implementación de este software.

4.1. Diseño y modelado

4.1.1. Casos de uso

El objetivo principal del sistema desarrollado es brindar un ambiente para la creación de objetos de aprendizaje a partir de contenido educativo previamente generado. La aplicación guía al usuario, que idealmente es un profesional del área de la educación, en la creación de objetos de aprendizaje a través de tres etapas. Cada etapa corresponde a un caso de uso distinto. En el anexo D de este documento de tesis se encuentra una descripción detallada de cada caso de uso, sin embargo, es importante hablar ahora de estas etapas.

Establecimiento de referencias a recursos educativos

En esta etapa el usuario “define” los recursos educativos que va a incluir en su paquete, entendiéndose como “recurso educativo” a un conjunto de archivos de contenido y/o otros recursos educativos. En la etapa de establecimiento de referencias a recursos educativos hay generación semiautomática de metadatos, la de los archivos que decida incluir el usuario en sus objetos de aprendizaje, pero este proceso es transparente para él. Sin embargo, el educador puede optar por editar estos metadatos generados y completarlos, así como agregar descripciones a nivel de recursos, actividades, organizaciones de contenido y a nivel global (para el paquete u objeto de aprendizaje como un todo).

Definición de organizaciones de contenido educativo

En esta etapa el educador define la manera en la que desea que el contenido educativo se presente en los LMS’s. Una “organización de contenido” es una jerarquía de actividades, en la que cada actividad contiene a otras actividades o tiene asociada a un recurso educativo, que es con el cual se desarrolla la actividad de aprendizaje.

Empaquetado del objeto de aprendizaje

Para esta etapa ya debe de contarse con definición de recursos educativos, y opcionalmente con organizaciones de contenido. Esta etapa consiste en generar un archivo contenedor de toda la información generada por el educador en la aplicación, referente a recursos educativos y organizaciones de contenido. El formato, estructura y contenido de este archivo es determinado por los estándares SCORM e IMS *Content Packaging*, y el proceso de empaquetado se realiza de manera transparente para el usuario.

La figura 4.1 muestra un diagrama UML de los principales casos de uso del software de este proyecto de tesis.

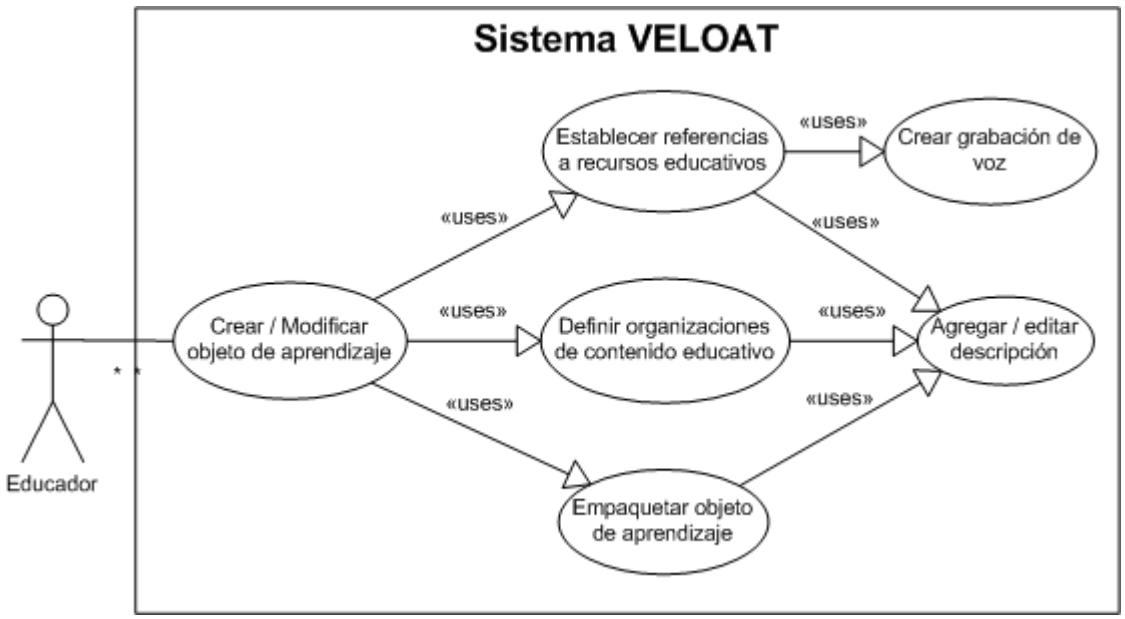

Figura 4.1. Diagrama UML de casos de uso de VELOAT

Como se puede observar en la figura 4.1, hay tres casos de uso adicionales a los correspondientes a las tres etapas de la creación de objetos de aprendizaje. Estos casos de uso adicionales corresponden cada uno a las funcionalidades principales de cada módulo del sistema, como se muestra a continuación:

- Crear/modificar objeto de aprendizaje - este caso de uso corresponde a la funcionalidad principal del módulo central del sistema, y engloba a las tres etapas de creación de objetos de aprendizaje.
- Agregar/editar descripción - corresponde al módulo de metadatos del sistema.
- Crear grabación de voz - corresponde al módulo de tecnologías de voz del sistema, siendo ésta una primera funcionalidad implementada.

4.1.2 Diseño de interfaz gráfica de usuario

La interfaz gráfica de usuario (GUI) de la aplicación desarrollada está basada en su mayoría en el uso de árboles y listas, debido a que estos componentes son los ideales para el manejo de jerarquías. *Java Web Start* permite el uso de *Swing* para el desarrollo de interfaces gráficas de usuario, así que, a pesar de tratarse de una aplicación de Internet, VELOAT tiene la apariencia de cualquier otra aplicación de escritorio. Son seis las principales interfaces gráficas de usuario de este programa; a continuación se presentan imágenes de estas interfaces y su relación con los casos de uso definidos anteriormente.

GUI principal

Esta GUI es la ventana principal del programa. Corresponde al módulo central del sistema y al caso de uso “Crear/Modificar objeto de aprendizaje” que, como muestra en la figura 4.1, engloba a las tres etapas de la creación de objetos de aprendizaje. Esta GUI contiene tres pestañas, cada una correspondiente a una de estas etapas. Además contiene una barra de menús y una barra de herramientas con las mismas funciones. El menú “Archivo” da opciones para crear nuevos objetos de aprendizaje o editar existentes para modificarlos. El menú “Preferencias” permite cambiar la apariencia de la aplicación, así como mostrar u ocultar una barra de *tips* de uso. El menú “Herramientas” proporciona funcionalidades relacionadas con los módulos de metadatos y de tecnologías de voz del sistema. Finalmente el menú “Ayuda” muestra información relacionada con la creación y uso del software, así como el manual de usuario. La figura 4.2 muestra esta ventana.

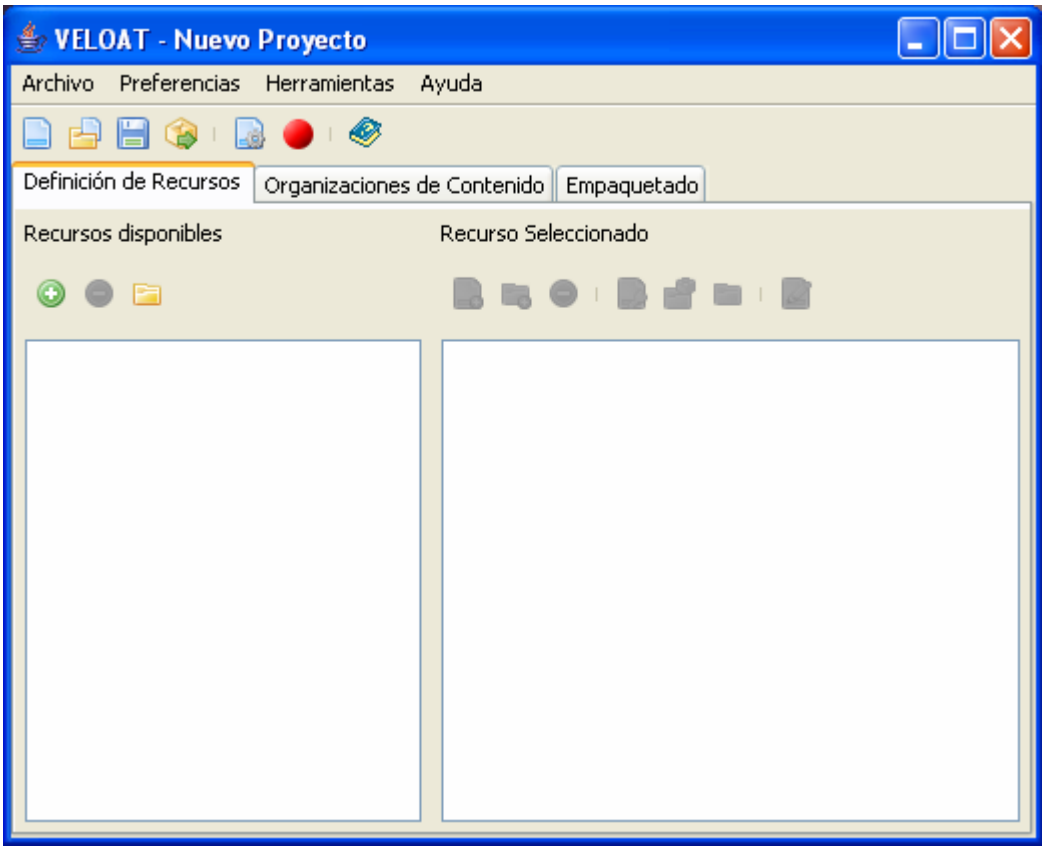

Figura 4.2. GUI correspondiente al programa principal

GUI para definición de recursos educativos

Corresponde al caso de uso de la primera etapa de creación de objetos de aprendizaje, la de "Establecer referencias a recursos educativos". Consta de un panel que permite al usuario buscar archivos y crear recursos educativos reutilizables a partir de ellos. Esta GUI muestra a la izquierda una lista de los recursos definidos por el usuario, y a la derecha un árbol con el contenido del recurso seleccionado en la lista. Al seleccionar un nodo del árbol y hacer clic derecho, se muestra un menú emergente que permite editar el recurso, ya sea modificando sus atributos (sub-ruta y granularidad), agregándole o quitándole archivos o dependencias a otros recursos, o definiendo descripciones (metadatos). Esta GUI también cuenta con barras de herramientas para realizar estas tareas (ver figura 4.3).

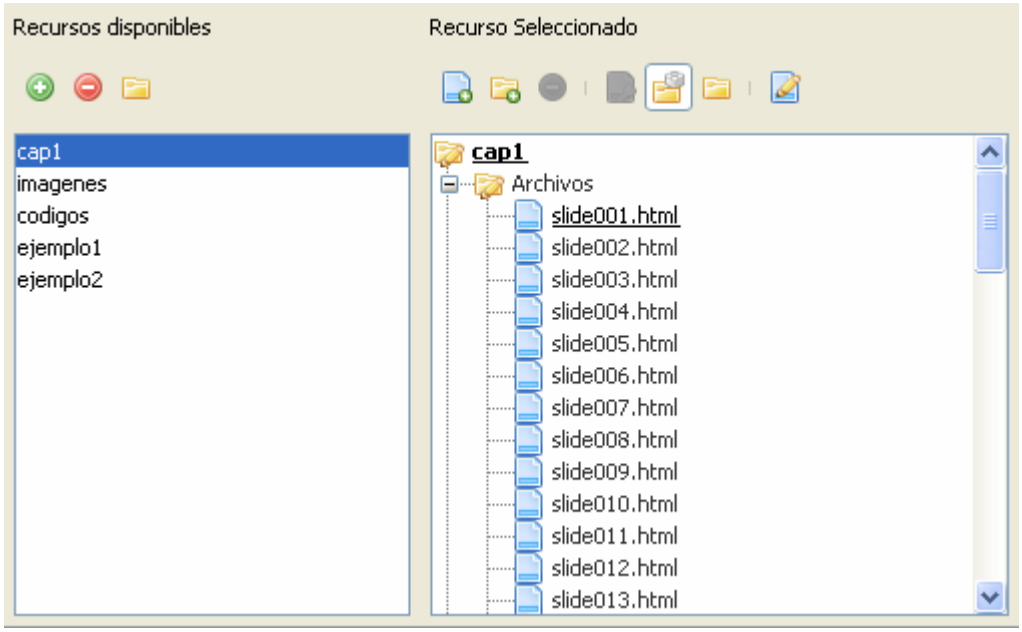

Figura 4.3. GUI para establecimiento de referencias a recursos educativos

GUI para organizaciones de contenido

Corresponde al caso de uso de la segunda etapa de creación de objetos de aprendizaje, “Definir organizaciones de contenido”. Esta interfaz muestra, al igual que la anterior, a la izquierda una lista, pero en esta ocasión, de las organizaciones de contenido definidas por el usuario. A la derecha también se muestra un árbol, que muestra y permite modificar la jerarquía de actividades de la organización de contenido seleccionada en la lista. Al igual que en el caso anterior, esta GUI cuenta con barras de herramientas para la realización de tareas comunes, mismas que también se pueden realizar si se selecciona un nodo del árbol y se hace click derecho, con lo que se muestra un menú emergente para editar a la organización de contenido seleccionada, agregándole o quitándole actividades, cambiando títulos a mostrar, agregando descripciones (metadatos) o asociando recursos. (Ver figura 4.4).

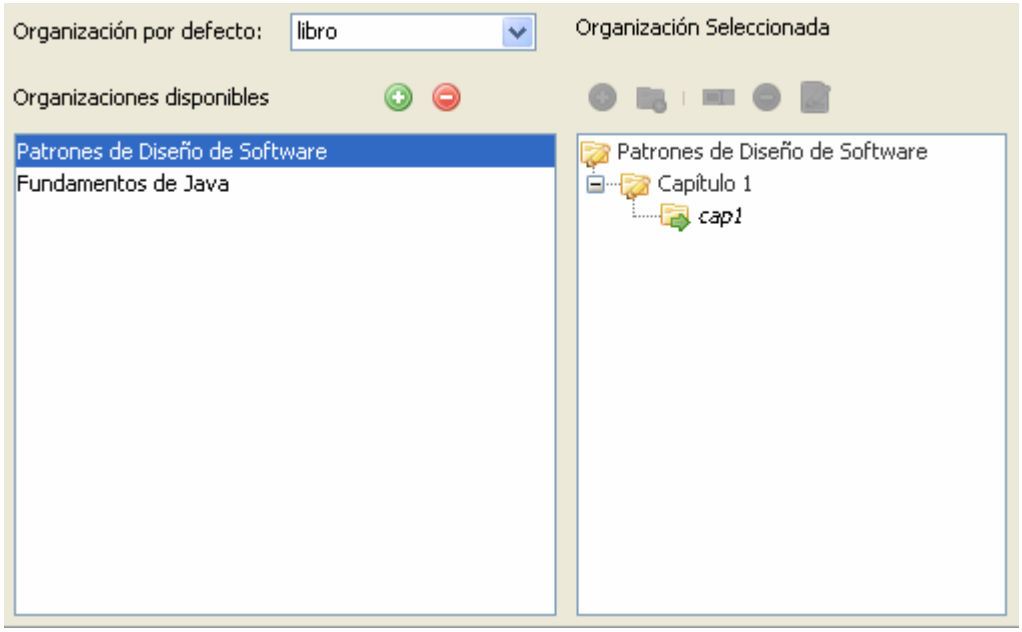

Figura 4.4. GUI para creación de organizaciones de contenido educativo

GUI para empaquetado de Objetos de Aprendizaje

Corresponde al caso de uso de la tercera etapa de creación de objetos de aprendizaje, "Empaquetar objeto de aprendizaje". Esta interfaz permite al usuario empaquetar sus recursos y organizaciones de contenido educativo en un objeto de aprendizaje o paquete de contenido educativo. SCORM especifica que el paquete debe ser un archivo comprimido (en formato ZIP). Esta interfaz muestra un árbol a manera de vista preliminar del contenido del archivo contenedor a generar y permite al usuario llevar a cabo el empaquetado. Esta GUI también cuenta una barra de herramientas para tareas comunes, como la edición de los atributos del paquete (su clave de identificación y su versión), la creación y edición de descripciones (metadatos) a nivel de paquete de contenido, y el mismo empaquetado (Ver figura 4.5).

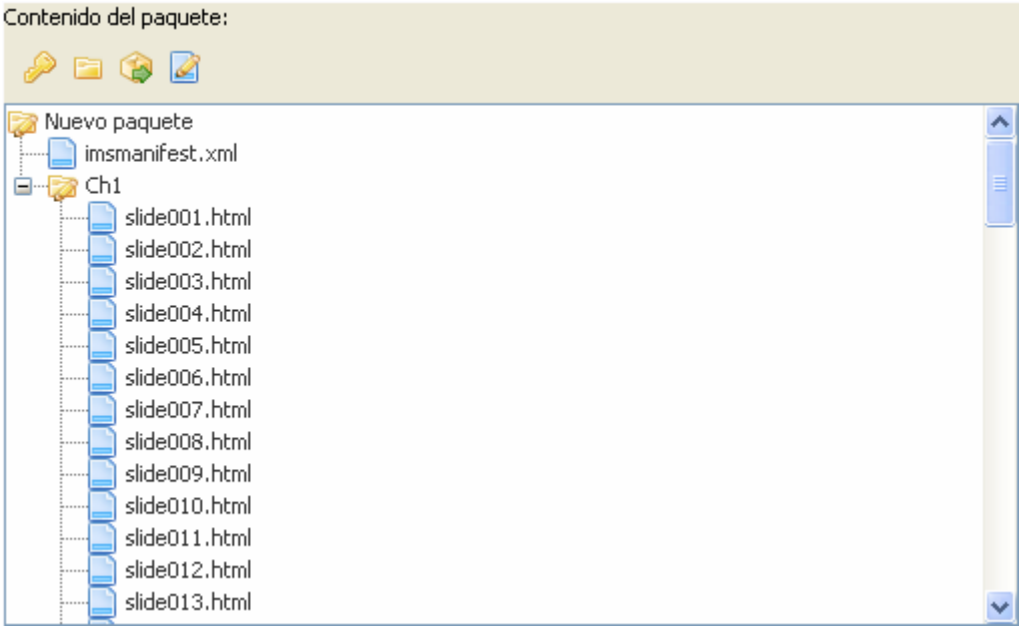

Figura 4.5. GUI para empaquetado de objetos de aprendizaje

GUI para edición de metadatos

Corresponde al módulo de metadatos del sistema, y al caso de uso “Agregar/editar descripción”, que está presente en las tres etapas de creación de objetos de aprendizaje para crear metadatos a varios niveles de granularidad. Como se mencionó anteriormente, se pueden agregar descripciones o metadatos a cualquier nivel, desde archivos físicos hasta a nivel de paquete de contenido educativo. Esta GUI consta de un conjunto de formas de llenado en las que el usuario pude agregar metadatos, o editarlos en el caso de los generados a nivel de archivos físicos. Cada categoría de metadatos se presenta en una pestaña distinta. Los metadatos se guardan en el archivo de manifiesto del paquete como etiquetas de XML de acuerdo al estándar IEEE LOM, pero esto es transparente para el usuario. La figura 4.6 muestra una imagen de esta GUI, que también puede usarse de manera independiente del módulo central del programa.

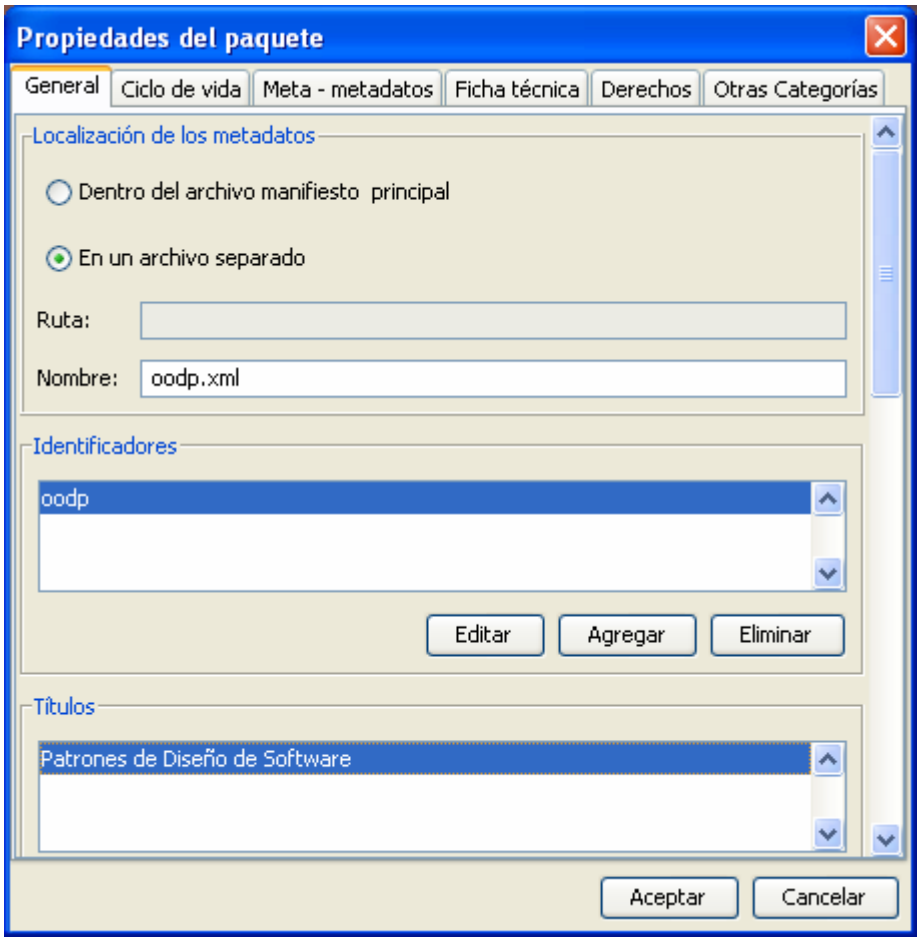

Figura 4.6. GUI para creación y edición de metadatos

GUI para grabación de voz

Esta GUI corresponde al primer avance del módulo de tecnologías de voz del sistema, que permite crear grabaciones de voz con ayuda de dispositivos de entrada, tales como micrófonos. Se trata de un diálogo que, al igual que la GUI anterior, puede usarse de manera independiente del módulo central, y permite al usuario elegir opciones sobre grabaciones de voz, como la calidad de la grabación y el archivo de salida de la grabación. Además, cuenta con tres botones para controlar el inicio y el fin de las grabaciones realizadas y para reproducirlas (Ver figura 4.7).

Figura 4.7. GUI para grabación de voz

4.1.3. Diseño de estructura

La aplicación desarrollada está basada en diversos patrones de diseño de software. Los principales son el *Model - View - Controller* (MVC) y el *Framework*, que combinados conforman la base de la estructura de clases del sistema. El anexo C de este documento provee mayor información acerca de patrones de diseño.

El patrón MVC se utilizó porque provee un mecanismo robusto para lograr consistencia entre interfaz gráfica de usuario y lógica aplicativa. En teoría un modelo puede tener una o varias vistas, pero para el diseño de este sistema se optó por tener una correspondencia uno a uno entre modelos y vistas en la mayoría de los casos. De esta manera, modelo, vista y controlador pueden encapsularse en un solo componente, al que se denominó "*Manager*". El manejo de estos componentes constituye la base o el *framework* del programa. Si se deseara agregar en el futuro una nueva funcionalidad, sólo sería necesario agregar uno o varios *managers* al sistema. La figura 4.8 ilustra esta idea.

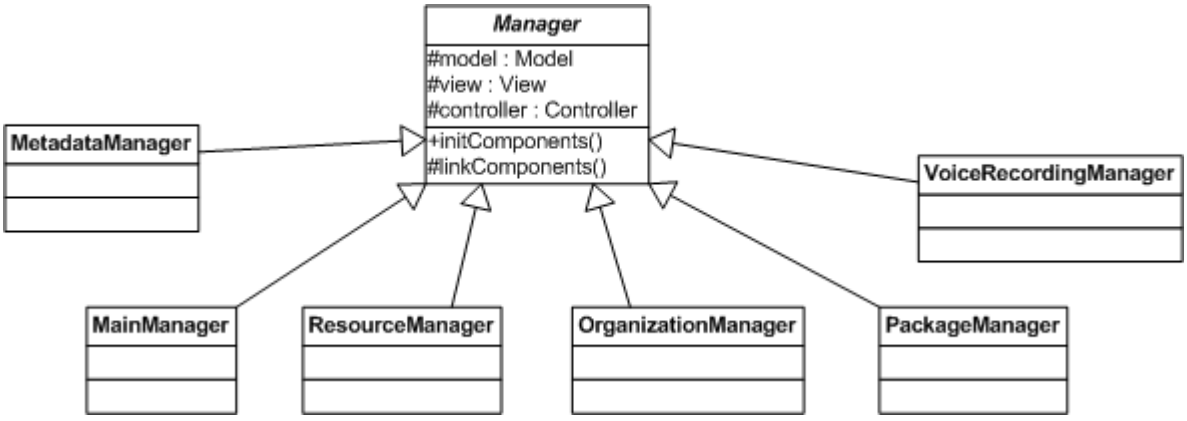

Figura 4.8. *Managers*, base del diseño estructural de VELOAT

Un *manager* puede contener como atributos o crear instancias de otros *managers*. La clase MainManager, por ejemplo, es la encargada del control principal del sistema, y tiene como atributos a otros *managers*, objetos de tipo ResourceManager, OrganizationManager y PackageManager, debido a que para su correcto funcionamiento necesita de la interacción de esos objetos.

A su vez, los *managers* que conforman a MainManager tienen algo en común, que es la necesidad de generar y editar metadatos. La clase MetadataManager proporciona esta funcionalidad, por lo que esos *managers* instancían objetos de este tipo. De manera similar, la clase MainManager crea una instancia de VoiceRecordingManager para mostrar al usuario la interfaz de grabación de voz. En la figura 4.9 se puede ver la relación entre estos componentes.

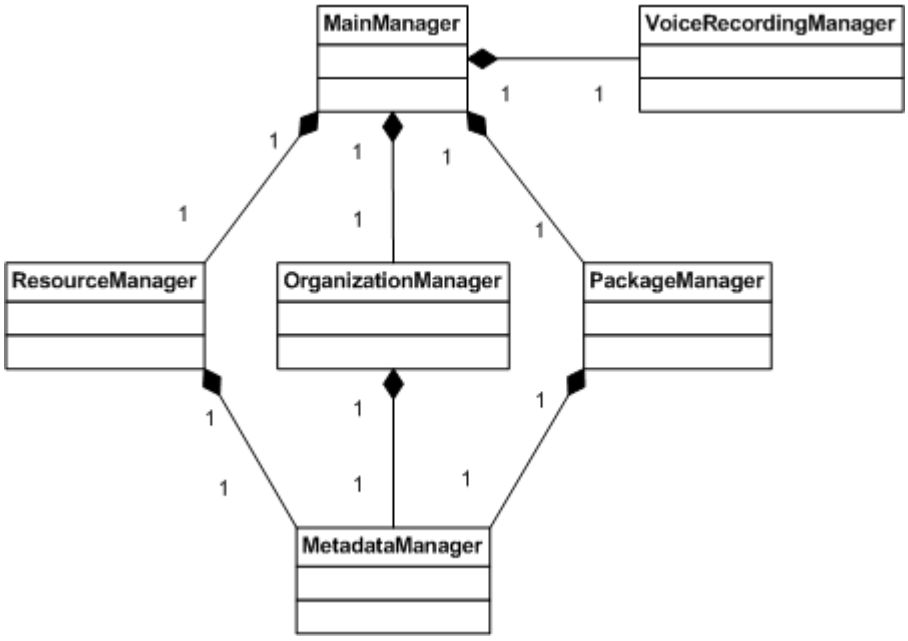

Figura 4.9. Relaciones entre *managers*

Otra característica importante de los *managers* es que, a pesar de especificar una relación uno a uno entre modelo, vista y controlador, permiten asociar varias vistas a un modelo, siempre y cuando las vistas adicionales estén contenidas dentro de la vista principal del *manager*. Por ejemplo, la clase MainView tiene como atributos a otras vistas, PackageView, OrganizationView y ResourceView. Todas estas vistas se vinculan a la clase MainModel, como se muestra en la figura 4.10, y así, un cambio en el modelo principal del programa se ve reflejado en las vistas correspondientes a cada etapa de la creación de objetos de aprendizaje.

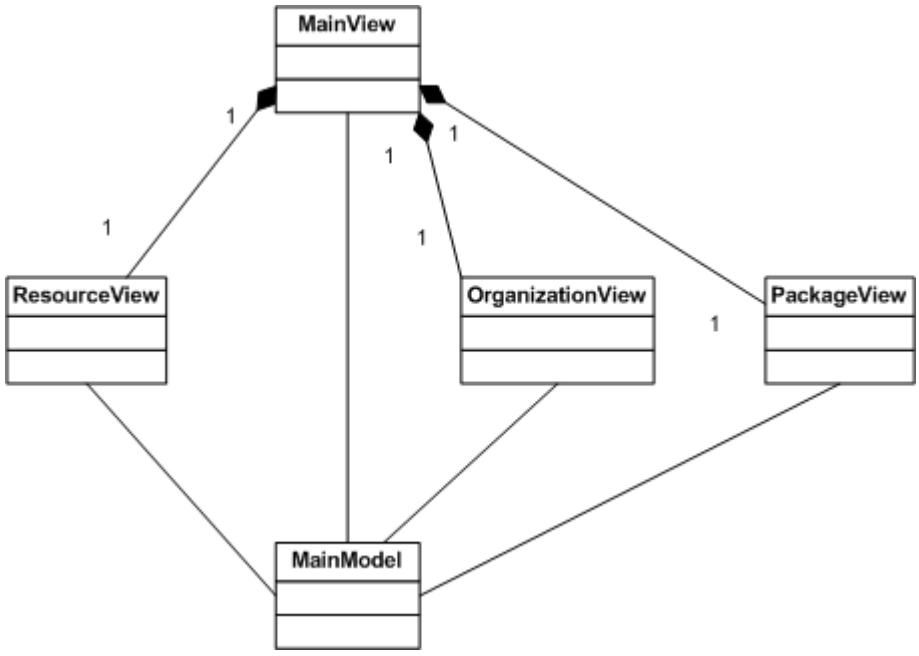

Figura 4.10. Vinculación de varias vistas a un solo modelo en los *managers*

En resumen, y de manera más conceptual, un *manager* puede incluir por composición o instanciar a otros *managers*; y la relación entre modelo y vista en estos componentes inicialmente es de uno a uno, pero puede volverse de uno a muchos.

4.1.4. Arquitectura del sistema

Como se ha venido mencionando, VELOAT se compone de tres módulos principales. Cada módulo está formado por uno o más *managers* que contribuyen a la realización de las funcionalidades de cada módulo. A continuación se describe la composición de cada módulo a detalle, incluyendo qué *managers* los conforman y cuáles son sus funciones.

4.1.4.1. Módulo de empaquetado

El módulo de empaquetado está conformado por los siguientes *managers*:

- MainManager - es el componente que se encarga del control principal de la aplicación. Engloba a los *managers* que proporcionan las funcionalidades de las tres etapas de creación de objetos de aprendizaje, además de funcionalidades como crear, abrir y guardar archivos manifiestos de objetos de aprendizaje.
- ResourceManager - se encarga de la definición de los recursos educativos a incluir en los objetos de aprendizaje. El educador debe dar como entrada principal a este componente URI's de archivos de contenido previamente creados, con las cuales se genera de manera transparente al usuario una parte del archivo de manifiesto del paquete, que es la correspondiente a los recursos.
- OrganizationManager - se encarga de la definición de organizaciones de contenido. A través de una vista de árboles y listas, el usuario define jerarquías de actividades para organizar sus recursos educativos, mientras de manera transparente se generan las etiquetas XML del archivo de manifiesto del paquete que corresponden a las organizaciones de contenido.

- PackageManager - se encarga del proceso de creación del archivo contenedor de los recursos y organizaciones de contenido definidos, el llamado *Package Interchange File* (PIF). Este archivo se genera en base a la información generada en el manifiesto XML que se esté manejando.

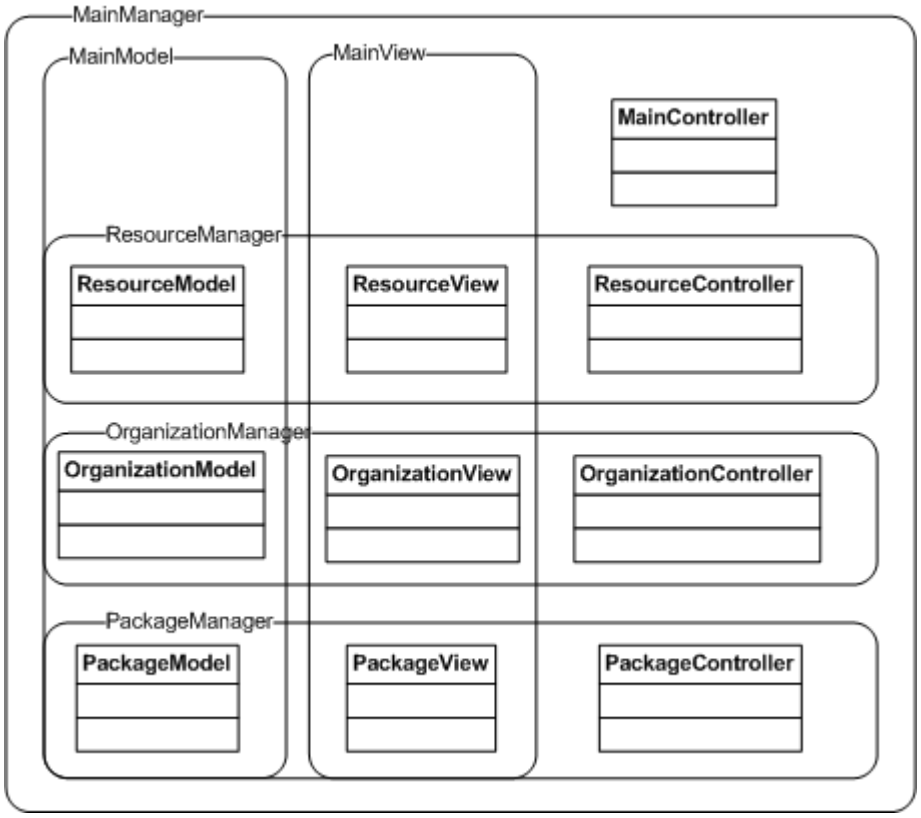

Figura 4.11. Arquitectura del módulo de empaquetado

4.1.4.2. Módulo de metadatos

En este módulo hay un *manager* para cada categoría de metadatos en estándar IEEE LOM que tenga multiplicidad 1, otro *manager* para las categorías que tengan permitida una multiplicidad mayor a 1, y otro *manager* principal que engloba a todos, de manera similar al caso del módulo anteriormente descrito. Los *managers* principales de este módulo son:

- LomManager - este es el *manager* que engloba a los demás, su función es formar un modelo y una vista completos para el manejo de metadatos, en la que la vista principal contiene pestañas con vistas para cada categoría de metadatos, y el modelo principal engloba a la descripción completa (las 9 categorías de metadatos) en cuestión.

- *Managers* para la gestión de metadatos por categorías - se trata de un conjunto de *managers* que tienen por objetivo dividir la funcionalidad principal del módulo. Hay un *manager* por cada categoría de metadatos en el estándar IEEE LOM, y un *manager* adicional para las categorías cuya multiplicidad permitida es mayor a 1. A continuación se presenta la lista de nombres estos mánagers:
    - GeneralManager
    - LifeCycleManager
    - MetaMetadataManager
    - TechnicalManager
    - EducationalManager
    - RightsManager
    - RelationManager
    - AnnotationManager
    - ClassificationManager
    - OtherCategoriesManager

Los modelos en cada uno de estos *managers* se encargan de la generación de valores de metadatos, en caso de que aplique. Debido a que en este trabajo no se abarca la generación de metadatos en todas las categorías del estándar IEEE LOM, algunas clases correspondientes a los modelos de estos *managers* sólo se limitan a las tareas de edición de metadatos. Se sugiere la implementación de nuevas funcionalidades de generación de metadatos en ellas como trabajo a futuro. La figura 4.12 muestra la arquitectura del módulo de metadatos de este sistema.

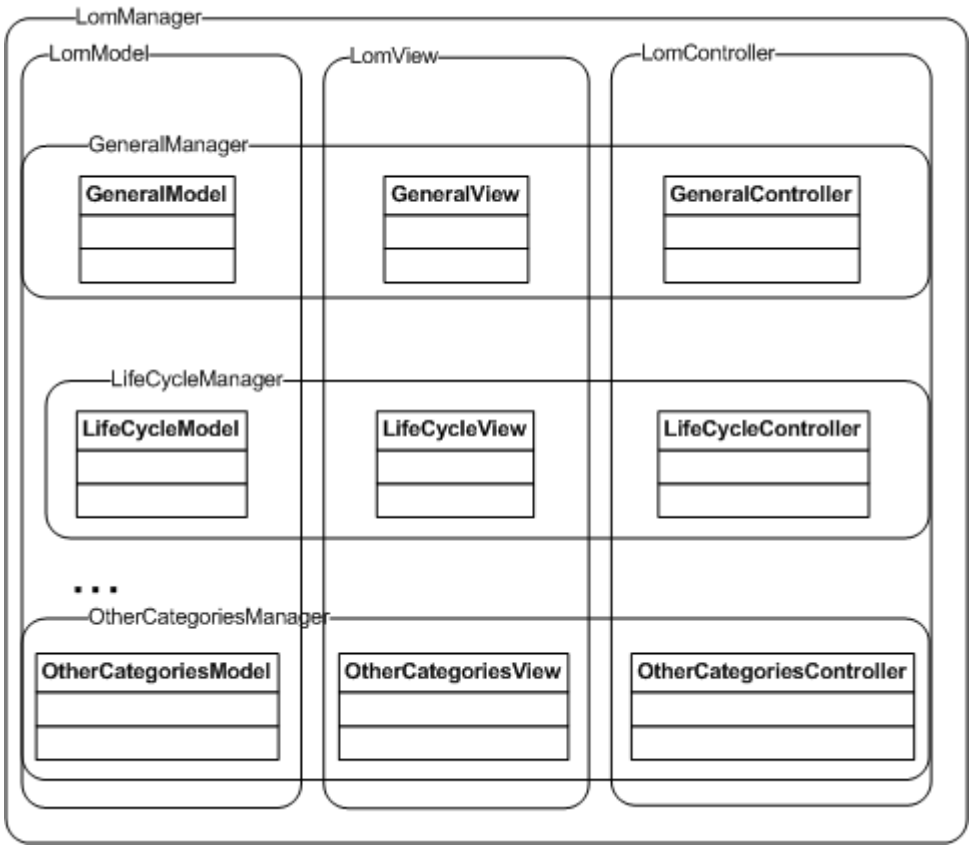

Figura 4.12. Arquitectura del módulo de metadatos

Arquitectura de sistemas generadores de valores de metadatos

Como se mencionó anteriormente, el código de generación de metadatos en VELOAT se encuentra dentro de las clases *Model* de cada categoría de LOM. Actualmente el sistema sólo cuenta con análisis de contenido, no analiza contextos. A continuación se presenta un ejemplo de arquitectura de un sistema cuya tarea principal es la generación de valores de metadatos, considerando como fuentes tanto el contexto como el contenido. Esta arquitectura puede ser combinable con la arquitectura de VELOAT a futuro para implementar funciones de generación de metadatos más elaboradas.

El principio fundamental de un sistema generador de metadatos es simple. Un componente del sistema obtiene tanta información como sea necesaria o posible del contenido del recurso proporcionado como entrada, mientras que otro componente obtiene la información que exista sobre el contexto del recurso. Los resultados después son mostrados en algún formato establecido, ya sea como Página Web o como XML [Kran, 2005].

Distintas fuentes de metadatos implican posibles valores duplicados para un mismo campo, por lo que es necesario además contar con un componente de solución de conflictos que se encargue de elegir el valor que sea más acertado para determinado metadato. Generalmente se hace uso de fórmulas matemáticas y/o técnicas heurísticas en este componente para este fin [Cardinaels *et al.*, 2005]. La figura 4.13 muestra un esquema de la arquitectura descrita.

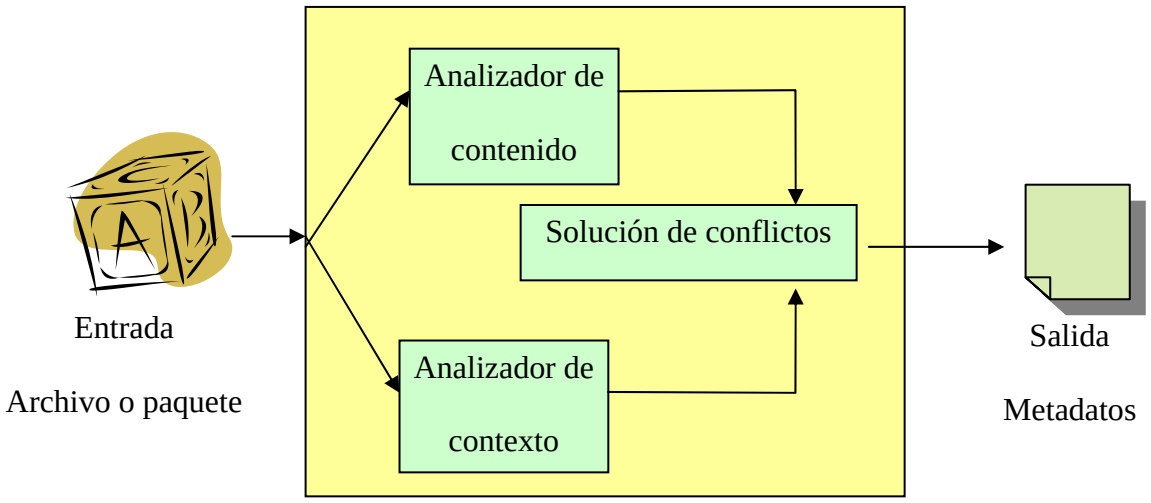

Figura 4.13. Ejemplo de arquitectura de un sistema generador de valores de metadatos

Los tres componentes principales de esta arquitectura son vistos como “cajas negras” en la figura anterior, pero cada componente puede analizarse a más detalle. El analizador de contenido, por ejemplo, puede constar de una superclase abstracta o interfaz que defina los métodos y propiedades de todo analizador de contenido, y un conjunto de subclases especializadas para cada tipo distinto de contenido soportado por el programa, como texto para archivos PDF, TXT, etc. Lo mismo sucede con el analizador de contexto, como se puede apreciar en las figuras 4.14.a y 4.14.b.

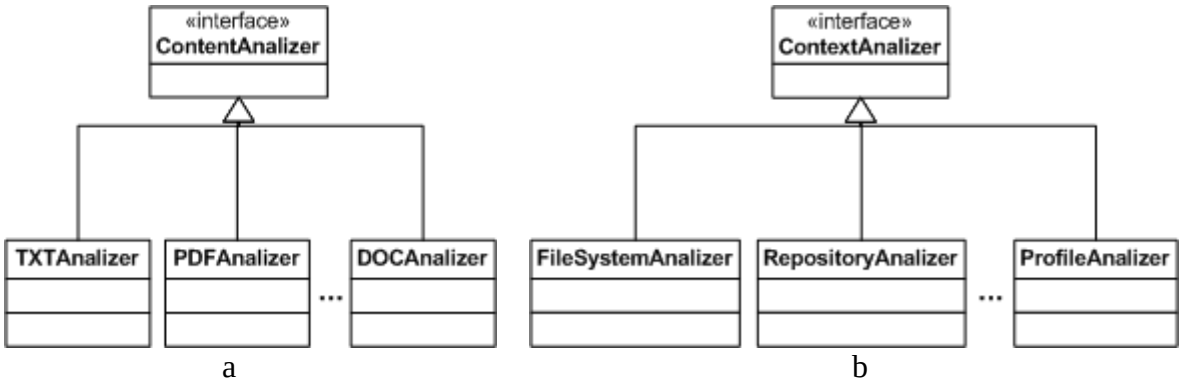

Figura 4.14. Ejemplo de diseño de módulos de obtención de valores de metadatos

a) analizador de contenido, b) analizador de contexto

El caso del diseño del módulo de solución de conflictos es distinto, pero no deja de ser una caja negra en el diagrama anterior. Este componente también puede estar formado por un grupo de clases que trabajen en conjunto para finalmente decidir el valor adecuado para un metadato específico. Su diseño puede variar dependiendo de las técnicas empleadas para asignar un valor adecuado a los metadatos.

4.1.4.3. Módulo de tecnologías de voz

Al igual que los módulos anteriores, este módulo también está basado en *managers*. Actualmente sólo la tecnología de grabación de voz está implementada, por lo que sólo se cuenta con un manager para crear grabaciones de voz, con sus respectivas clases *Model*, *View*, *Controller*, y otras clases auxiliares, como se ilustra en la figura 4.15. Queda como trabajo a futuro la implementación de nuevos *managers* relacionados con el desarrollo de contenidos educativos basados en otros tipos de tecnologías de voz.

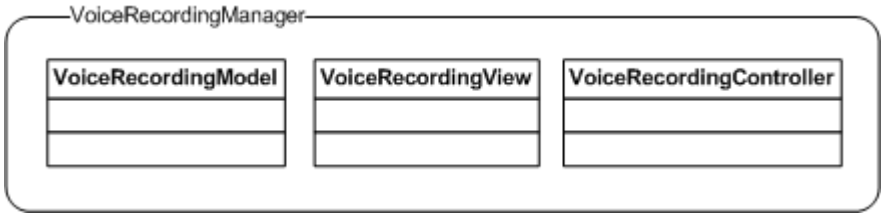

Figura 4.15. Arquitectura del módulo de grabación de voz

4.2. Detalles de implementación

4.2.1. Lenguaje de programación y tecnologías usadas

VELOAT está programado enteramente en Java. Debido a que se eligió *Java Web Start* como opción de implementación, VELOAT debe distribuirse en forma de archivos JAR, que deben incluir tanto archivos CLASS como archivos correspondientes a recursos utilizados por el programa, como son los archivos de las imágenes de los botones en la GUI o archivos de configuración. La figura 4.16 muestra el diagrama de paquetes del sistema.

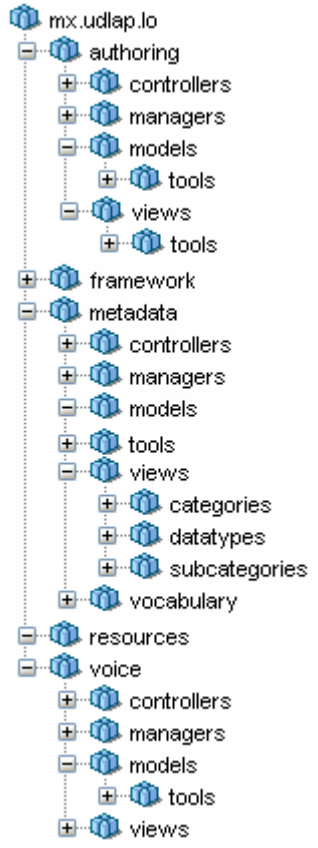

Figura 4.16. Diagrama de paquetes de VELOAT

Como se puede observar en la figura 4.16, el sistema consta de varios paquetes, siendo el principal *mx.udlap.lo*. Hay tres sub-paquetes dentro del paquete principal: *authoring*, *metadata* y *voice*. Estos paquetes corresponden a los tres módulos principales de los que se ha venido hablando en este capítulo y en capítulos anteriores. Cada uno de estos paquetes tiene sub-paquetes para clases *Model*, *View*, *Controller* y *Manager*.

Además de los paquetes correspondientes a los módulos del sistema, también están presentes otros dos paquetes: los paquetes *framework* y *resources*. El primero contiene clases e interfaces que conforman el *framework* del sistema, y en base a las cuales se desarrollan el resto de las clases del sistema. El segundo contiene archivos adicionales a las clases de Java del sistema, como imágenes y *schemas*, que en su momento requieren ser cargados para ayudar a realizar funciones específicas. Este paquete cuenta también con una clase *Factory* para instanciar objetos de Java en base a estos recursos adicionales.

4.2.2. Implementación de patrones de diseño y bases de la estructura del sistema

Como se mencionó anteriormente, se usó el patrón MVC en la definición de la base estructural del programa. Como se describe en el Anexo C, este patrón de diseño es compuesto, ya que hace uso del patrón *Observer*. Java ofrece un *framework* para la implementación de este último patrón, que consta de las clases *Observer* y *Observable* del paquete *java.util*. Se optó por utilizar estas clases en la implementación del programa como parte del patrón MVC, como se muestra en la figura 4.17.

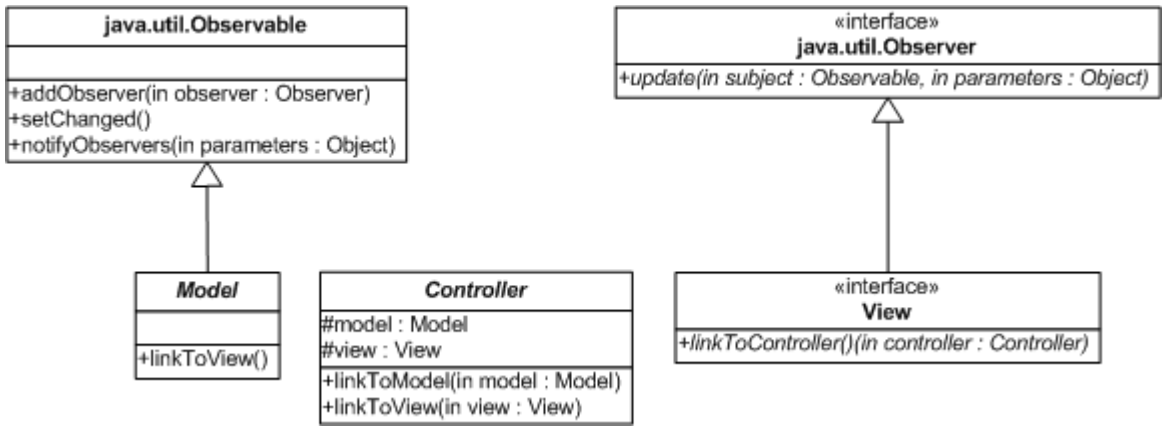

Figura 4.17. Uso del paquete *java.util* para la implementación del patrón *Observer*

Dentro del paquete *framework* del sistema, hay cuatro componentes que implementan las funcionalidades comunes a todos los *managers*. Tres de ellos se pueden observar en la figura anterior: las clases abstractas *Model* y *Controller*, y la interfaz *View*. A continuación se habla de manera general sus diseños.

- *Model* - es una clase abstracta, subclase de *Observable*, lo que la hace tomar el papel de *subject* desde el punto de vista del patrón *observer*. Esto quiere decir que todos los cambios realizados en el modelo se notifican a todos los *observers* que el *subject* tenga registrados. En este caso, los *observers* son implementaciones de la interfaz *View*, y corresponden a las vistas o interfaces gráficas de usuario del sistema. Las vistas se vinculan como *observers* de un modelo a través del método *linkToView(View v)* definido en la clase *Model*. Para el manejo de notificaciones a las vistas, esta clase define una propiedad heredable de tipo entera para almacenar el cambio hecho al modelo. Cada subclase de *Model* debe definir constantes enteras que indiquen los cambios al modelo que requieran notificación a las vistas, y las vistas deben usar esta propiedad heredable para determinar qué cambio ocurrió en el modelo.

- *View* - se trata de una interfaz que se extiende de la interfaz *Observer*, lo que hace que las vistas del sistema tengan el rol de observadores desde el punto de vista del patrón *observer*. La interfaz *View* contiene un método llamado *linkToController(Controller c)*, que cada implementación debe definir, agregando el controlador pasado como parámetro en este método como *listener* de todos los componentes gráficos definidos en la vista y que requieran control de eventos.
- *Controller* - se trata de una clase que contiene a dos atributos, uno de tipo *Model* y otro de tipo *View*, y dos métodos, *linkToModel(Model m)* y *linkToView(View v)*, para inicializarlos. En las subclases de *Controller* se deben implementar interfaces de control de eventos y se debe llevar a cabo la comunicación entre modelo y vista que sea necesaria para mantener consistencia entre dichos componentes.

El cuarto componente importante en el paquete *framework* es la clase *Manager*, que es la que define los atributos y métodos comunes a todos los *managers* del sistema. Como se puede observar, en la figura 4.18, esta clase tiene atributos de tipo *Model*, *View* y *Controller*, y métodos para inicializarlos y vincularlos entre sí. De los dos métodos en esta clase, el método *initComponents()* es abstracto, y debe definirse en cada *manager* codificando la construcción o inicialización de los atributos de tipo *Model*, *View* y *Controller*, además de vinculaciones adicionales a otras vistas, si son requeridas.

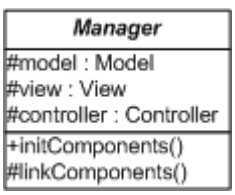

Figura 4.18. Diagrama de clase *Manager*

4.2.2. Manejo de XML

Para implementar la lógica aplicativa relacionada con XML, que es la base de los modelos del sistema, se recurrió al uso del proyecto XMLBeans de Apache:

http://xmlbeans.apache.org/

Este proyecto permite generar librerías de Java en forma de archivos JAR a partir de documentos de validación como *DTD's* o *schemas*. Las librerías generadas contienen una clase de Java por cada *tag* y atributo definido en los documentos de validación. Cada clase ya tiene implementadas operaciones elementales sobre XML, como son adición, eliminación y edición de *tags*, e inclusive hasta consultas en XPath.

XMLBeans consta de un conjunto de librerías y comandos que permiten la generación de librerías de Java para manejar XML. Un comando importante es *scomp*, que es precisamente el comando que genera estas librerías. Este comando requiere como parámetros la ubicación deseada del archivo JAR a generar y el nombre del documento de validación principal a partir del cual se desee formar la librería. En el caso de este proyecto, se utilizó este comando, en conjunto con los *schemas* de los estándares SCORM CAM, IMS CP y LOM, para generar un archivo JAR al que se llamó *ScormCam.jar*, que es el que contiene la lógica aplicativa central del sistema. Para hacer extensiones de estos estándares sólo es necesario hacer cambios en los *schemas* de los estándares correspondientes y volver a generar este JAR, como se indica a continuación:

```
> scomp -out ScormCam.jar imscp_v1p1.xsd
```

4.2.3. Adición de nuevas funcionalidades

Para agregar una nueva funcionalidad a VELOAT generalmente es necesario crear uno o varios *managers*. Para crear un nuevo *manager* es necesario crear clases nuevas en los paquetes *model*, *view*, *controller* y *manager* de algún módulo o paquete principal del programa, o crear un nuevo módulo si se considera conveniente, y vincular el *manager* creado con uno de los existentes modificando su código.

Si se desean agregar funcionalidades que sean auxiliares de uno o varios *managers*, es conveniente agregarlos dentro de paquetes adicionales, como es el caso de los paquetes *tools* existentes. Si se trata de funcionalidades nuevas de lógica aplicativa, se aconseja el uso de clases públicas con métodos estáticos.

En el anexo E se presentan casos concisos de posibles escenarios en donde se pueden requerir agregar nuevas funcionalidades.