Capítulo 5. Implementación realizada

1. Introducción

Esta sección describe la estructura de la implementación realizada de la norma QTI del IMS [43] mediante una **Aplicación Web**. En primer lugar explica el modelo de diseño de la Aplicación Web utilizado (punto 2). Seguidamente se pasa a describir el mapa de las páginas Web de la Aplicación (punto 3). Luego cómo son las fases durante el proceso de creación de un ítem XML o de un test XML de la norma QTI (punto 4). Tras eso se comenta la distribución de los directorios de la Aplicación Web (punto 5), el sistema de tratamiento de errores (punto 6) y la distribución de los paquetes de las clases Java de la Aplicación Web (punto 7). Finalmente se explica el entorno de desarrollo utilizado para la realización del proyecto (punto 8).

2. Modelo de diseño *Modelo 2*

El modelo de diseño que se ha utilizado para implementar la Aplicación Web es el *Modelo 2* o Modelo Vista Controlador (MVC). Comúnmente se considera que es el mejor método para implementar una Aplicación Web usando Servlets y JSPs. La arquitectura del *Modelo 2* se popularizó en la comunidad de Servlet y JSP por el Framework de Jakarta Struts [23], aunque se puede implementar sin él igualmente.

El *Modelo 2* define una separación limpia entre la lógica de negocio de la Aplicación Web y su lógica de presentación. La lógica de negocio consiste en todo lo requerido para construir la información de usuario en tiempo de ejecución. La lógica de presentación consiste en todo lo necesitado para dar el formato que el cliente espera a la información que se le va a presentar. Al separarlas, ambas serán simples y fácilmente manipulables. El *Modelo 2* también es conocido como MVC porque comúnmente la separación implica la creación de un Modelo, una Vista, y un Controlador. El *componente Modelo* es la representación de los datos de la aplicación y el código implicado con la lectura, escritura y validación. El *componente Vista* es responsable de interactuar con el usuario y de la presentación correcta de los datos. El *componente Control* enlaza los otros dos componentes y es responsable de proporcionar una vista adecuada a un usuario y mantener el actual Modelo.

2.1 ¿Por qué utilizar un modelo de diseño?

Hay numerosas razones por las que utilizar un modelo de diseño es favorable. A continuación hay una lista formal de los beneficios:

- **Reduce el tiempo de desarrollo**: Un buen modelo de diseño ayuda a dividir conceptualmente un sistema complejo en tareas manejables. De esta forma un desarrollador se puede centrar en codificar partes individuales de la Aplicación Web, o dividir el trabajo entre los distintos especialistas. También permite modificar partes individuales sin tener que modificar todo el código.
- **Reduce el tiempo de mantenimiento**: Los buenos modelos de diseño permiten planear con antelación para simplificar futuros aspectos de mantenimiento.

- **Colaboración**: Un buen diseño permite separar las funcionalidades de un proyecto en áreas en las que cada uno de los desarrolladores colaboradores es especialista, permitiendo luego unirlas sin problemas.

A medida que crece el tamaño de un proyecto crece la importancia de tener un modelo de diseño. De esta forma, un buen modelo de diseño de una Aplicación Web nunca debería necesitar una revisión completa.

2.2 Implementación realizada del Modelo 2 con Servlet y JSP

Aplicándolo a los Servlets y JSP, el paradigma del Modelo 2 se puede implementar la siguiente manera:

- cada componente Vista se realiza mediante una JSP, ya que proporcionan un método excelente para crear componentes Vista en formato HTML,
- el componente Modelo se ha encapsulado mediante clases Java,
- y el componente Control es un Filtro diseñado para aceptar y dirigir de forma apropiada las peticiones y las respuestas enviadas por el cliente.

En la Figura 5.1 se puede ver el esquema de la implementación del Modelo 2:

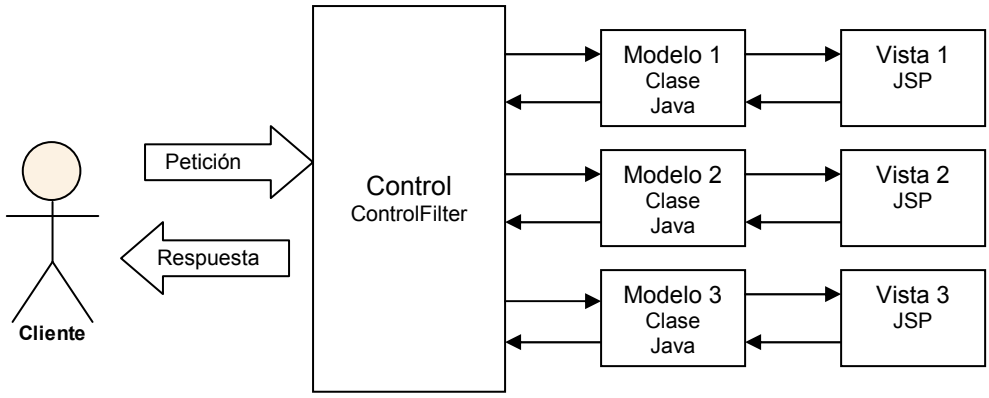


Figura 5.1. Esquema de la implementación del Modelo 2 realizada

Así, en primer lugar, cuando un cliente envía una petición a la Aplicación Web, ésta será capturada y procesada por el filtro de control, `control.ControlFilter`, definido el descriptor de despliegue de la Aplicación, `web.xml`, de la forma siguiente:

```
<filter>
      <filter-name>ControlFilter</filter-name>
      <filter-class>control.ControlFilter</filter-class>
</filter>

<filter-mapping>
      <filter-name>ControlFilter</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

El filtro, como se observa, está configurado para capturar todas las peticiones que lleguen a la Aplicación Web. De esta forma controla todas las peticiones que se produzcan. Su labor es comprobar el nombre del recurso requerido por la petición (basándose en la URL), y buscar una implementación de la interfaz `control.Control`, que tenga ese mismo nombre, pero con la primera letra en mayúsculas, y ejecutar su método `doLogic`, definido en esa interfaz, si existe.

Por lo tanto, todos los componentes Modelo, que contienen la lógica de negocio asociada a cada JSP de la Aplicación, serán clases Java que deben implementar la interfaz `control.Control`. Al hacerlo, contendrán el método `doLogic`, que es donde se ejecuta verdaderamente esa lógica de negocio. Este método recibe los objetos `HttpServletRequest request` y `HttpServletResponse response` con la petición y la respuesta del cliente, para poder realizar su función correctamente.

En el método `doLogic` se preparan los datos que se presentarán en la JSP, principalmente tomando datos procesados y colocándolos en el ámbito de la petición (en request). Al finalizar la ejecución del método, se pasarán la petición y la respuesta (normalmente ya modificadas) al resto de los filtros de la cadena de filtros (si los hubiera), para llegar finalmente a la JSP destino, donde se utilizan los datos generados por el método `doLogic` correspondiente para generar la respuesta HTML dinámica en la JSP que se enviará finalmente al cliente.

Éste es el corazón de la implementación del Modelo 2 en la Aplicación Web, que permite separar perfectamente la lógica de negocio de la lógica de presentación. De esta forma, en las clases Java que implementan la interfaz `control.Control` se tiene toda la lógica de negocio asociada a la Aplicación Web (que a su vez hacen uso de otros paquetes, claro), en donde no se crea nada HTML. Y las páginas JSP sólo se deben preocupar de los detalles de presentación, haciendo uso de las etiquetas estándar y personalizadas, y del EL definido expresamente para su uso en ellas para acceder a los datos elaborados por el Modelo, sin usar ningún elemento de script.

3. Mapa de las páginas de la Aplicación Web

A continuación se muestran 3 figuras con los esquemas de navegación por las páginas Web de la Aplicación. Reflejan los pasos posibles que se pueden dar en una utilización normal y correcta de la Aplicación Web. Están representados todos los enlaces disponibles excepto el de volver al inicio que se encuentra en la cabecera de casi todas las páginas, que se ha obviado por claridad y porque no debería formar parte de un funcionamiento "normal" a la hora de crear un ítem o un test, aunque está disponible.

En primer lugar, en la Figura 5.2 se encuentra el Mapa de las páginas de la Aplicación Web. Ahí están todas las páginas que muestra la Aplicación en el proceso de creación de un ítem o de un test XML. Las páginas que se abren en una nueva ventana de tipo "pop up" se muestran en esquemas a parte.

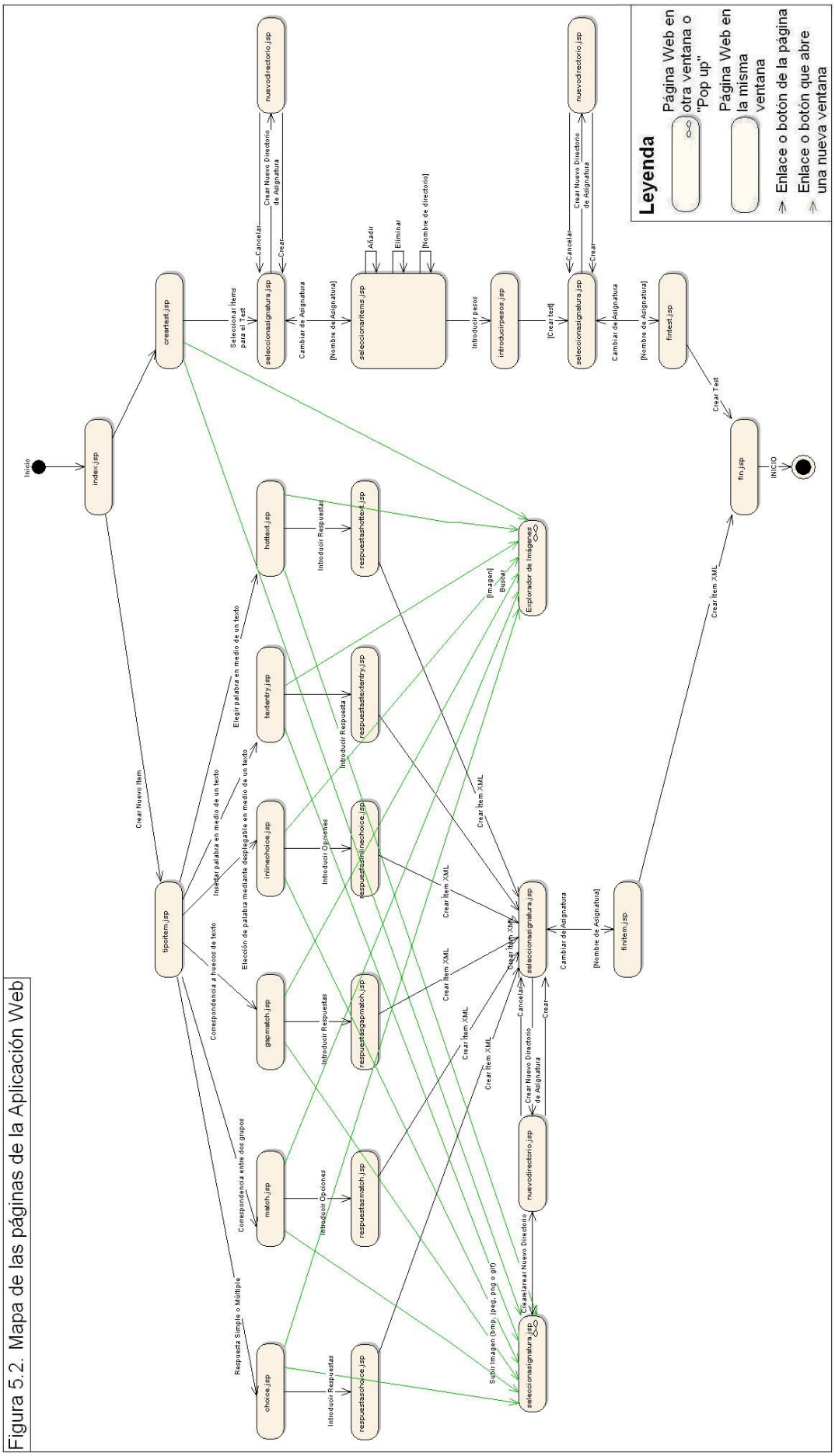


Figura 5.2. Mapa de las páginas de la Aplicación Web

En la figura 5.3 se muestra el mapa de las páginas que se representan en un "pop up" en la Aplicación Web cuya finalidad es explorar las imágenes que contiene el servidor para seleccionar una para incluirla en las instrucciones de un ítem o un test. Y en la Figura 5.4 se muestra el mapa de las páginas que se representan en un "pop up" en la Aplicación Web con la finalidad de subir imágenes al servidor para poder incluirlas luego en las instrucciones de un ítem o un test.

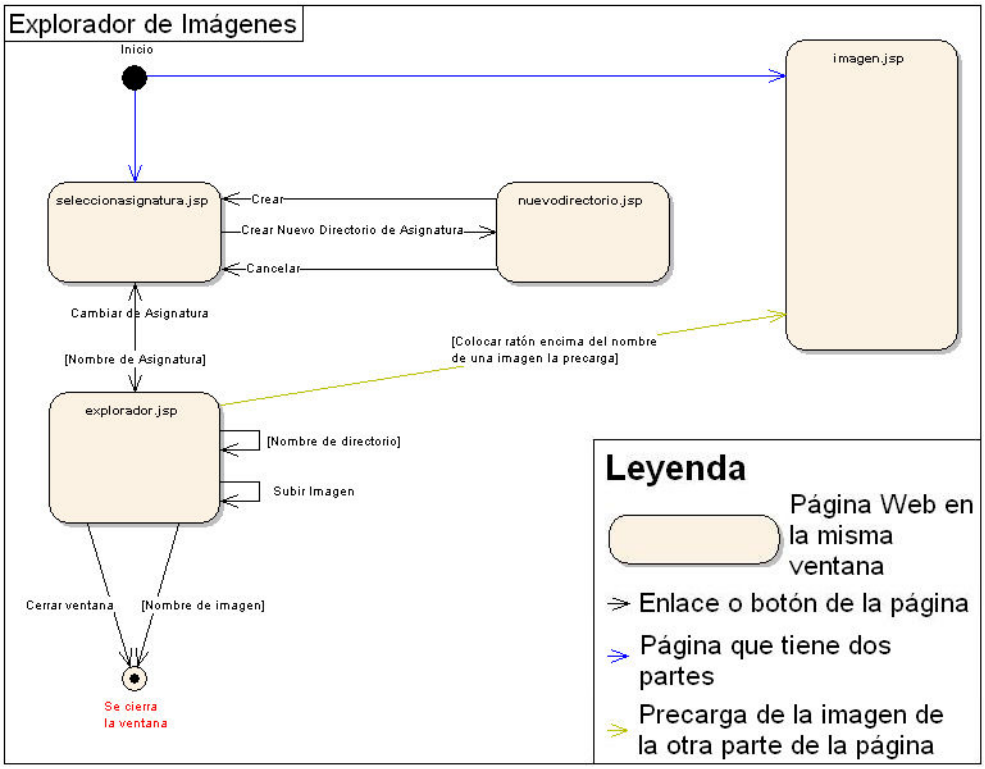


Figura 5.3. Mapa del "pop up" del explorador de imágenes

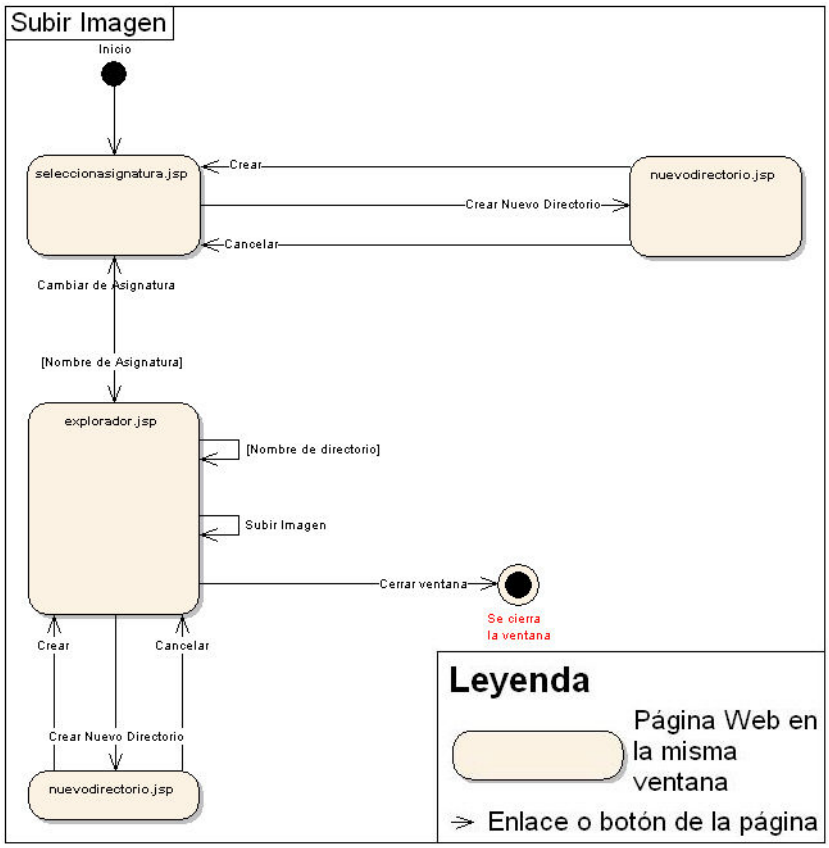


Figura 5.4. Mapa del "pop up" para subir imágenes al servidor

No se encuentran representadas las páginas de error de la Aplicación Web ya que no se sabe a priori cuándo va a aparecer alguna de ellas.

En todas las páginas con campos de formulario está disponible el botón de "Restablecer" para poner a su estado inicial todos los campos.

Para pasar a la página siguiente, normalmente está disponible un botón HTML de tipo "submit", o un enlace, tal y como está descrito en los esquemas de más arriba.

4. Creación de contenido QTI

El método para crear un ítem o un test XML de la norma QTI se puede dividir principalmente en dos fases: la toma de datos y la creación propiamente dicha del archivo XML que lo representa. A continuación se verá en detalle el proceso de creación de un ítem, en primer lugar, y luego de un test.

4.1 Creación de un nuevo ítem

El proceso de creación de un nuevo ítem de la norma QTI se puede dividir en 2 pasos, la toma de datos del ítem y la creación del archivo XML.

4.1.1 Toma de datos

La toma de datos de los ítems se divide en dos fases. En una primera fase se

toman los datos comunes a todos los ítems y los necesarios para construir la segunda página de la toma de datos. Los datos relacionados con un ítem se almacenan en una clase que contiene los campos correspondientes a esos datos. Hay una clase por cada tipo de ítem implementado, con los campos particulares que contengan.

4.1.1.1 Primera Fase de la Toma de datos

La primera fase de la toma de datos se realiza en una JSP con el mismo nombre que el tipo de pregunta, y con extensión `.jsp`, por supuesto. Ahí se recogen los datos comunes siguientes:

- **Identificador** del ítem: definido por la norma QTI, el atributo `identifier` de la clase `assessmentItem`, raíz de todos los ítems. Elemento obligatorio de un ítem. Es una cadena de texto que se utiliza como nombre del archivo XML que representa al ítem, así no habrá dos ítems con el mismo nombre (en el mismo directorio, es decir, no habrá dos ítems con igual URI [9]).
- **Título** del ítem: definido por la norma QTI, el atributo `title`, también de la clase `assessmentItem`. Es una cadena de texto. Elemento obligatorio de un ítem.
- **Instrucciones** del ítem: contenido XHTML [1] que permite la norma QTI (con determinadas restricciones) en el elemento `itemBody`, que es el cuerpo, propiamente dicho, del ítem, donde están los elementos de ayuda al candidato (estas instrucciones), y los elementos de interacción para resolver el ítem.

En esa primera fase también se recogen algunos datos particulares de cada tipo de ítem, necesarios para crear el ítem, o para construir correctamente y con los campos adecuados la segunda página de la toma de datos. A continuación se describen el resto de los datos que se recogen en esta primera fase según cada tipo de ítem:

Ítem tipo Choice

- **Pregunta** del ítem: definido por la norma QTI. Cadena de texto con la pregunta del ítem. Lo contienen los ítems cuya clase fundamental de la interacción herede de la clase `blockInteraction`, que es la que contiene el elemento `prompt`. En este caso, la clase fundamental del ítem es `choiceInteraction`, que efectivamente hereda de `blockInteraction`.
- **Número de respuestas** del ítem: Este campo no se corresponde con ningún atributo ni elemento definido en la normal QTI, pero sí es necesario para conocer de antemano el número de elementos de respuesta que se requerirán en la segunda fase de la toma de datos de este tipo de ítem.
- **¿Mezclar aleatoriamente las respuestas?**: definido por la norma QTI. El valor se recoge en un elemento HTML "checkbox" a marcar por el usuario. Se corresponde con el atributo `shuffle` de la clase fundamental del ítem, `choiceInteraction`, y puede tomar el valor de *true* o *false*, para mezclar aleatoriamente las respuestas cuando se presenten al candidato, respectivamente. Elemento obligatorio de este tipo de ítem.

Ítem tipo Match

- **Pregunta** del ítem: definido por la norma QTI. Cadena de texto con la pregunta del ítem. Lo contienen los ítems cuya clase fundamental de la interacción herede de la clase

`blockInteraction`, que es la que contiene el elemento `prompt`. En este caso, la clase fundamental del ítem es `matchInteraction`, que efectivamente hereda de `blockInteraction`.

- **Número de opciones del grupo 1 (filas)** del ítem: Este campo no se corresponde con ningún atributo ni elemento definido en la normal QTI, pero sí es necesario para conocer de antemano el número de elementos del grupo 1 (número de filas) que se requerirán en la segunda fase de la toma de datos de este tipo de ítem.
- **Número de opciones del grupo 2 (columnas)** del ítem: Este campo no se corresponde con ningún atributo ni elemento definido en la normal QTI, pero sí es necesario para conocer de antemano el número de elementos del grupo 2 (número de columnas) que se requerirán en la segunda fase de la toma de datos de este tipo de ítem.
- **¿Mezclar aleatoriamente las opciones?**: definido por la norma QTI. El valor se recoge en un elemento HTML "checkbox" a marcar por el usuario. Se corresponde con el atributo `shuffle` de la clase fundamental del ítem, `matchInteraction`, y puede tomar el valor de *true* o *false*, para mezclar aleatoriamente las opciones dentro de cada grupo cuando se presenten al candidato, respectivamente. Elemento obligatorio de este tipo de ítem.

Ítem tipo Gap Match

- **Pregunta** del ítem: definido por la norma QTI. Cadena de texto con la pregunta del ítem. Lo contienen los ítems cuya clase fundamental de la interacción herede de la clase `blockInteraction`, que es la que contiene el elemento `prompt`. En este caso, la clase fundamental del ítem es `gapMatchInteraction`, que efectivamente hereda de `blockInteraction`.
- **Número de opciones** del ítem: Este campo no se corresponde con ningún atributo ni elemento definido en la normal QTI, pero sí es necesario para conocer de antemano el número de elementos opción disponibles para rellenar los huecos que se requerirán en la segunda fase de la toma de datos de este tipo de ítem.
- **Número de huecos** del ítem: Este campo no se corresponde con ningún atributo ni elemento definido en la normal QTI, pero sí es necesario para conocer de antemano el número de huecos que contendrá el texto para dividirlo y requerir los identificadores de los huecos en la segunda fase de la toma de datos de este tipo de ítem.
- **¿Mezclar aleatoriamente las opciones?**: definido por la norma QTI. El valor se recoge en un elemento HTML "checkbox" a marcar por el usuario. Se corresponde con el atributo `shuffle` de la clase fundamental del ítem, `gapMatchInteraction`, y puede tomar el valor de *true* o *false*, para mezclar aleatoriamente las opciones dentro de cada grupo cuando se presenten al candidato, respectivamente. Elemento obligatorio de este tipo de ítem.

Ítem tipo Inline Choice

- **Número de opciones posibles** del ítem: Este campo no se corresponde con ningún atributo ni elemento definido en la normal QTI, pero sí es necesario para conocer de antemano el número de elementos de opciones disponibles para el hueco que se requerirán en la segunda fase de la toma de datos de este tipo de ítem.
- **¿Mezclar aleatoriamente las opciones?**: definido por la norma QTI. El valor se recoge en un elemento HTML "checkbox" a marcar por el usuario. Se corresponde con el atributo `shuffle` de la clase fundamental del ítem, `inlineChoiceInteraction`, y

puede tomar el valor de *true* o *false*, para mezclar aleatoriamente las opciones dentro de cada grupo cuando se presenten al candidato, respectivamente. Elemento obligatorio de este tipo de ítem.

Ítem tipo Text Entry

Para este tipo de ítem no se pide ningún dato adicional, a parte de los comunes, ya que no hay elementos que mezclar, ni hace falta saber de antemano ningún valor para construir la página de la segunda parte de la toma de datos de este ítem. La clase fundamental de este tipo de ítem, `textEntryInteraction`, no tiene ningún valor obligatorio adicional que se necesite recoger aquí.

Ítem tipo Hot Text

- **Pregunta** del ítem: definido por la norma QTI. Cadena de texto con la pregunta del ítem. Lo contienen los ítems cuya clase fundamental de la interacción herede de la clase `blockInteraction`, que es la que contiene el elemento `prompt`. En este caso, la clase fundamental del ítem es `hottextInteraction`, que efectivamente hereda de `blockInteraction`.
- **Número de Hot Texts** del ítem: Este campo no se corresponde con ningún atributo ni elemento definido en la normal QTI, pero sí es necesario para conocer de antemano el número de elementos "hottext" que contendrá el texto para dividirlo y requerir los "hottext" en la segunda fase de la toma de datos de este tipo de ítem.

Comprobaciones de la Primera Fase de la Toma de datos

En esta primera fase se comprueba que se hayan introducido al menos todos los datos obligatorios de la página, que los números de elementos introducidos sean números enteros válidos y mayores que uno, que los identificadores no contengan caracteres reservados del sistema y que las instrucciones sean código XML correcto.

4.1.1.2 Segunda Fase de la Toma de datos

En la segunda fase se recogen los datos que ya son más concretos de cada tipo de ítem. Muchos campos mostrados en esta fase dependen de los datos introducidos en la primera fase. Las páginas Web que recogen los datos de la segunda fase de la toma de datos tienen un nombre que empieza por "`respuestas`" seguido del nombre del tipo de pregunta, con extensión `.jsp` por supuesto, puesto que son JSPs.

A continuación se ve la segunda fase de la toma de datos para cada tipo de ítem:

Ítem tipo Choice

- **Número máximo de elecciones del candidato**: definido por la norma QTI, el atributo `maxChoices` de la clase básica de una interacción de tipo Choice, `choiceInteraction`. Elemento obligatorio de este ítem.
- **Respuesta X** del ítem: definido por la norma QTI, es el contenido de cada uno de los elementos `simpleChoice` que contiene la clase `choiceInteraction`. Elemento obligatorio de este tipo de ítem. Habrá tantos elementos como respuestas se hayan indicado en la primera página de la toma de datos de este tipo de ítem.

- **Identificador**: definido por la norma QTI, es el atributo `identifier` del elemento `choice`, del cual heredan los elementos `simpleChoice`, que representan cada una de las respuestas posibles del ítem. Atributo obligatorio de este elemento. Habrá tantos elementos como respuestas posibles. Se inicializa a un valor válido por defecto.
- **¿Correcta?**: no se corresponde directamente con ningún elemento de la norma QTI, pero se deben marcar las respuestas correctas, para definirlas en la declaración de la variable de respuesta. Habrá tantos elementos como respuestas posibles.
- **¿Fija?**: definido por la norma QTI, es el atributo `fixed` del elemento `choice`, del cual hereda el elemento `simpleChoice`, que representa cada una de las respuestas posibles del ítem. Atributo opcional de este elemento. Habrá tantos elementos como respuestas posibles. Este elemento sólo está disponible si se ha marcado en la primera página de la toma de datos del ítem que se mezclen aleatoriamente las respuestas, ya que en caso contrario no tendría sentido.

Comprobaciones realizadas:

Se comprueba que se hayan introducido todos los parámetros, que el número de elecciones sea un número entero mayor o igual que el número de respuestas marcadas como correctas o cero, que se haya marcado al menos una respuesta como correcta y que los identificadores sean identificadores válidos según el tipo `identifier` definido en la norma QTI y diferentes entre sí.

Ítem tipo Match

- **Número máximo de asociaciones totales**: definido por la norma QTI, el atributo `maxAssociations` de la clase básica de una interacción de tipo Match, `matchInteraction`. Elemento obligatorio de este ítem.
- **Opción 1.X** del ítem: definido por la norma QTI, es el contenido de cada uno de los elementos `simpleAssociableChoice` que contiene el primer elemento de la clase `simpleMatchSet` que contiene la clase `matchInteraction` (cada `simpleMatchSet` representa cada uno de los dos grupos de opciones que contiene este tipo de ítem). Elemento obligatorio de este tipo de ítem. Habrá tantos elementos como se haya indicado en la primera página de la toma de datos de este tipo de ítem para el número de opciones del grupo 1.
- **Opción 2.X** del ítem: definido por la norma QTI, es el contenido de cada uno de los elementos `simpleAssociableChoice` que contiene el segundo elemento de la clase `simpleMatchSet` que contiene la clase `matchInteraction` (cada `simpleMatchSet` representa cada uno de los dos grupos de opciones que contiene este tipo de ítem). Elemento obligatorio de este tipo de ítem. Habrá tantos elementos como se haya indicado en la primera página de la toma de datos de este tipo de ítem para el número de opciones del grupo 2.
- **Identificador**: definido por la norma QTI, es el atributo `identifier` del elemento `choice`, del cual hereda el elemento `associableChoice`, del cual hereda el elemento `simpleAssociableChoice`, que representa cada una de las opciones que contiene cada uno de los dos grupos del ítem. Atributo obligatorio de este elemento. Habrá tantos elementos como opciones totales posibles. Se inicializa a un valor válido por defecto.
- **Máximas asociaciones**: definido por la norma QTI, el atributo `matchMax` de la clase básica de cada opción del ítem, `simpleAssociableChoice`. Elemento obligatorio de este ítem.

- **¿Combinación Correcta?**: no se corresponde directamente con ningún elemento de la norma QTI, pero se deben marcar las opciones correctas, para definirlas en la declaración de la variable de respuesta. Habrá tantos elementos como combinaciones posibles.
- **¿Fija?**: definido por la norma QTI, es el atributo `fixed` del elemento `choice`, del cual hereda el elemento `associableChoice`, del cual hereda el elemento `simpleAssociableChoice`, que representa cada una de las respuestas posibles del ítem. Atributo opcional de este elemento. Habrá tantos elementos como respuestas posibles. Este elemento sólo está disponible si se ha marcado en la primera página de la toma de datos del ítem que se mezclen aleatoriamente las opciones, ya que en caso contrario no tendría sentido.

Comprobaciones realizadas:

Se comprueba que se hayan introducido todos los parámetros, que el número de elecciones totales sea un número entero mayor o igual que el número de combinaciones totales marcadas como correctas o cero, que se haya marcado al menos una respuesta como correcta en cada fila, que los números máximos de asociaciones de cada fila y columna sean mayores que el número de combinaciones correctas marcadas en ella o cero, y que los identificadores sean identificadores válidos según el tipo `identifier` definido en la norma QTI y diferentes entre sí.

Ítem tipo Gap Match

- **Texto X** del ítem: Elementos que se corresponden con el contenido del elemento `blockStatic` del cuerpo del ítem. Por razones de organización, se ha utilizado su subclase `blockquote` para incluir los textos y los huecos (a la hora de escribir en el archivo XML el ítem). Aunque es un elemento obligatorio de este ítem, no es obligatorio que los textos sean distintos de vacío.
- **Opción X** del ítem: definido por la norma QTI, es el contenido de cada uno de los elementos `gapText`, subclase de `gapChoice`, que contiene la clase `gapMatchInteraction` (cada `gapChoice` representa cada opción que contiene este tipo de ítem). Elemento obligatorio de este tipo de ítem. Habrá tantos elementos como opciones se haya indicado en la primera página de la toma de datos de este tipo de ítem.
- **Identificador** (para los huecos): definido por la norma QTI, es el atributo `identifier` del elemento `choice`, del cual hereda el elemento `associableChoice`, del cual hereda el elemento `gap`, que representa cada uno de los huecos que contiene el texto. La clase `gap` va en medio del texto en las posiciones en las que van los huecos definidos por el usuario. Atributo obligatorio de este elemento. Habrá tantos elementos como huecos. Se inicializa a un valor válido por defecto.
- **Identificador** (para las opciones): definido por la norma QTI, es el atributo `identifier` del elemento `choice`, del cual hereda el elemento `associableChoice`, del cual hereda el elemento `gapChoice`, que representa cada una de las opciones que hay disponibles. Atributo obligatorio de este elemento. Habrá tantos elementos como opciones. Se inicializa a un valor válido por defecto.
- **Máximas asociaciones**: definido por la norma QTI, el atributo `matchMax` de la clase básica de cada opción del ítem, `gapChoice`. Elemento obligatorio de este ítem.
- **¿Combinación Correcta?**: no se corresponde directamente con ningún elemento de la norma QTI, pero se deben marcar las opciones correctas, para definirlas en la

declaración de la variable de respuesta. Habrá tantos elementos como combinaciones posibles.

- **¿Fija?**: definido por la norma QTI, es el atributo `fixed` del elemento `choice`, del cual hereda el elemento `associableChoice`, del cual hereda el elemento `gapChoice`, que representa cada una de las opciones disponibles para rellenar los huecos. Atributo opcional de este elemento. Habrá tantos elementos como respuestas posibles. Este elemento sólo está disponible si se ha marcado en la primera página de la toma de datos del ítem que se mezclen aleatoriamente las opciones, ya que en caso contrario no tendría sentido.

Comprobaciones realizadas:

Se comprueba que se hayan introducido todos los parámetros, que se haya marcado al menos una respuesta como correcta para cada hueco, que los números máximos de asociaciones de cada columna sea mayor que el número de combinaciones correctas marcadas en ella o cero, y que los identificadores sean identificadores válidos según el tipo `identifier` definido en la norma QTI y diferentes entre sí.

Ítem tipo Inline Choice

- **Texto X** del ítem: Elementos que se corresponden con el contenido del elemento `blockStatic` del cuerpo del ítem. Por razones de organización, se ha utilizado su subclase `blockquote` (a la hora de escribir en el archivo XML el ítem), en el que se incluye un elemento XHTML `p` para agrupar el texto en un párrafo. Aunque es un elemento obligatorio de este ítem (ya que en medio del texto va la propia interacción), no es obligatorio que los textos sean distintos de vacío.
- **Opción X** del ítem: definido por la norma QTI, es el contenido de cada uno de los elementos `inlineChoice`, subclase de `choice`, que contiene la clase `inlineChoiceInteraction` (cada `inlineChoice` representa cada opción que contiene este tipo de ítem). Elemento obligatorio de este tipo de ítem. Habrá tantos elementos como opciones se haya indicado en la primera página de la toma de datos de este tipo de ítem.
- **Identificador**: definido por la norma QTI, es el atributo `identifier` del elemento `choice`, del cual hereda el elemento `inlineChoice`, que representa cada una de las opciones del ítem. La clase `inlineChoiceInteraction`, que representa a este tipo de interacción, va en medio del texto en la posición en la que van las opciones. Atributo obligatorio de este elemento. Habrá tantos elementos como opciones. Se inicializa a un valor válido por defecto.
- **¿Correcta?**: no se corresponde directamente con ningún elemento de la norma QTI, pero se debe marcar la opción correcta, para definirla en la declaración de la variable de respuesta. Habrá tantos elementos como opciones posibles.
- **¿Fija?**: definido por la norma QTI, es el atributo `fixed` del elemento `choice`, del cual hereda el elemento `inlineChoice`, que representa cada una de las opciones disponibles para rellenar los huecos. Atributo opcional de este elemento. Habrá tantos elementos como respuestas posibles. Este elemento sólo está disponible si se ha marcado en la primera página de la toma de datos del ítem que se mezclen aleatoriamente las opciones, ya que en caso contrario no tendría sentido.

Comprobaciones realizadas:

Se comprueba que se hayan introducido todos los parámetros, que se haya marcado una opción como correcta, y que los identificadores sean identificadores válidos según el tipo `identifier` definido en la norma QTI y diferentes entre sí.

Ítem tipo Text Entry

- **Texto X** del ítem: Textos que rodean al cuadro de introducción de la respuesta. Van en el cuerpo del ítem, en la clase `itemBody`. Por razones de organización, van dentro de una clase `blockquote` (a la hora de escribir en el archivo XML el ítem), en el que se incluye un elemento XHTML `p` para agrupar ambos textos y el cuadro de introducción de la respuestas en un párrafo. Aunque es un elemento obligatorio de este ítem (ya que en medio del texto va la propia interacción), no es obligatorio que los textos sean distintos de vacío.
- **Respuesta correcta** del ítem: no se corresponde directamente con ningún elemento de la norma QTI, pero se debe escribir la respuesta correcta, para definirla en la declaración de la variable de respuesta.
- **Longitud esperada**: definido por la norma QTI, es el atributo `expectedLength` del elemento `stringInteraction`, del cual hereda el elemento `textEntryInteraction`, que representa este tipo de interacción. Atributo opcional de este elemento.

Comprobaciones realizadas:

Se comprueba que se haya introducido al menos la respuesta correcta.

Ítem tipo Hot Text

- **Número máximo de elecciones** del ítem: definido por la norma QTI, el atributo `maxChoices` de la clase básica de una interacción de tipo Hot Text, `hottextInteraction`. Elemento obligatorio de este ítem.
- **Texto X** del ítem: Elementos que se corresponden con el contenido del elemento `blockStatic` del cuerpo del ítem (a la hora de escribir en el archivo XML el ítem). Por razones de organización, se ha utilizado su subclase, un elemento XHTML, `p`, para agrupar el texto en un párrafo. Aunque es un elemento obligatorio de este ítem (ya que en medio del texto va la propia interacción), no es obligatorio que los textos sean distintos de vacío.
- **Hot Text X** del ítem: definido por la norma QTI, es el contenido de cada uno de los elementos `hottext`, que contiene la clase `hottextInteraction` (cada `hottext` representa cada texto a elegir que contiene este tipo de ítem). Elemento obligatorio de este tipo de ítem. Habrá tantos elementos como se haya indicado en la primera página de la toma de datos de este tipo de ítem.
- **Identificador**: definido por la norma QTI, es el atributo `identifier` del elemento `choice`, del cual hereda el elemento `hottext`, que representa cada una de las opciones del ítem. La clase `hottext` va en medio del texto en la posición en la que van cada uno de los Hot Texts a seleccionar. Atributo obligatorio de este elemento. Habrá tantos elementos como elementos Hot Text. Se inicializa a un valor válido por defecto.
- **¿"Hot Text" Correcto?**: no se corresponde directamente con ningún elemento de la norma QTI, pero se debe marcar la opción correcta, para definirla en la declaración de la variable de respuesta. Habrá tantos elementos como opciones posibles.

Comprobaciones realizadas:

Se comprueba que se hayan introducido todos los parámetros obligatorios, que se haya marcado al menos una opción como correcta, que el números máximo de elecciones sea mayor que el número de opciones correctas marcadas o cero, y que los identificadores sean identificadores válidos según el tipo `identifier` definido en la norma QTI y diferentes entre sí.

4.1.2 Creación del archivo XML del ítem

Para crear el archivo XML, primero se debe seleccionar la localización final del ítem. Se selecciona la asignatura en una página que lista todas las disponibles, pudiendo crear nuevos directorios de asignatura en la propia página. Dentro del subdirectorio de los ítems se pueden crear nuevos subdirectorios a su vez, y navegar por ellos ya que se listan enlaces a los ya disponibles y para volver atrás.

La creación propiamente dicha, tiene lugar en una JSP llamada `finitem.jsp`. Cuando se va a crear el ítem XML en la localización deseada dentro de la asignatura elegida, se comprueba que no exista ya uno con ese mismo identificador (mismo nombre de archivo), y si es así, se requiere un nuevo identificador para el ítem. Si finalmente ese nuevo identificador ya es correcto y válido, se pasa a crear el archivo XML del ítem QTI.

Las clases que representan a cada uno de los ítems contienen un método para escribir el ítem XML en un archivo.

En primer lugar se escribe el elemento raíz de los ítems, con los atributos correspondientes, como el título y el identificador del ítem, entre otros. Este elemento es común a todos los ítems.

A continuación se declara la variable de respuesta, cuyo nombre debe ser `RESPONSE`, ya que así está definida en la plantilla de respuesta que se va a utilizar. Esta variable es la que contendrá la respuesta del candidato al ítem. Se define del tipo concreto de la variable de respuesta en función del tipo de respuesta que se espere: `identifier`, si se espera un identificador; `directedPair`, si se espera una pareja de identificadores; o tipo `string`, si se espera una cadena de texto. Otro parámetro es la cardinalidad de la variable de respuesta, que depende del tipo de pregunta, y del número de respuestas que se le permite al candidato introducir. También se define cuál es su valor correcto, para poder corregir y puntuar al candidato.

Luego se define la variable de puntuación, que será siempre de tipo numérico y entero, con un valor por defecto de 0. Al ser de tipo numérico, la variable se debe llamar `SCORE` (este nombre es obligatorio por la norma QTI) y para poder utilizar la plantilla de corrección declarada al final del ítem, ya que ésta establece su valor en función de las respuestas dadas por el candidato.

Seguidamente se encuentra el cuerpo del ítem, representado por el elemento `itemBody`, que es el que contiene los elementos necesarios para la interacción con el candidato, las instrucciones, la pregunta (si la hay), y demás elementos necesarios para la correcta representación del ítem. Aquí se escribe el elemento interacción, que es el particular de cada tipo de pregunta.

Finalmente se incluye el proceso de respuesta del ítem. Aquí simplemente se hace referencia a la plantilla utilizada para corregir el ítem, `match_response.xml`, cuya URI relativa está definida como constante en el archivo descriptor de despliegue de la Aplicación Web, `web.xml`. Esta plantilla recoge el valor de la variable `RESPONSE`, que contiene la respuesta del candidato, y en función de ella establece el valor de la variable de puntuación `SCORE`, variable con la puntuación final del ítem. Ambas variables están definidas arriba en el ítem. Finalmente se cierran todos los elementos y así se tiene el archivo XML del ítem.

4.2 Creación de un test

El proceso de creación de un nuevo test de la norma QTI se puede dividir en 2 pasos, la toma de datos del test y la creación del archivo XML.

4.2.1 Toma de datos

La toma de datos del test se divide en tres fases. En una primera fase se toman los datos básicos y algunos necesarios para construir la tercera página de la toma de datos del test. En una segunda fase se seleccionan los ítems que el test va a incluir, también necesario para la tercera fase. Y en esa tercera fase se le dan a los ítems los pesos y las propiedades finales. Los datos relacionados con el test se almacenan en una serie de clases que contienen los campos correspondientes a esos datos. La estructura de clases es la equivalente a la definida en la norma QTI, con tipos Java equivalentes; de esta forma, es mucho más fácil ampliar la Aplicación añadiendo nuevas funcionalidades para el test definidas por la norma pero no implementadas aún.

4.2.1.1 Primera Fase de la Toma de datos

La primera fase de la toma de datos se realiza en una JSP llamada `creartest.jsp`. Ahí se recogen los datos comunes siguientes:

- **Identificador del Test**: definido por la norma QTI, el atributo `identifier` de la clase `assessmentTest`, elemento raíz del test. Atributo obligatorio de un test. Se utiliza como nombre del archivo XML que representa al test, así no habrá dos con el mismo nombre (en el mismo directorio, es decir, no habrá dos ítems con igual URI).
- **Título del test**: definido por la norma QTI, el atributo `title`, también de la clase `assessmentTest`. Atributo obligatorio de un test.
- **Instrucciones** del test: contenido XHTML que permite la norma QTI (con determinadas restricciones) en el elemento `assessmentSection`, que representa una sección (una agrupación de ítems o de subsecciones) de una parte del test, `testPart` (agrupación de secciones). Las instrucciones contienen elementos de ayuda al candidato. Cada `assessmentSection` contiene sus propias instrucciones, pero esta implementación sólo contempla que el test contenga una parte, que contiene sólo una sección, así que sólo hay unas instrucciones para el test.
- **Permitir revisar las respuestas del test al finalizar**: elemento que corresponde con el atributo `allowReview` de la clase `itemSessionControl`. En la implementación realizada sólo la clase `testPart` contiene un elemento `itemSessionControl` para controlar los estados por los que puede pasar la sesión del candidato en la parte del test. Sólo se ha implementado el paso al estado de revisión, para que el candidato pueda

revisar el test realizado y ver las respuestas que ha dado, una vez que haya finalizado de introducir todas las respuestas.

- **Número de ítems a seleccionar del conjunto**: es el atributo `select` del elemento `selection`, elemento que contiene la clase `assessmentSection`. Representa el número de ítems que se van a presentar al candidato cuando vaya a realizar los ítems de esa sección. El otro atributo del elemento `selection`, `withReplacement`, siempre tendrá el valor de `false` en esta implementación, ya que no tiene sentido repetir algún ítem en el test cuando los ítems no contienen variables de plantilla que haga que distintas instancias del mismo ítem sean distintas.
- **¿Mezclar aleatoriamente los ítems?**: es el atributo `shuffle` del elemento `ordering`, elemento que contiene la clase `assessmentSection`. Representa si se deben mezclar aleatoriamente los ítems de esa sección.

Comprobaciones de la Primera Fase de la Toma de datos

En esta primera fase las comprobaciones que se realizan son que se hayan introducido al menos todos los datos obligatorios de la página, que el identificador no contenga caracteres reservados del sistema y que si se ha introducido un número de ítems a seleccionar del conjunto, que sea un número entero positivo válido o cero.

4.2.1.2 Segunda Fase de la Toma de datos

Esta fase tiene lugar en una JSP llamada `seleccionaritems.jsp`. Ahí se pueden seleccionar los ítems que se desean incluir en el test. Se puede navegar por los subdirectorios del directorio de ítems de la asignatura, y cambiar de una asignatura a otra. Así se pueden seleccionar para añadir los ítems que se deseen, viendo en todo momento los que ya se han añadido.

Con esos ítems se crea una tabla en el programa con los ítems que se van a añadir al test Los ítems se terminarán de añadir al objeto que representa a la clase `assessmentSection` (sección de test) una vez que se tengan sus pesos y demás datos sobre ellos, que se toman en la tercera fase de la toma de datos. Mientras tanto, la tabla de ítems a añadir al test se almacena en la sesión del usuario.

Comprobaciones de la Segunda Fase de la Toma de datos

En esta segunda fase se comprueba que se haya seleccionado algún ítem, y si se introdujo en la primera fase de la toma de datos algún número de ítems a seleccionar del conjunto, que el número de ítems seleccionados para el test sea mayor o igual que ese número. No se comprueba que se hayan repetido ítems, aunque no tiene sentido repetirlos, ya que le aparecerían al candidato dos ítems exactamente iguales, pero se deja la posibilidad ya que la norma lo permite, y por si en un futuro se implementan plantillas de ítems, por ejemplo.

4.2.1.3 Tercera Fase de la Toma de datos

La tercera fase de la toma de datos se lleva a cabo en la JSP llamada `introducirpesos.jsp`. Ahí se recogen los datos siguientes:

- **Identificador** de la referencia al ítem en el test: definido por la norma QTI, el atributo `identifier` de la clase `sectionPart`, del cual hereda la clase `assessmentItemRef`, que representa cada una de las referencias a los ítems que incluye

el test. Elemento obligatorio de la referencia a un ítem. Habrá tantos elementos como ítems se van a incluir en el test. Se inicializa a un valor válido por defecto.

- **Peso Correcto** del ítem en el test: definido por la norma QTI, es el atributo `value` (o el contenido de la clase, como sale en la guía de implementación y como finalmente se ha decidido implementar) de la clase `weight`, que representa un peso de la puntuación del ítem dentro del test. El objeto `weight` correspondiente con este peso tiene el identificador `WeightOK`. De esta forma en el proceso de corrección del test se utiliza este peso cuando el ítem es correcto. Habrá tantos elementos como ítems se van a incluir en el test. Elemento obligatorio del peso de una referencia a un ítem. Un peso no es obligatorio para una referencia a un ítem, pero esta implementación asume en el proceso de corrección del test que todos los ítems tienen un peso que se usará cuando el ítem sea correcto. Si se deja este valor en blanco, la implementación asumirá un peso de 1 cuando el ítem sea correcto.
- **Peso Incorrecto** del ítem en el test: definido por la norma QTI, es el atributo `value` (o el contenido de la clase, como sale en la guía de implementación y como finalmente se ha decidido implementar) de la clase `weight`, que representa un peso de la puntuación del ítem dentro del test. El objeto `weight` correspondiente con este peso tiene el identificador `WeightNOK`. De esta forma en el proceso de corrección del test se utiliza este peso cuando el ítem no es correcto. Habrá tantos elementos como ítems se van a incluir en el test. Elemento obligatorio del peso de una referencia a un ítem. Un peso no es obligatorio para una referencia a un ítem, pero esta implementación asume en el proceso de corrección del test que todos los ítems tienen un peso que se usará cuando el ítem no sea correcto. Si se deja este valor en blanco, la implementación asumirá un peso de 0 cuando el ítem no sea correcto.
- **Fija**: definido por la norma QTI, es el atributo `fixed` de la clase abstracta `SectionPart`, de la cual hereda la clase `assessmentItemRef`, que representa cada una de las referencias a los ítems que incluye el test. Atributo opcional de este elemento. Habrá tantos elementos como ítems incluya el test. Este elemento sólo está disponible si se ha marcado en la primera página de la toma de datos del ítem que se mezclen aleatoriamente los ítems, ya que en caso contrario no tendría sentido.
- **Requerido**: definido por la norma QTI, es el atributo `required` de la clase abstracta `SectionPart`, de la cual hereda la clase `assessmentItemRef`, que representa cada una de las referencias a los ítems que incluye el test. Atributo opcional de este elemento. Habrá tantos elementos como ítems incluya el test. Este elemento sólo está disponible si se ha marcado en la primera página de la toma de datos del ítem que se seleccionen algunos de los ítems del conjunto cuando se vayan a presentar al candidato, ya que en caso contrario no tendría sentido.

Comprobaciones de la Tercera Fase de la Toma de datos

En esta tercera fase se comprueba que se hayan introducido todos los identificadores, que sean identificadores válidos según el tipo `identifier` definido en la norma QTI y diferentes entre sí. Los números de los pesos que se hayan introducido deben ser números decimales válidos, y positivos en el caso de los pesos correctos (los incorrectos que se introduzcan positivos se considerarán negativos).

4.2.2 Creación del archivo XML del test

Para crear el archivo XML, primero se debe seleccionar la localización final del ítem. Se selecciona la asignatura en una página que lista todas las disponibles, pudiendo crear nuevos directorios de asignatura en la propia página. Dentro del subdirectorio de

las baterías de test se pueden crear nuevos subdirectorios a su vez, y navegar por ellos, ya que se listan los disponibles.

La creación propiamente dicha, tiene lugar en una JSP llamada `fintest.jsp`. Cuando se va a crear el test XML en la localización deseada dentro de la asignatura elegida, se comprueba que no exista ya uno con ese mismo identificador (mismo nombre de archivo), y si es así, se requiere un nuevo identificador para el test. Si finalmente ese nuevo identificador ya es correcto y sus caracteres válidos, se tendrá la localización definitiva del test.

Una vez que se sabe la localización final del test, se crean las referencias a los ítems. Ya que éstas son URIs [9] relativas, hasta que no se sabe la localización final del test no se pueden crear. A partir de las URIs absolutas del test y los ítems, se calculan las URIs relativas de los ítems, que son las que incluirá el archivo XML del test para hacer referencia a ellos. Una vez que se tienen todos los datos se pasa a crear el archivo XML del test QTI.

La estructura de clases que representa al test es la equivalente con clases Java a la definida por la norma QTI. Cada una de estas clases contiene un método, cuyo nombre comienza por `crea`, seguido del nombre de la misma clase. Así, para escribir en el archivo el test XML sólo hay que llamar al método que escribe el elemento raíz del test,

```
public MensajeEstado creaAssessmentTest(java.lang.String path,
            ServletContext sc) throws java.io.IOException
```

, y este método escribe los atributos y elementos que contiene, y llama a los métodos que escriben cada una de las clases que contiene que no es capaz de escribir en el archivo por sí mismo.

En primer lugar se escribe el elemento raíz del test, con los atributos correspondientes, como el título y el identificador del test, entre otros. Luego se define la variable de puntuación, que será siempre de tipo numérico y decimal, con un valor inicial por defecto de 0. Al ser de tipo numérico, la variable se debe llamar `SCORE` (este nombre es obligatorio por la norma QTI y se hace referencia a esa variable en el proceso de puntuación al final del test, ya que ahí se establece su valor).

También se definen dos variables más: una variable llamada `CORRECTO`, y otra llamada `INCORRECTO`, de valores 1.0 y -1.0, respectivamente. Se utilizan en el proceso de puntuación del test para compararlas con el resultado de la puntuación de un ítem, para saber si éste está correcto o incorrecto, y aplicar cada peso en consecuencia.

A continuación se escribe la parte del test, ya que la implementación sólo soporta una parte dentro de un test, con un identificador válido predefinido en el programa. El modo de navegación se establece a no lineal (se pueden hacer los ítems en cualquier orden), y el modo de presentación en simultáneo (se entregan las respuestas de los ítems todas a la vez cuando se hayan respondido), ya que es la única forma implementada por el motor de entrega al candidato.

A continuación, antes de la sección de la parte del test, se escribe el objeto de control de sesión, diciendo si se le permite al candidato revisar las respuestas que ha dado tras responder el test o no.

Luego se escribe la única sección que contiene la parte del test, con un identificador válido predefinido en el programa. Se establece como una sección "visible", con el título, las instrucciones, el método de ordenación y de selección introducidos por el usuario.

Dentro de la sección se escriben las referencias a cada uno de los ítems. Cada referencia lleva su identificador, la URI relativa de referencia al ítem, si es un elemento requerido, si es un elemento fijo, y cada uno de los dos pesos asociados al ítem (cada uno con su identificador y su valor).

Finalmente se escribe el proceso de puntuación del test. Este proceso realiza lo siguiente por cada uno de los ítems que contiene el test:

- Si el valor de la variable `SCORE` del ítem es igual al valor de la variable `CORRECTO` definida al comienzo del test, entonces al valor de la variable `SCORE` del test se le suma el valor de la variable `SCORE` del ítem por su peso de identificador `WeightOK`.
- Si no se cumple la condición anterior se comprueba si el valor de la variable `SCORE` del ítem es igual al valor de la variable `INCORRECTO` definida al comienzo del test, entonces al valor de la variable `SCORE` del test se le suma el valor de la variable `SCORE` del ítem por su peso de identificador `WeightNOK`.

Finalmente se cierran todos los elementos y así se tiene el archivo XML del test.

5. Organización de los archivos de la Aplicación Web

Este punto va a explicar cómo está organizado físicamente el directorio en el que se encuentra la Aplicación Web, explicando qué subdirectorios se pueden encontrar y qué contiene cada uno.

5.1 Directorio raíz

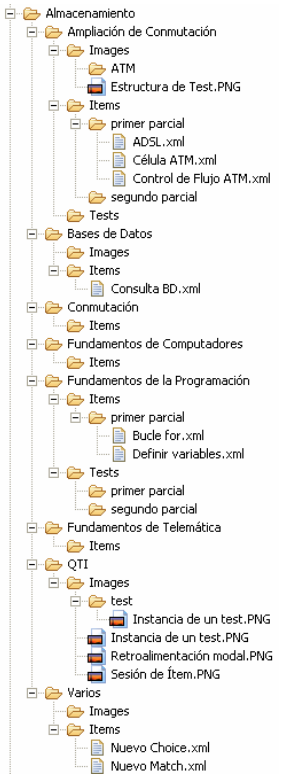
En el directorio raíz se encuentran tan sólo dos tipos de archivos:

- Todos los archivos fuente de todas las JSPs de la Aplicación están en el directorio raíz directamente.
- Imágenes que se incluyen en las JSPs, que son las siguientes: la imagen del icono de una imagen, la del icono de un directorio, la imagen de la cabecera de casi todas las JSPs, y una imagen en blanco para el explorador de imágenes para que cuando no se previsualiza ninguna otra imagen del servidor no aparezca un error de imagen vacía o no encontrada.

A continuación se enumeran los subdirectorios que contiene el directorio raíz y su contenido.

5.1.1 Subdirectorio “Almacenamiento”

Aquí se guardan todos los archivos de imagen que se incluyan en las instrucciones, los ítems, y las baterías de test de cada asignatura, que se creen mediante la Aplicación.



De este directorio *cuelgan* todos los directorios de asignaturas, un directorio para cada una. En el ejemplo de la Figura 5.5 se pueden observar las siguientes asignaturas: *Ampliación de Conmutación*, *Bases de Datos*, *Conmutación*, *Fundamentos de Computadores*, *Fundamentos de la Programación*, *Fundamentos de Telemática*, *QTI*, y *Varios*.

Dentro de cada directorio de asignatura se encuentran a su vez tres subdirectorios: *Images*, *Items*, y *Tests*. Estos directorios son los que contendrán, respectivamente, los archivos de imágenes, de los ítems, y de la batería de test correspondientes a esa asignatura.

A su vez, estos tres directorios pueden tener cualquier estructura de subdirectorios por debajo que el usuario quiera mantener. Un ítem, por ejemplo, se puede almacenar en el directorio *Ítems* de una asignatura directamente, o en algún otro subdirectorio por debajo cualquiera. Así, en el ejemplo de la Figura 5.5 se puede observar cómo la asignatura de *Ampliación de Conmutación* tiene los ítems en el subdirectorio *primer parcial* de *Ítems*, y *Bases de Datos* lo tiene directamente en el directorio *Ítems*.

Figura 5.5. Estructura de directorios de Almacenamiento

5.1.2 Subdirectorio “plantillas”

En este subdirectorio sólo se encuentra un archivo, `match_response.xml`, que es el archivo XML con la única plantilla que se utiliza para comprobar la respuesta de los ítems de la norma QTI. Al hacer referencia a la plantilla, el sistema de entrega de los ítems al candidato sólo tiene que comprobar que el sistema de corrección es esta plantilla, y no tiene que leerla entera si ya sabe cómo funciona. Aún así, debe estar disponible por si algún sistema no la conociera, acceder a ella para saber cómo se corrige el ítem.

Para usar la plantilla, el ítem debe tener definida una variable `RESPONSE` para el valor de la respuesta del candidato, y con un valor correcto definido, y una variable de puntuación `SCORE`, para generar el resultado de la corrección del ítem, de tipo entero. A continuación se lista el archivo `match_response.xml` de la plantilla:

```
<?xml version="1.0" encoding="UTF-8" ?>
<responseProcessing xmlns="http://www.imsglobal.org/xsd/imsqti_v2p0"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.imsglobal.org/xsd/imsqti_v2p0
                    imsqti_v2p0.xsd">
```

```
    <responseCondition>
        <responseIf>
            <isNull>
                <variable identifier="RESPONSE" />
            </isNull>
            <setOutcomeValue identifier="SCORE">
                <baseValue baseType="integer">0</baseValue>
            </setOutcomeValue>
        </responseIf>
        <responseElseIf>
            <match>
                <variable identifier="RESPONSE" />
                <correct identifier="RESPONSE" />
            </match>
            <setOutcomeValue identifier="SCORE">
                <baseValue baseType="integer">1</baseValue>
            </setOutcomeValue>
        </responseElseIf>
        <responseElse>
            <setOutcomeValue identifier="SCORE">
                <baseValue baseType="integer">-1</baseValue>
            </setOutcomeValue>
        </responseElse>
    </responseCondition>
</responseProcessing>
```

Se ha creado la plantilla de corrección ya que se requería que todos los ítems se corrigieran de la forma indicada en ella. Su acción de corrección es bien sencilla: genera un valor para la variable de salida `SCORE` de 0 si el ítem se ha dejado sin responder (variable respuesta del candidato igual a `null`), un valor de 1 si el valor de la variable de respuesta es igual a su valor correcto (el candidato ha acertado el ítem), y un valor de -1 si el valor de la variable de respuesta es distinto a su valor correcto (el candidato ha fallado el ítem).

5.1.3 Subdirectorio “Scripts”

Contiene los archivos de JavaScript [24] con las diversas funciones de utilidades que se utilizan en las JSPs de la Aplicación. Los archivos se describen a continuación pudiéndose encontrar su código en el Anexo III, *Planos de código*. Los comentarios están hechos en el formato “Javadoc” de Java, aunque no se vaya a generar dicha documentación de las funciones JavaScript.

checkbox.js

Esta función JavaScritp sólo se utiliza en la página de selección de los ítems a incluir en un test, facilitando la labor del usuario si quiere seleccionar todos los ítems incluidos en un directorio para añadirlos al test, o para seleccionar todos los ítems incluidos ya en el test para eliminarlos. Permite que funcionen los elementos “checkbox” especiales con nombre de *Seleccionar Todos*.

confirmSubmit.js

Esta función de JavaScript se utiliza en las páginas en las que continuar a la siguiente supone un paso importante, pidiendo confirmación al usuario y si está seguro de continuar, como pueden ser las páginas finales del proceso de creación de un ítem o

de un test, o el de selección de los ítems a incluir en el propio test. De esta forma se evita que el usuario pase a la página siguiente al pulsar algún botón *involuntariamente* del teclado (como puede ser el "intro" en algún campo del formulario) o de la página con el ratón.

imagen.js

Este archivo de JavaScript contiene dos funciones, ya que ambas se utilizan en la misma JSP, `explorador.jsp`. Se utilizan para ayudar a la página a mostrar las imágenes en el "frame" derecho de la página para previsualizarlas y para seleccionar la imagen a incluir en las instrucciones.

popup.js

Contiene la función de JavaScript que se encarga de abrir las nuevas ventanas de tipo "pop up" en la Aplicación Web y de que el "focus" se coloque en ellas.

resetValues.js

Este archivo contiene la función de JavaScript que se encarga de que el botón "Restablecer" funcione en los formularios de las páginas correctamente. Ya que éstas se recargan siempre que los datos introducidos no sean correctos o no estén todos los obligatorios, el botón de tipo "reset" por defecto de un formulario HTML [49] no funciona, puesto que los campos que ya hubiera introducido el usuario antes de una recarga de la misma página se dejan en el formulario mediante el atributo `value` del campo del formulario.

Mediante esta función en JavaScript, al pulsar el botón "Restablecer" se ponen los campos del formulario a su valor inicial. Esto es, todos los campos de tipo "checkbox" y "radio" se desmarcan; los campos de tipo "textarea" se ponen vacíos; los campos de tipo "text" que no sean de identificadores se ponen vacíos, y los que sean identificadores se inicializan a su valor por defecto, en función de un valor por defecto *base* seguido de la posición de ese campo identificador dentro de su grupo.

Subdirectorio "whizzery"

En este subdirectorio se encuentran los archivos de JavaScript necesarios para el funcionamiento del editor *WYSIWYG* (*What You See Is What You Get*, lo que ves es lo que obtienes) incorporado para dar formato a las instrucciones de un ítem o un test. Este editor se llama Whizzywig [25], y la versión incorporada en la Aplicación Web es la 54a.

Éste es un editor realizado con JavaScript, que soporta la mayoría de los navegadores de Internet, y que permite aplicar cierto formato de estilo XHTML [1] a las instrucciones, como puede ser negrita, cursiva, subrayado, etc., al texto, introducir tablas, imágenes, enlaces, etc., limitado, claro está, a las permitidas por la norma QTI. Por ello, se incluye en ese subdirectorio el archivo `xhtml.js`, que hace que el código generado por el editor sea XHTML. Otro archivo incluido es `espanol.js`, que hace que los textos aparezcan en castellano. Y finalmente el tercer archivo de JavaScript incluido en dicho subdirectorio es `whizzywig.js`, que es el propio editor.

También se encuentra en el subdirectorio la licencia de uso del editor, que básicamente viene a decir que su uso en todos los ámbitos es gratuito y que permite ser modificado.

Al editor se le pueden añadir imágenes para los botones, pero se han obviado para mejorar la velocidad de carga de la página.

Se le han añadido algunos botones y funcionalidades al editor, ya que permite añadírselas de forma fácil y sencilla, como pueden ser el carácter nueva línea XHTML `<br />`, la cita, la definición, la salida de programa, el subíndice o el superíndice. La llamada al editor y el añadido de las nuevas funcionalidades se han incorporado en la etiqueta personalizada `whizzywig.tag` (explicado posteriormente) que se llama en las páginas que requieren el editor.

5.1.4 Subdirectorio "WEB-INF"

El directorio `WEB-INF` está definido en la especificación Servlet [7], y debe incluir las clases Java generadas, los archivos de etiquetas, las bibliotecas externas importadas, los archivos descriptores de biblioteca de etiquetas y el archivo `web.xml`, el descriptor de despliegue de la Aplicación Web.

A continuación se describen los subdirectorios y archivos que contiene el directorio `WEB-INF` de la Aplicación Web.

web.xml

Es el archivo descriptor de despliegue de la Aplicación Web. A continuación se muestra el código del archivo que está comentado y, por lo tanto, es autoexplicativo:

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4">

    <!-- E-mail de contacto del administrador. Aparece en las páginas de
           error -->
    <context-param>
          <param-name>admin email</param-name>
          <param-value>admin@mail.com</param-value>
    </context-param>

    <!-- Nombre del directorio en el se encuentran los directorios de las
           asignaturas que guardan las imágenes, los ítems, y las baterías de
           test -->
    <context-param>
          <param-name>Directorio Trabajo</param-name>
          <param-value>Almacenamiento</param-value>
    </context-param>

    <!-- Nombre del directorio que guarda las imágenes dentro de cada
           asignatura -->
    <context-param>
          <param-name>Directorio Imágenes</param-name>
          <param-value>/Images</param-value>
    </context-param>

    <!-- Nombre del directorio que guarda los ítems dentro de cada
           asignatura -->
```

```
<context-param>
      <param-name>Directorio Items</param-name>
      <param-value>/Items</param-value>
</context-param>

<!-- Nombre del directorio que guarda las baterías de test dentro de cada
       asignatura -->
<context-param>
      <param-name>Directorio Tests</param-name>
      <param-value>/Tests</param-value>
</context-param>

<!-- Nombre de la Aplicación Web. Es el nombre de la herramienta que se ha
       utilizado para crear el ítem o el test -->
<context-param>
      <param-name>nombre herramienta</param-name>
      <param-value>Herramienta de Creación de Examen QTI. Universidad de
Sevilla.</param-value>
</context-param>

<!-- Versión de la Aplicación Web. Es la versión de la herramienta que se
       ha utilizado para crear el ítem o el test -->
<context-param>
      <param-name>versión herramienta</param-name>
      <param-value>1.0</param-value>
</context-param>

<!-- URI en la que se encuentra la plantilla de respuesta. Se añade en los
       ítems para indicar la localización de la plantilla de corrección a
       utilizar -->
<context-param>
      <param-name>Uri match_response</param-name>
      <param-value>/plantillas/match_response.xml</param-value>
</context-param>

<!-- Definición del nombre del Filtro de Control de la Aplicación asociado
       a su clase -->
<filter>
      <filter-name>ControlFilter</filter-name>
      <filter-class>control.ControlFilter</filter-class>
</filter>

<!-- Mapeo del nombre del Filtro de Control con el esquema de URL que
       interceptará. Está configurado para interceptar todas las peticiones
a
       la Aplicación Web -->
<filter-mapping>
      <filter-name>ControlFilter</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- Definición de un objeto escuchador de la Aplicación -->
<listener>
      <listener-
class>utilidades.etiquetas.util.logging.AplicacionLogger</listener-class>
</listener>

<!-- Definición del archivo de bienvenida de la aplicación. Para cuando se
       introduce sólo la dirección de contexto, sin indicar nada más. -->
<welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
```

```
    </welcome-file-list>

    <!-- Definición de la página de error a la que serán dirigidas las
           excepciones de tipo java.io.IOException -->
    <error-page>
          <exception-type>java.io.IOException</exception-type>
          <location>/errorio.jsp</location>
    </error-page>

    <!-- Definición de la página de error a la que serán dirigidas las
           excepciones de tipo javax.servlet.ServletException -->
    <error-page>
          <exception-type>javax.servlet.ServletException</exception-type>
          <location>/errorservlet.jsp</location>
    </error-page>

    <!-- Definición de la página de error a la que serán dirigidas las
           excepciones de tipo javax.servlet.jsp.JspException -->
    <error-page>
          <exception-type>javax.servlet.jsp.JspException</exception-type>
          <location>/errorjsp.jsp</location>
    </error-page>

    <!-- Definición de la página de error a la que serán dirigidas las
           excepciones de tipo java.lang.Exception -->
    <error-page>
          <exception-type>java.lang.Exception</exception-type>
          <location>/errorgeneral.jsp</location>
    </error-page>

    <!-- Definición de la página de error a la que serán dirigidos los errores
           con el código HTTP 404 -->
    <error-page>
          <error-code>404</error-code>
          <location>/errorarchivonoencontrado.jsp</location>
    </error-page>

</web-app>
```

El archivo contiene todas las definiciones necesarias para el funcionamiento del filtro y el escuchador de la Aplicación Web, así como los parámetros de configuración de la Aplicación Web y la definición de la página de bienvenida y de las páginas de error.

util.tld

Descriptor de la biblioteca de etiquetas llamada *util*. Mediante la URI que define se incluye en las JSPs para poder usar las etiquetas que agrupa. Esta biblioteca agrupa tres etiquetas en ella: la etiqueta *adminMail*, la etiqueta *reqURI* y la etiqueta *logger*. El código se puede ver en el Anexo III, *Planos de código*.

La etiqueta *adminMail* tiene el cuerpo vacío y no tiene atributos. Su llamada sólo imprime por pantalla la dirección de correo electrónico del administrador de la Aplicación Web definida en el descriptor de despliegue de la Aplicación, `web.xml`.

La etiqueta *reqURI* tiene el cuerpo vacío y no tiene atributos. Su llamada sólo imprime por pantalla la URL de una página que previamente ha producido un error.

La etiqueta *logger* tiene el cuerpo vacío y dos atributos: *nivel*, una cadena representando un nivel de registro compatible, y *mensaje*, una cadena con un mensaje de error para registrar. Esta etiqueta registra un error en el archivo que corresponda con el mensaje y el nivel indicado. En el punto 6 de este capítulo se explica en detalle el sistema de registro de errores de la Aplicación Web.

Subdirectorio "classes"

Contiene los archivos `.class` de las clases Java de la Aplicación Web compiladas con su estructura de paquete.

Subdirectorio "lib"

Contiene los archivos `.jar` de las bibliotecas externas importadas porque se utilizan sus funciones en la Aplicación Web. Las bibliotecas importadas son:

- **`commons-fileupload-1.1.1.jar`**: Biblioteca desarrollada por Jakarta [22] para la subida de archivos al servidor. Ver Capítulo 3, *Tecnología Java Servlet*, punto 11.
- **`commons-io-1.2.jar`**: Biblioteca desarrollada por Jakarta necesaria para el funcionamiento de la biblioteca anterior, y que contiene utilidades para distintas acciones de entrada/salida. Ver Capítulo 3, *Tecnología Java Servlet*, punto 11.
- **`jstl.jar`** y **`standard.jar`**: Bibliotecas necesarias para utilizas la biblioteca estándar de etiquetas de JSP en su versión 1.1.2. Ver Capítulo 3, *Tecnología Java Servlet*. Se pueden descargar de la siguiente localización: http://www.apache.org/dist/jakarta/taglibs/standard/.
- **`stax-1.2.0.jar`** y **`stax-api-1.0.jar`**: Bibliotecas necesarias para utilizar la API StAX. Ver Capítulo 2, *Introducción a XML*, punto 7. Se pueden descargar de la siguiente localización: http://stax.codehaus.org/Download.

Subdirectorio "log"

Aquí se guardan los archivos que produzca el sistema de registro de errores de la Aplicación, con extensión `.log`. Hay un fichero por cada día en el que se haya producido un error. En el punto 6 de este capítulo se explica en detalle el sistema de registro de errores de la Aplicación Web.

Subdirectorio "src"

Directorio de los archivos Java fuente de la Aplicación Web. Aquí se guardan en una estructura de directorios como los paquetes las clases Java realizadas en el proyecto. Más adelante se explican en detalle los paquetes y las clases realizadas.

Subdirectorio "tag"

Contiene los archivos de etiquetas `.tag`. Son etiquetas de tipo especial que se pueden implementar utilizando lenguaje JSP. La Aplicación contiene dos etiquetas de este tipo:

aviso.tag

En esta etiqueta se ha abstraído el código para informar al usuario de que hay algún error en los datos introducidos en la página. Define que tiene un atributo, el mensaje para el usuario, que lo recibe con separadores de línea en formato HTML [49]. El mensaje lo muestra en la JSP en un párrafo con color rojo (color HTML #FF0000). También muestra en una ventana el mismo mensaje de aviso al usuario mediante una función "`alert()`" de JavaScript. Antes de pasarlo a dicha función se sustituyen los caracteres nueva línea HTML `<br />` por el carácter nueva línea '`\n`' para representarlo correctamente.

whizzywig.tag

En esta etiqueta se ha abstraído el código necesario para configurar y cargar el editor de texto enriquecido *WYSIWYG* (*What You See Is What You Get*, lo que ves es lo que obtienes) de la Aplicación. Sólo es el código JavaScript necesario para utilizarlo correctamente con los botones personalizados, indicando cuáles se quieren y en qué orden. Declara un atributo, el nombre del "textarea" con el que se va a asociar el editor.

5.1.5 Subdirectorio "work"

Este directo no se encuentra si la Aplicación Web se encuentra comprimida en un archivo WAR. Aunque el contenedor Tomcat lo crea para utilizarlo para guardar las JSPs compiladas a Servlets y los archivos `.tag` compilados a clases Java, y sus versiones compiladas a archivos `.class`, entre otros archivos temporales de la Aplicación Web que utiliza el contenedor. Otro contenedor puede que cree un directorio con otro nombre, o que lo cree fuera del espacio de la Aplicación Web.

6. Tratamiento de los errores de la Aplicación Web

El sistema de tratamiento de los errores que se producen en la Aplicación Web comienza con la definición de las páginas de error en el descriptor de despliegue, `web.xml`. Consta de una serie de páginas que presentan al usuario pantallas de error más "amigables" que las que pueda presentar por defecto el contenedor. Para ayudar a estas páginas se han creado algunas etiquetas personalizadas que aportan información extra en ellas. Finalmente, también cuenta con un sistema de registro de errores utilizando una etiqueta personalizada y las clases que contiene el paquete `utilidades.etiquetas.util.logging`.

6.1 Definición de las páginas de error en `web.xml`

Como se ve en el archivo `web.xml` se han definido una serie de páginas de error para recoger las siguientes excepciones: `java.io.IOException`, `javax.servlet.ServletException`, `javax.servlet.jsp.JspException`, y `java.lang.Exception`. Las dos primeras son las excepciones que se pueden producir normalmente en un Servlet, y las que se pueden lanzar en los métodos que ejecutan el componente Modelo de la Aplicación Web. La tercera se puede producir cuando algún componente de una JSP produzca algún error, como una etiqueta mal usada. El último

tipo de excepción es la superclase de todas las excepciones, y se ha definido para cuando se produzca alguna excepción inesperada de algún otro tipo.

En el archivo `web.xml` también se ha definido una página de error para el común error HTTP [52] 404 de "Página no encontrada".

Las JSPs que recogen los errores se encargan de informar al usuario de forma "amigable" de que se ha producido un error en el sistema, informando del tipo de error, la página Web que lo produjo y proporcionando una dirección de correo electrónico de contacto.

También se ha definido una página de error para el caso en el que finaliza la sesión de usuario, debido a un exceso en el tiempo de espera del servidor, en la que se informa del error producido, indicando también al usuario una dirección de correo para ponerse en contacto con el administrador de la Aplicación.

6.2 Etiquetas de presentación del error

Para ayudar a informar al usuario para que tome las medidas oportunas, se proporcionan en la página la URL que produjo el error y la dirección de correo electrónico del administrador de la Aplicación Web, mediante el uso de sendas etiquetas personalizadas de JSP. Ambas etiquetas están definidas en la biblioteca de etiquetas llamada *util* y ahí se describe su uso. Estas etiquetas imprimen en la JSP la URL del recurso que produjo el error, accediendo a la constante que el contendor coloca en una página definida como página de error, y la dirección de correo electrónico del administrador leyendo el parámetro definido en `web.xml`.

6.3 Sistema de registro de errores

El sistema de registro de error se inicializa con la clase `utilidades.etiquetas.util.logging.AplicacionLogger`, que es un "escuchador" que se ejecuta al cargarse la Aplicación Web. Aquí está definido el nivel de importancia de los errores que se van a registrar, en la variable *`nivelLogger`*; se puede ajustar su valor al nivel a partir del cual se quieran registrar los errores. En esta clase también se define el nombre del objeto "logger" para acceder a él más adelante.

Cuando se cierra la Aplicación Web, esta misma clase se encarga de eliminar todos los manejadores de archivo asociados con este "logger".

La clase que se encarga de registrar los errores es una etiqueta personalizada que se llama en las páginas de error, `utilidades.etiquetas.util.logging.LoggerTag`. Esta etiqueta recoge el mensaje y el nivel del error a registrar. En esta etiqueta se crea un manejador para registrar los errores en un archivo. El nombre del archivo será el día de la semana seguido de la fecha actual, con extensión `.log`, dentro del directorio de registro de errores definido en `web.xml`, por defecto llamado `log`. Se define el manejador para ese archivo, y se le añade el objeto para aplicar el formato personalizado deseado, y el administrador de error personalizado del manejador de archivo, por si ocurriera un nuevo error al intentar registrar un error de la Aplicación Web.

Estas clases registran el error añadiéndolo al final del archivo, con la hora a la que se ha producido el error, la fecha, el mensaje de información del error, la página que ha producido el error y la traza del error, en caso de que sea una excepción.

La Aplicación está configurada para que registre por defecto los errores a partir del nivel de importancia `WARNING` (las constantes están definidas en el objeto `java.util.logging.Level`). Todos los registros de excepciones se realizan con nivel `SEVERE`, y los de fin de sesión y de error HTTP 404 con nivel `INFO`. Por lo tanto, sólo se registrarán en el archivo los errores de excepción, no los otros. Aunque cambiando la configuración, bajando el nivel de registro del "logger", se registrarían en el archivo todos los errores, si hiciera falta depurar algún error relacionado con ellos.

7. Diseño de los paquetes de las clases de la Aplicación Web implementada

A continuación se muestran los paquetes que componen las clases Java de la Aplicación Web, mostrando el diagrama de clases de cada uno de ellos y comentando las clases que contienen.

7.1 Paquete `control`

En el paquete `control` se encuentran las clases encargadas de ejecutar la lógica asociada con cada una de las JSPs de la Aplicación Web que requieren de un componente Modelo para realizar alguna lógica o preparación de datos.

Aquí se encuentra la interfaz `Control` que implementan todos los componentes *Modelo* de las JSPs asociadas y el componente *Control* representado mediante la clase `ControlFilter`, que es un Filtro (implementa la interfaz `Filter`) que intercepta todas las peticiones a la Aplicación Web y toma decisiones sobre qué componente *Modelo* se ejecutará en cada caso.

También están todas las clases que representan los distintos componentes *Modelo* de la Aplicación, y que, por lo tanto, implementan la interfaz `Control`.

En la Figura 5.6 se puede ver el diagrama de clases del paquete.

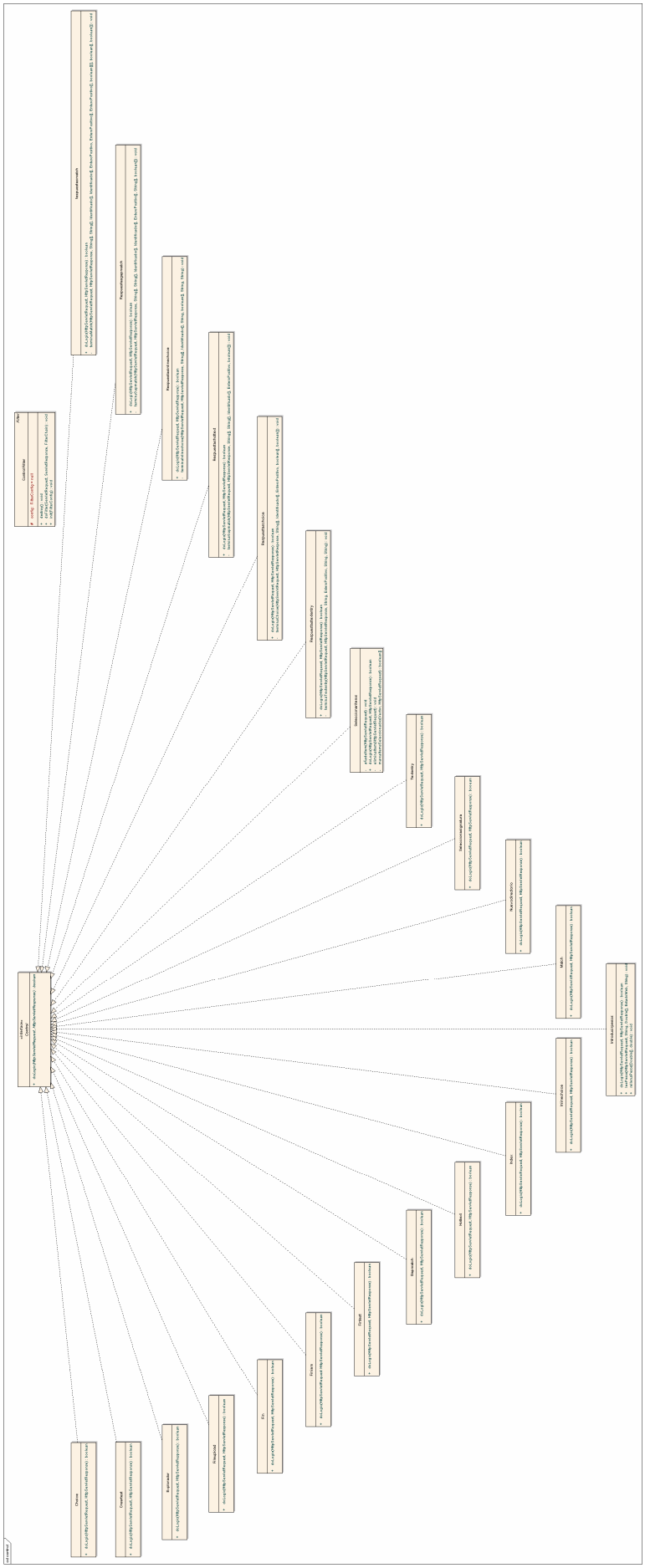


Figura 5.6. Diagrama de clases del paquete `control`

7.2 Paquete `utilidades`

El paquete `utilidades` contiene clases que sirven para ayudar a realizar determinadas acciones repetitivas en la Aplicación Web:

- Clases que representan tipos de datos, con métodos para manejarlos, como `EnteroPositivo`, que representa un número entero positivo a partir de un `String`; `Identificador`, un tipo *identifier* definido según la norma QTI, a partir de un `String`; o `ResumenItem`, elemento con campos que resumen un ítem.
- Clases que representan el estado final tras una comprobación, `MensajeEstado`, o el estado de comprobación final de un formulario de una página Web, `EstadoWeb`.
- Clases que contienen métodos estáticos para realizar acciones repetitivas a lo largo de la Aplicación Web, como la clase `Comprobaciones`, con métodos de ayuda para leer los datos de los formularios y comprobar que sean correctos; `ListaArchivos`, que contiene métodos para listar determinados contenidos de los directorios; `ParserNeutro`, con métodos para escribir y tratar de forma especial cadenas de caracteres que pueden contener código XHTML; o `UrlUtils`, que contiene métodos para codificar una URL.

Su diagrama de clases se puede observar en la Figura 5.7, a continuación:

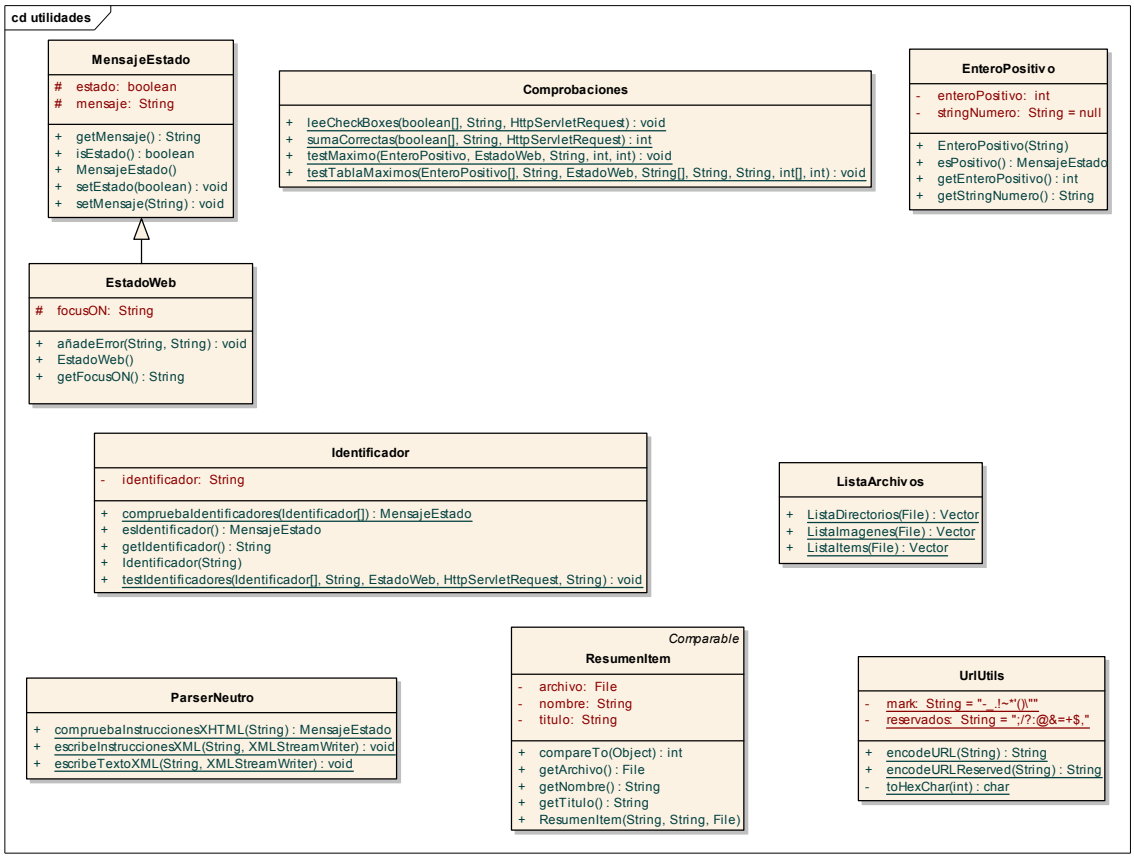


Figura 5.7. Diagrama de clases del paquete `utilidades`

7.3 Paquete `utilidades.etiquetas`

Este paquete no contiene ninguna clase, simplemente contiene los paquetes con el nombre de las bibliotecas de etiquetas de la Aplicación Web. Así, cada paquete de una biblioteca contiene las clases correspondientes a las distintas etiquetas de esa biblioteca, o más paquete relacionados con sus etiquetas.

7.4 Paquete `utilidades.etiquetas.util`

Contiene dos clases de dos etiquetas personalizadas de la biblioteca *util*. Éstas son `RequestedURITag`, que representa a la etiqueta que imprime en la JSP la URI que provocó el error en la Aplicación, y `AdminMailTag`, que lee de la configuración de la Aplicación la dirección de correo electrónico del administrador y la imprime en la JSP.

La tercera etiqueta de la biblioteca está en un paquete dentro de éste, ya que su funcionamiento correcto agrupa varias clases.

Su diagrama de clases se puede observar en la Figura 5.8, a continuación:

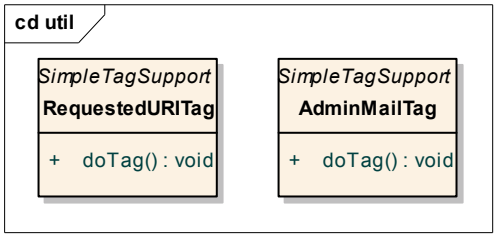


Figura 5.8. Diagrama de clases del paquete `utilidades.etiquetas.util`

7.5 Paquete `utilidades.etiquetas.util.logging`

Este paquete contiene todas las clases necesarias para el correcto funcionamiento de la etiqueta definida con el nombre de *logger*.

Las clases que contiene el paquete son la clase `AplicacionLogger`, un escuchador que arranca el objeto de registro asignándole un nombre y un nivel de importancia a partir del cual se registran los eventos, y de eliminarlo correctamente; `LoggerTag`, que es la clase de la etiqueta propiamente dicha, y se encarga de asignarle un manejador del archivo en el que registrar la información, configurar correctamente el objeto de registro y mandar la información a registrar; `FormatterPersonalizado`, que aplica el formato personalizado deseado a la información a registrar; y `ErrorManagerPersonalizado`, que maneja los errores que se produzcan en el manejador de archivo en caso de error al intentar registrar información en él. Para más información ver el punto 6 de este capítulo.

Su diagrama de clases se puede observar en la Figura 5.9, a continuación:

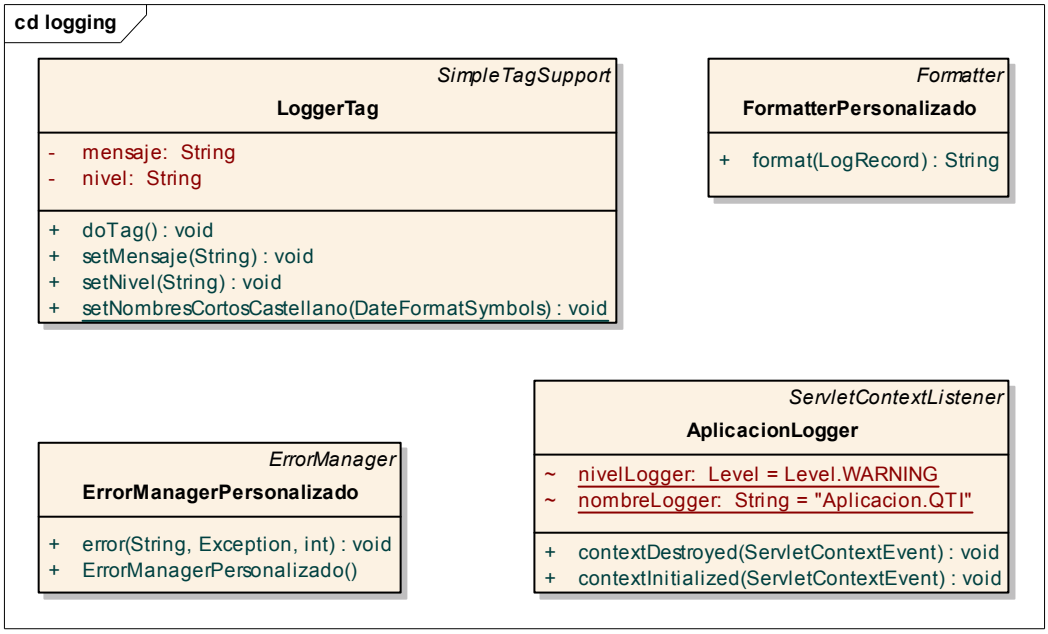


Figura 5.9. Diagrama de clases del paquete `utilidades.etiquetas.util.logging`

7.6 Paquete `xml.items`

Este paquete contiene una clase abstracta, `AssessmentItem`, que representa un ítem según la norma QTI, y es la superclase de todos los tipos de ítems. Contiene los campos comunes de todos los ítems, métodos para acceder a ellos, un constructor que los inicializa, y dos métodos para escribir la cabecera común y el pie común a todos los ítems.

Contiene también una clase por cada tipo de ítem implementado, heredando todas de la anterior, `AssessmentItem`. Estas clases tienen el resto de los campos particulares de cada tipo de ítem, un constructor, y métodos para establecer el resto de los campos que no se establezcan en el constructor. También contienen un método para escribir el ítem XML en el archivo de salida.

Su diagrama de clases se puede observar en la Figura 5.10, a continuación:

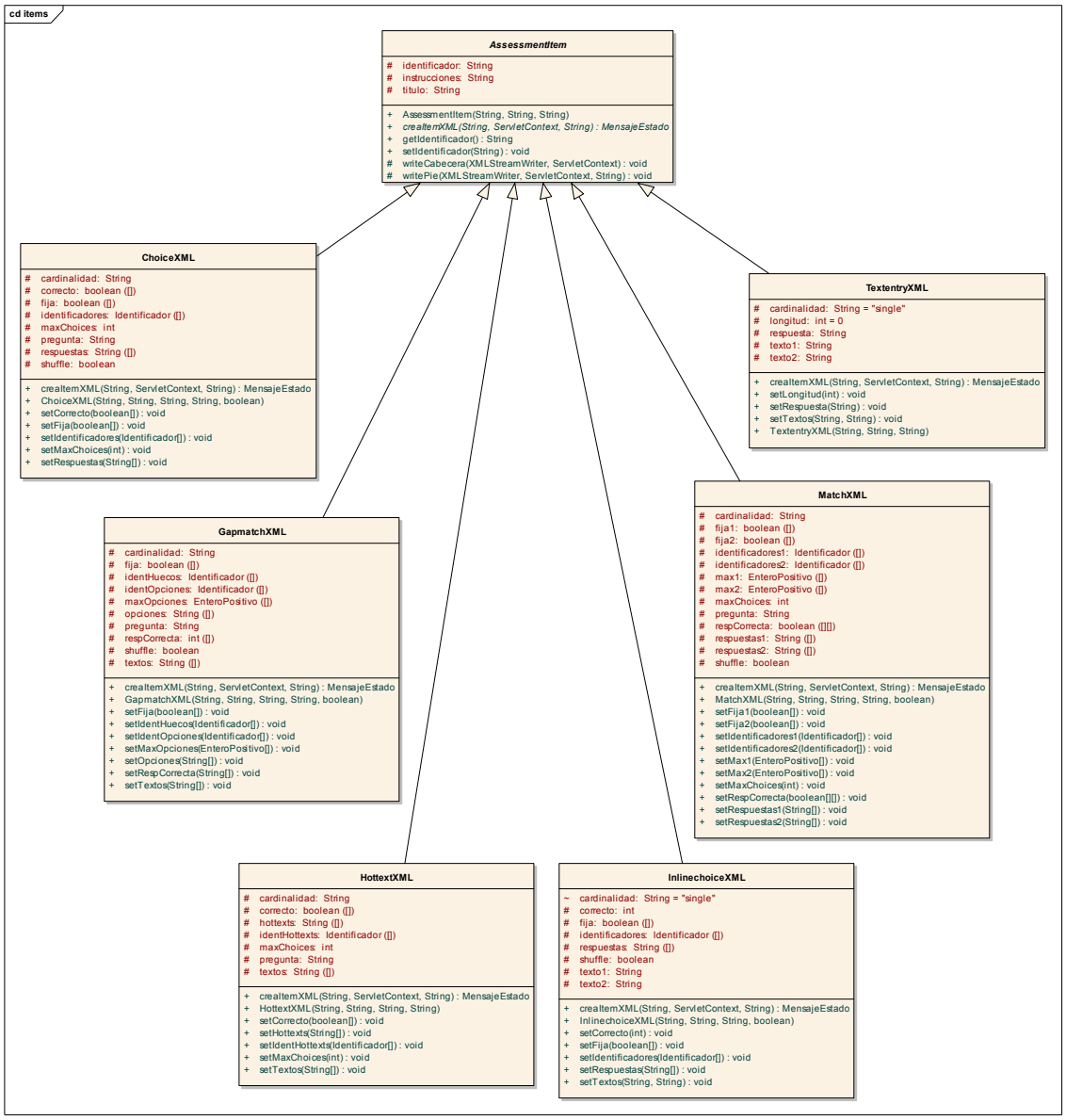


Figura 5.10. Diagrama de clases del paquete `xml.items`

7.7 Paquete `xml.test`

Este paquete contiene todas las clases que definen un test, correspondiéndose con la estructura de clases que define la norma QTI para él.

Todas las clases contienen los campos correspondientes con los atributos implementados de los definidos según la norma QTI para cada clase, métodos para acceder a ellos y constructores para inicializar algunos. También contienen un método para escribir un objeto instancia de una clase en una salida.

A continuación se puede ver la estructura de clases representada en la Figura 5.11, tal y como se describe en la norma QTI para un test, seguida de una explicación de es estructura.

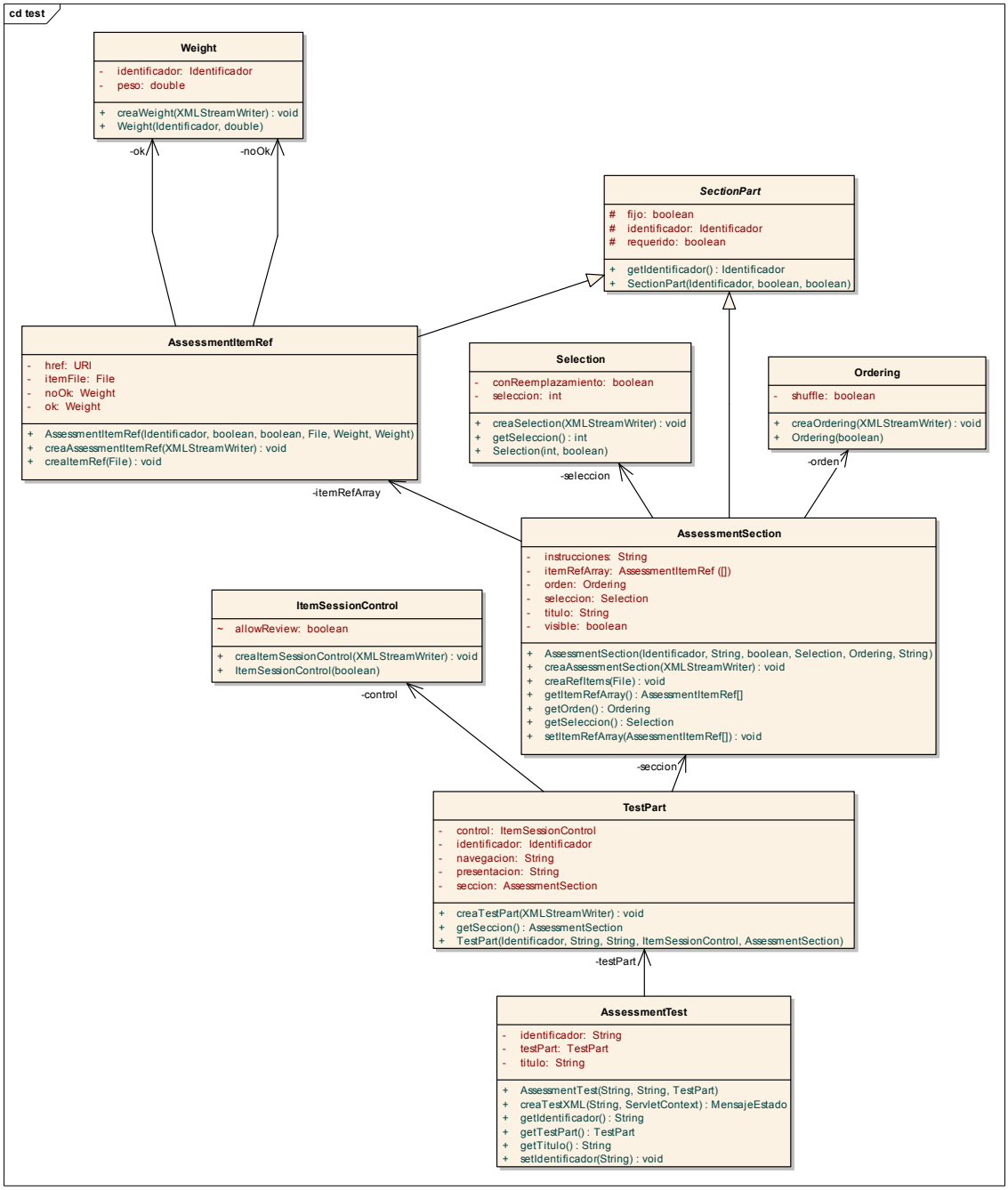


Figura 5.11. Diagrama de clases del paquete `xml.test`

Como se puede observar, la clase básica del test es `AssessmentTest`. Esta clase es el elemento raíz de un test, y contiene sus atributos generales.

Esta clase, según la norma, contiene un cierto número de elementos `TestPart` (en esta implementación sólo contiene un elemento `TestPart`), que representan divisiones del test en partes, o agrupaciones de ítems o secciones.

Cada elemento `TestPart` contiene a su vez un elemento `ItemSessionControl`, que controla por qué estados puede pasar la sesión de los ítems (en esta implementación sólo se controla si se permite la revisión al finalizar o no). También contiene un cierto

número de elementos `AssessmentSection` (en esta implementación la parte del test sólo incluye un elemento `AssessmentSection`), que representan secciones en las que se divide una parte del test.

El elemento `AssessmentSection` hereda de la clase `SectionPart`, y contiene sendas clases para controlar el método de ordenación utilizado de los ítems que incluye y el método de selección utilizado de los ítems a presentar al candidato, representados por las clases `Ordering` y `Selection`, respectivamente.

Ya la clase `AssessmentSection` contiene los elementos referencia a los ítems, representados por la clase `AssessmentItemRef`, que también hereda de `SectionPart`. Y cada una de las referencias contiene dos elementos `Weight`, `ok` y `noOk`, representando los pesos del ítem en caso de respuesta correcta e incorrecta, respectivamente.

8. Entorno de desarrollo

Para la realización del proyecto se ha utilizado la plataforma de desarrollo Eclipse [50], en su versión 3.1.2. Es una plataforma ampliamente extendida y utilizada para desarrollo de aplicaciones, y muy conocida como un IDE para Java (además porque es gratuita). Su utilización es simple, y contiene multitud de opciones para facilitar la labor del desarrollador.

En particular para el desarrollo de este proyecto, ha sido muy útil a la hora de programar el código Java, ya que contiene estupendas herramientas que van desde compilación en tiempo real, corrección del código a medida que se va escribiendo, generación automática de determinados fragmentos de código, marcas para resaltar variables, tipos y clases, hasta los típicos colores para diferenciar a los distintos elementos del código. La versión que se ha utilizado, sin embargo, no posee este tipo de facilidades para el código XML o de las JSPs, así que su realización ha sido algo más *engorrosa* e incómoda que la del resto. A pesar de ello, ha supuesto una gran ayuda a la hora del desarrollo del proyecto, permitiendo desde un único punto desarrollar todas sus partes, realizar pruebas, y mantener la Aplicación Web en general.

Además, Eclipse posee multitud de "plug-ins" o módulos, que permiten ampliar su funcionalidad. En particular, se ha utilizado el módulo de Tomcat para Eclipse [51] de Sysdeo, versión 31, que permite lanzar y detener el servidor desde el mismo Eclipse, y ver en su consola los mensajes aquél.

La instalación de Eclipse es muy fácil. Antes de instalarlo es necesario instalar el entorno de ejecución Java necesario, y el contenedor Tomcat, tal y como se explica en el Anexo I, *Instrucciones de instalación*, en los puntos 1.1, y 1.2, respectivamente. Luego, sólo hay que instalar el propio Eclipse, que se puede descargar de su página Web, e instalarlo siguiendo las instrucciones. Para añadir el módulo de Tomcat, basta con descomprimir el archivo descargado en el subdirectorio "Plugins" de la carpeta donde está instalado Eclipse. La próxima vez que se ejecute Eclipse, automáticamente, se reconocerá y añadirá.