Sistemas Informáticos
Curso 2005-2006

Editor y motor de QTI

Lourdes Costero Ranz
Susana Oliva Pérez
Miguel Ángel Sánchez Fernández

Dirigido por:
Baltasar Fernández Manjón
Iván Martínez Ortiz
Dpto. S.I.P.

Facultad de Informática
Universidad Complutense de Madrid

1. Índice

1. Índice ... 2
2. Índice de figuras ... 4
3. Agradecimientos ... 6
4. Autorización a la UCM ... 7
5. Resumen del proyecto ... 8
6. Abstract of the project ... 8
7. Palabras clave ... 9
8. Manual de usuario ... 10
8.1. Instalación de la aplicación ... 10
8.2. Uso de la aplicación ... 11
8.2.1 Edición ... 13
8.2.2 Exámenes ... 16
9. Módulos de la aplicación ... 18
9.1. Módulo de persistencia ... 18
9.2. Módulo de la interfaz gráfica ... 20
9.3. Módulo de tipos ... 22
9.4. Módulo de tipos Enum ... 24
9.5. Módulo de datos ASI ... 26
9.6. Módulo de datos ItemSession ... 28
9.7. Módulo de excepciones ... 30
9.8. Módulo de datos CorrectResponse ... 31
10. QTI ... 32
10.1. Introducción a QTI ... 32
10.2. Especificación de Casos de Uso ... 33
10.2.1 Caso de uso de Autoría ... 34
10.2.2 Caso de uso de Evaluación ... 34
10.2.3 Caso de uso interactivo basado en el contenido ... 36
10.3. Preguntas, ítems y respuestas ... 36
10.3.1. Taxonomía de Respuestas ... 36
10.3.2. Tipos de ítems ... 38
10.4. Descripción general del Modelo de Datos ... 46
10.4.1. Información básica. ... 46
10.4.2. Evaluaciones, secciones, ítems y bancos de objetos. ... 49
10.4.3. Contenidos ... 60
10.5. La sesión de un ítem ... 63
10.6. El Procesamiento de la respuesta ... 66
11. Tecnologías utilizadas ... 67
11.1. Ant y Maven ... 67
11.1.1.- Introducción ... 67
11.1.2.- Ant ... 67
11.1.3.- Maven ... 69
11.1.4.- Comparación de características de Ant y Maven ... 74
11.2. HSQLDB ... 75
11.2.1. Introducción ... 75
11.2.2. Componentes del paquete jar HSQLBD ... 76
11.2.3. Ejecutando HSQLDB ... 77
11.2.4. Modos Servidor (Server Modes) ... 77

11.2.5. Características generales de la Base de Datos 81
11.2.6. Usar el motor de bases de datos 82
11.3. Hibernate 84
11.3.1. Mapeador Objeto-Relacional 84
11.3.2. Características 85
11.4. Spring 90
11.4.1. Un framework de aplicación 90
11.4.2. Arquitectura de Spring 91
11.4.3. Administración de Transacciones con Spring 99
11.5. Interconexión entre las distintas tecnologías 102
11.6. J2EE 111
11.6.1. Introducción a J2EE 111
11.6.2. Componentes J2EE 112
11.7. Aplicaciones Web con tecnologías Java 112
11.7.1. Java Servlets 112
11.7.2. JavaServer Pages (JSP) 113
11.7.3. JavaServer Pages Standard Tag Library (JSTL) 114
11.7.4. JavaServer Faces (JSF) 114
11.8. Qué es XML. 118
11.8.1. Historia del XML. 119
11.8.2. Sintaxis del XML. 120
11.8.3. Contenidos: DTD o XML Schema 120
11.8.4. Diseño: CSS o XSL. 121
11.8.5. Programación: SAX o DOM. 121
12. Librerías utilizadas 123
13. Gestión de configuración 125
13.1. Modelo de proceso 125
13.2. Personal 125
13.2.1. Participantes 125
13.2.2. Jefe de equipo 125
13.2.3. Equipo de desarrollo 126
13.2.4. Seguimiento y reuniones 126
13.2.5. Comunicación entre el grupo 126
13.2.6. Gestión de archivos 126
13.3. Hardware necesario 127
13.4. Software necesario 127
14. Bibliografía 128

2. Índice de figuras

Figura 1 Instalación a través de *Tomcat Manager* 10
Figura 2 Despliegue de la aplicación........ 11
Figura 3 Página inicial de la aplicación........ 12
Figura 4 Página de Bienvenida........ 12
Figura 5 Edición de un examen – Descripción........ 13
Figura 6 Presentación de un examen – Edición........ 13
Figura 7 Edición de una pregunta de Verdadero/Falso 14
Figura 8 Edición de una pregunta de elección múltiple 15
Figura 9 Página de inicio de un examen........ 16
Figura 10 Ejemplo de una pregunta........ 16
Figura 11 Resultados de un examen........ 17
Figura 12 Módulo de persistencia 18
Figura 13 Módulo de la interfaz gráfica........ 20
Figura 14 Módulo de tipos 22
Figura 15 Módulo de tipos Enumerados 24
Figura 16 Módulo de datos ASI 26
Figura 17 Módulo de datos ItemSession 28
Figura 18 Módulo de excepciones........ 30
Figura 19 Módulo de datos CorrectResponse 31
Figura 20 Representación de un sistema de Evaluación........ 33
Figura 21 Taxonomía de Tipos de Respuestas........ 36
Figura 22 Relación entre tipos de respuestas y representación. 37
Figura 23 Verdadero/falso estándar........ 38
Figura 24 Elección múltiple estándar basada en texto 38
Figura 25 Elección múltiple estándar basada en imágenes 39
Figura 26 Elección múltiple basada en audio........ 39
Figura 27 Respuesta múltiple estándar basada en texto 39
Figura 28 Elección múltiple basada en imagen........ 40
Figura 29 Respuestas múltiples basadas en imagen........ 40
Figura 30 Elección múltiple basada........ 40
Figura 31 Ordenación de objetos estándar basados en texto........ 41
Figura 32 Ordenación de objetos basado en imágenes........ 41
Figura 33 Unión de puntos basado en imágenes 41
Figura 34 Imagen con zonas seleccionables........ 42
Figura 35 Unión de los puntos basado en imágenes........ 42
Figura 36 Rellenar un solo espacio en blanco estándar........ 42
Figura 37 Rellenar múltiples espacios en blanco estándar........ 43
Figura 38 Respuesta corta estándar........ 43
Figura 39 Rellenar huecos con números 43
Figura 40 Entrada numérica con barra de desplazamiento........ 44
Figura 41 Arrastrar y soltar estándar basado en imágenes........ 44
Figura 42 Elección múltiple con rellenar huecos 45
Figura 43 Respuesta múltiple basada en matriz 45
Figura 44 Modelo de objetos de datos de QTI 46
Figura 45 Principio de intercambio de objetos de datos en QTI 47
Figura 46 Estructura genérica del esquema en árbol XML........ 48
Figura 47 Árbol de esquema XML de ítem........ 49

Figura 48 Árbol de esquema XML del elemento Itemmetadata 50
Figura 49 Árbol de esquema XML del elemento Presentation 51
Figura 50 Árbol de esquema XML del elemento Response_lid.................................... 51
Figura 51 Árbol de esquema XML del elemento Resprocessing 52
Figura 52 Árbol de esquema XML del elemento Material.. 53
Figura 53 Árbol de esquema XML del elemento section.. 56
Figura 54 Árbol de esquema XML del elemento assessment 58
Figura 55 Flujos... 60
Figura 56 Diseño de imágenes múltiples... 62
Figura 57 Ciclo de vida de la sesión de un ítem (*itemSession*) 64
Figura 58 Diagrama de estados Feedback 66
Figura 59 Estructura funcional de Maven 72
Figura 60 Arquitectura de Hibernate 85
Figura 61 Arquitectura de Spring 91
Figura 62 Ciclo de vida de una petición Spring. 98
Figura 63 Generación de los beans basado en la configuración de Spring 101
Figura 64 Aplicación J2EE.. 111
Figura 65 Tecnologías Web... 112
Figura 66. Ejemplo de apliación.. 115

3. Agradecimientos

Queremos agradecer la ayuda y comprensión de nuestras familias, que han vivido muy de cerca tanto los buenos momentos como los malos, porque gracias a su apoyo incondicional hemos podido llegar hasta aquí.

Agradecemos a Iván Martínez Ortiz, que no sólo ha dirigido, ayudado y colaborado en la elaboración de este proyecto, sino que nos ha entregado su conocimiento, su tiempo y su paciencia, y por la total disponibilidad que ha tenido durante el transcurso de este año.

También agradecemos al profesor Baltasar Fernández Manjón que ha tutelado este proyecto con eficacia, y sobre todo, agradecemos la confianza que ha depositado en nosotros.

Finalmente, agradecemos a todos nuestros compañeros y demás profesores que nos han acompañado en este largo camino, y muy especialmente a nuestros amigos, que nos han aconsejado y animado.

4. Autorización a la UCM

Los autores de este proyecto, Lourdes Costero Ranz, Susana Oliva Pérez y Miguel Ángel Sánchez Fernández, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos (no comerciales) tanto la memoria como el código y el prototipo desarrollado.

Fdo. Lourdes Costero Ranz

Fdo. Susana Oliva Pérez

Fdo. Miguel Ángel Sánchez Fernández

5. Resumen del proyecto

El objetivo de este proyecto es el desarrollo de un editor y motor de QTI (*Question and test interoperability*). QTI es un estándar que especifica la interoperabilidad de evaluaciones entre sistemas propuesta por IMS (http://www.imsglobal.org) y que describe la estructura básica para la representación de preguntas y sus correspondientes informes de resultados. La idea es poder reflejar todas las condiciones de un examen o autoevaluación en un fichero XML que pueda ser ejecutable por cualquier sistema compatible con QTI.

El proyecto tiene como objetivo principal crear un entorno de aula virtual que, mediante los lenguajes de marcado y las tecnologías Web asociadas, especifica e implementa un sistema de aprendizaje virtual de acuerdo al modelo de referencia IMS QTI. Este aula virtual es flexible, abierta y ampliable de modo que se mejora la gestión, el acceso, la interactividad y la utilidad de la información educativa proporcionada mediante la red.

6. Abstract of the project

The aim of this project is to develop an engine and editor for QTI (*Question and test interoperability*). QTI is a standard that specifies the interoperability of evaluations among systems proposed by IMS (http://www.imsglobal.org), and it describes the basic structure for the representation of questions and its corresponding outcome reports. The idea is to be able to reproduce all the conditions of an exam or assessment in an XML file that might be executed in any compatible system with QTI.

The main purpose of the project is to create a virtual classroom environment that, using mark-up languages and Web technologies, specifies and implements a virtual learning system according to the IMS QTI model reference. This virtual classroom is flexible, open and extendable so that management, access, interactivity and the utility of the educational information provided by the network are improved

7. Palabras clave

- IMS QTI
- E-learning
- Examen
- Web
- HSQLDB
- Hibernate
- Spring
- Groovy
- JSF
- XML

8. Manual de usuario

8.1. Instalación de la aplicación

La aplicación se puede instalar con el fichero WAR (**W**eb **AR**chive) que se proporciona de dos formas posibles con Tomcat:

- Instalación manual:
  1. Copiar el fichero WAR en el directorio <TOMCAT_HOME>/ webapps/
  2. Arrancar Apache Tomcat y el fichero se desplegará automáticamente.

- Instalación a través de *Tomcat Manager:*
  1. Desde la página inicial de Tomcat, ir a *Administration* → *Tomcat Manager* y seleccionar el fichero WAR para realizar el despliegue.

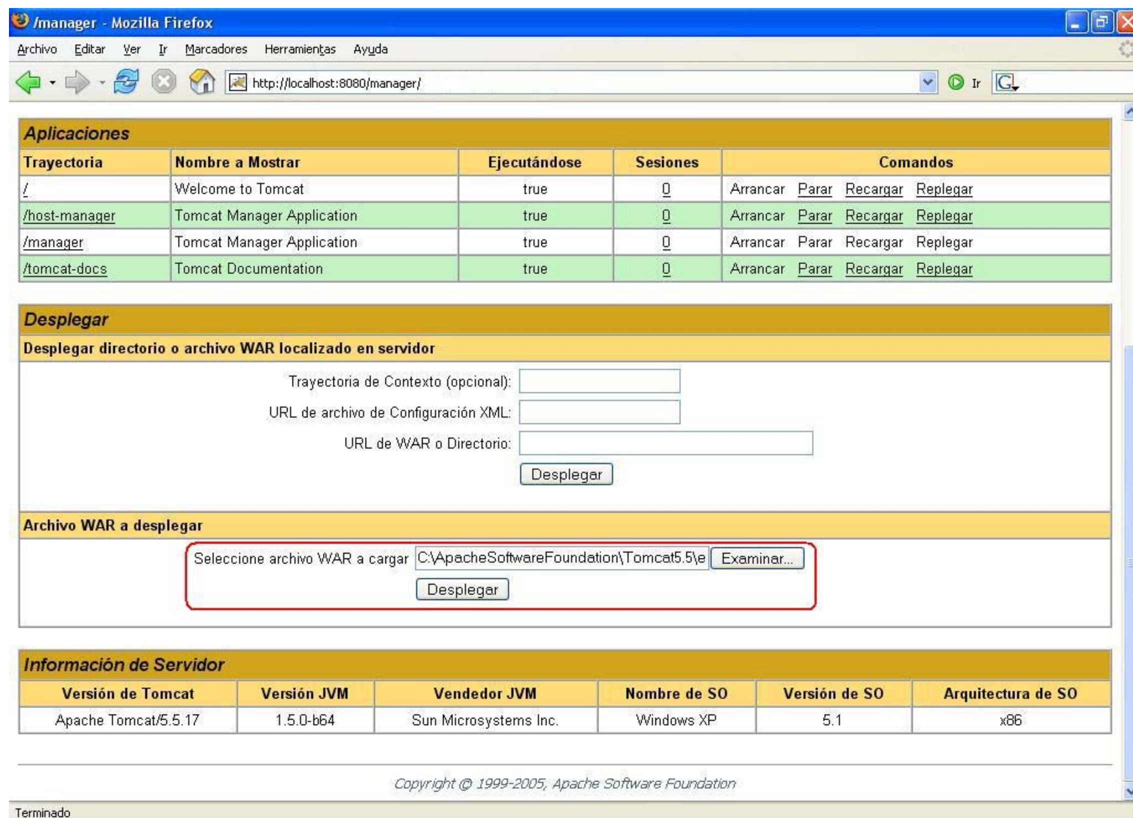


Figura 1 Instalación a través de *Tomcat Manager*

2. Cuando la aplicación se haya desplegado aparecerá una nueva entrada en la tabla *Aplicaciones* llamada *eQTI* que deberemos iniciar pulsando *Arrancar* en la columna *comandos*.

Figura 2 Despliegue de la aplicación

Una vez realizado el despliegue, es necesario modificar un parámetro de configuración del fichero *WEB-INF/web.xml:*

```
<context-param>
        <param-name>
                workingDir
        </param-name>
        <param-value>
                C:/ApacheSoftwareFoundation/Tomcat5.5/webapps/eQTI/
        </param-value>
</context-param>
```

Deberemos modificar el parámetro de contexto `workingDir` añadiendo la ruta completa en la que está instalada nuestra aplicación.

8.2. Uso de la aplicación

Para acceder a la página principal de *eQTI* se debe introducir la siguiente URL en el navegador:

http://localhost:8080/eQTI

Esta es la primera página con la que un usuario se encontrará al entrar en el portal:

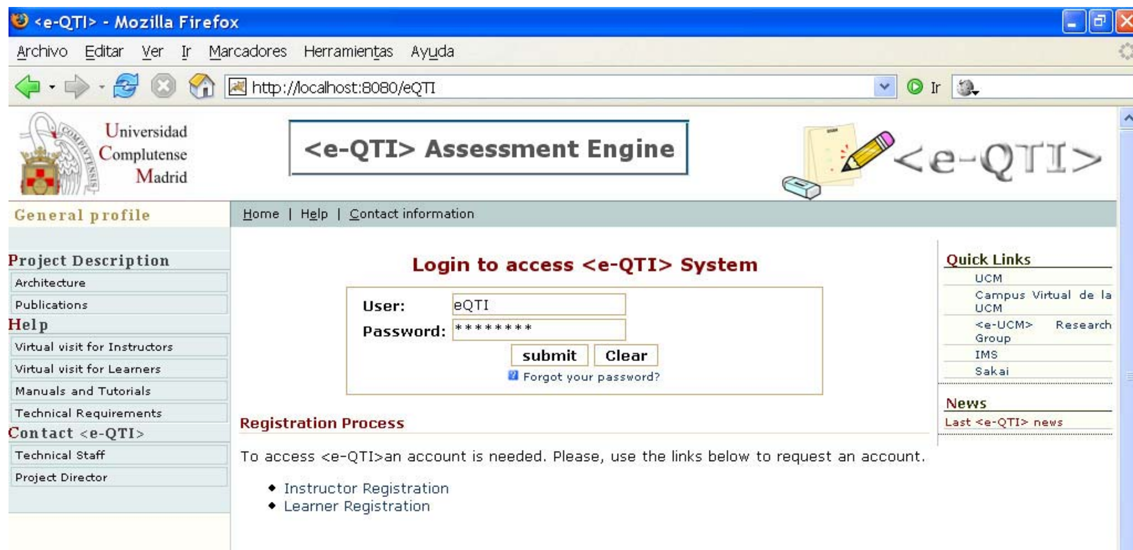


Figura 3 Página inicial de la aplicación

En esta pantalla, el usuario se validará introduciendo su usuario / contraseña para acceder al sistema.

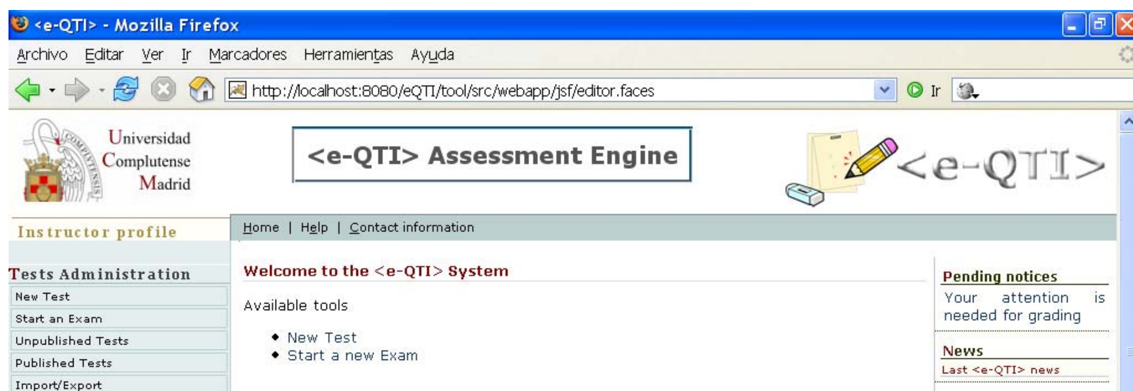


Figura 4 Página de Bienvenida

Aquí se presentan todas las herramientas disponibles que ofrece la aplicación. Por ahora, sólo están disponibles las opciones de editar un examen y ejecutar un examen con las preguntas aleatorias disponibles. Todas las preguntas (ítems) se almacenan en ficheros XML siguiendo el estándar de QTI.

8.2.1 Edición

Para editar un nuevo test, lo primero que hay que hacer es introducir el título así como una breve descripción de la temática del examen. Una vez realizado esto, se pulsa el botón *create* para pasar a la pantalla de edición.

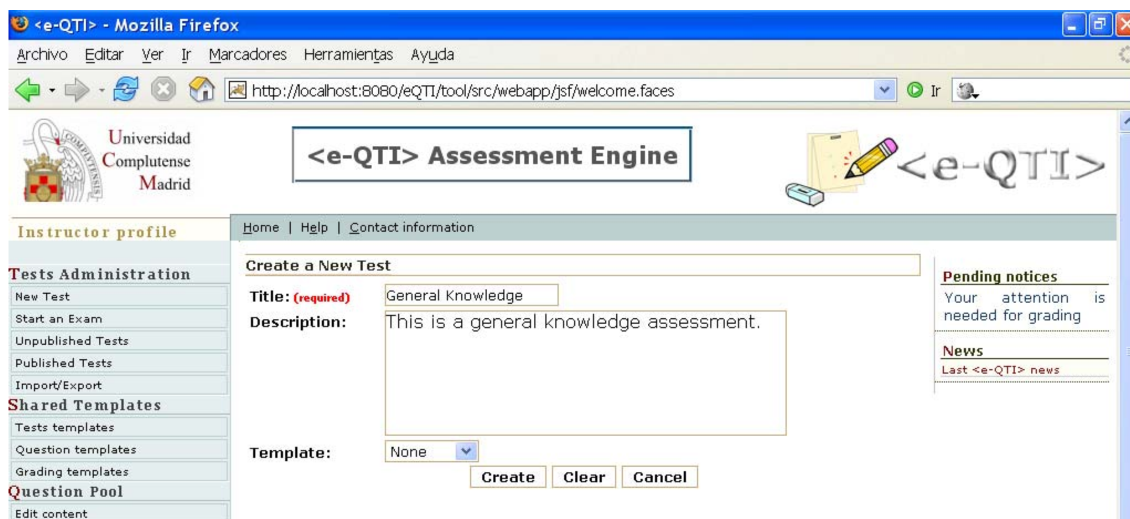


Figura 5 Edición de un examen – Descripción

A continuación se muestra la pantalla de edición que presenta las distintas partes (secciones) que componen un examen:

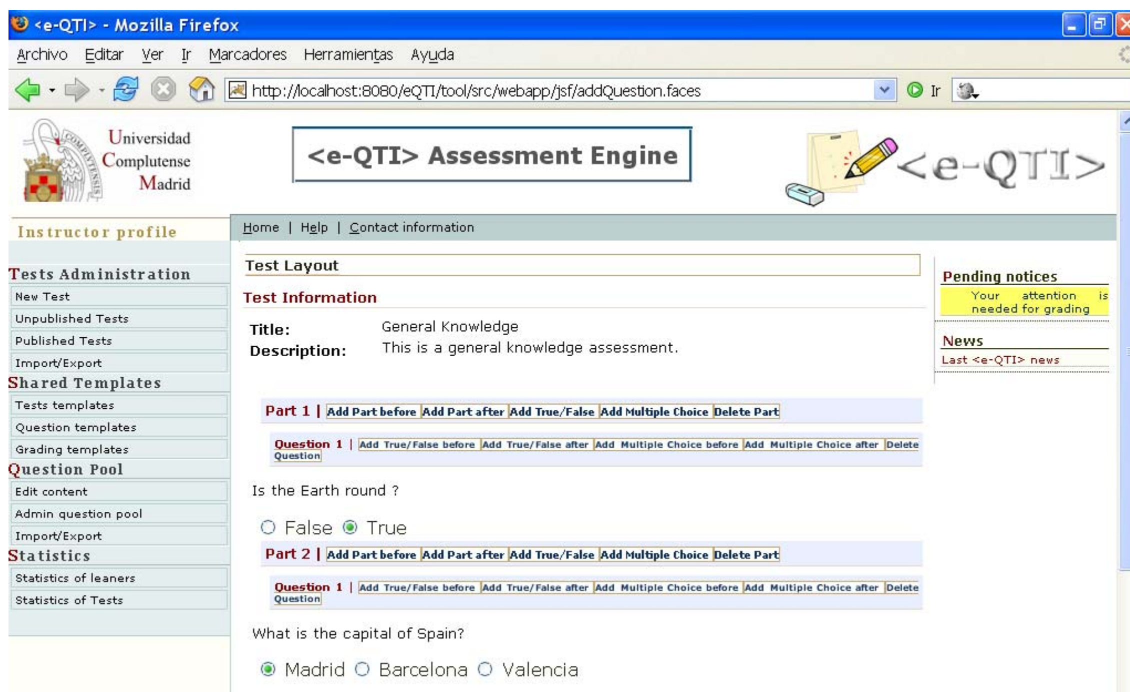


Figura 6 Presentación de un examen – Edición

Cada una de estas partes (secciones) contiene a su vez preguntas. En cada sección tenemos las siguientes operaciones disponibles:

- *Add Part before*: añade una sección antes de la actual.
- *Add Part after*: añade una sección después de la actual.
- *Add True/False*: añade una pregunta de Verdadero/Falso en la sección.
- *Add Multiple Choice*: añade una pregunta de elección múltiple en la sección.
- *Delete Part*: borra la sección del examen.

Dentro de cada pregunta tenemos las siguientes operaciones:

- *Add True/False before*: añade una pregunta de Verdadero/Falso antes de la actual.
- *Add True/False after*: añade una pregunta de Verdadero/Falso después de la actual.
- *Add Multiple Choice before*: añade una pregunta de elección múltiple antes de la actual.
- *Add Multiple Choice after*: añade una pregunta de elección múltiple después de la actual.
- *Delete Question*: borra la pregunta de la sección.

8.2.1.1 Edición de preguntas de Verdadero/Falso

A continuación se muestra la página de edición de preguntas de Verdadero/Falso.

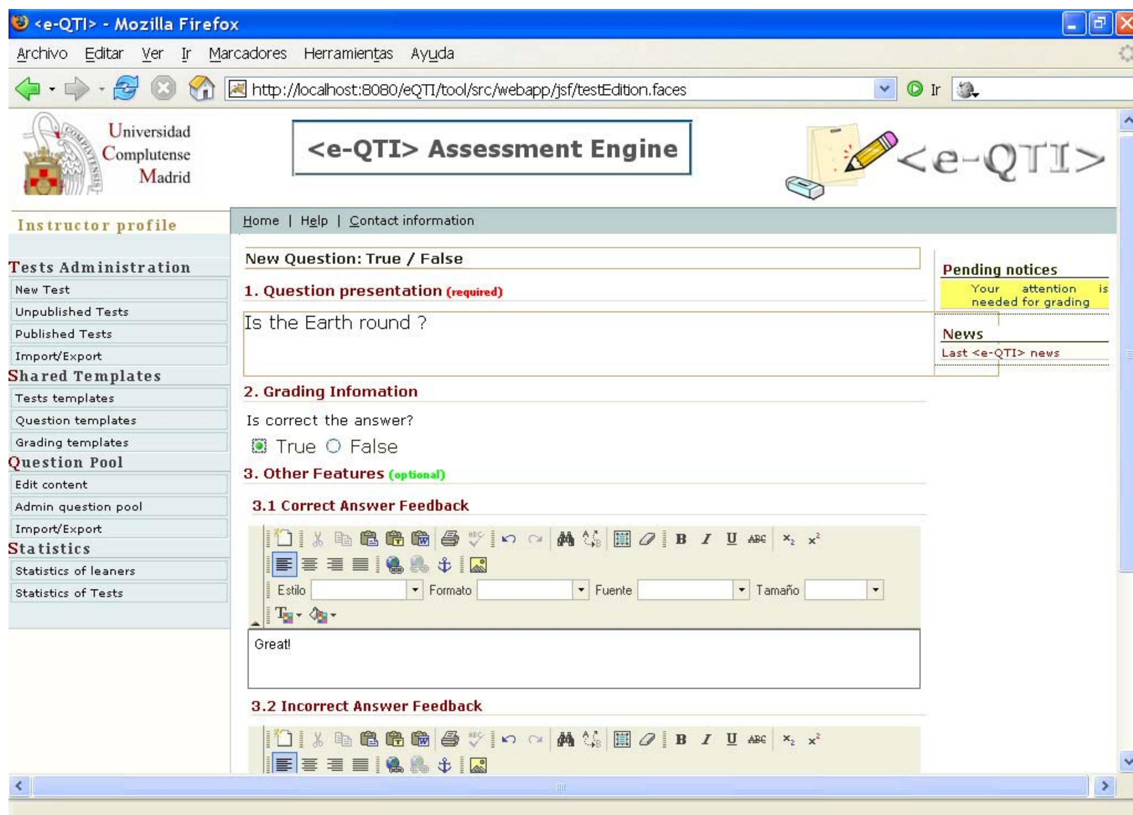


Figura 7 Edición de una pregunta de Verdadero/Falso

En esta página se muestran los campos necesarios para editar una pregunta:

- *Question Presentation:* contiene la cadena de texto que se le presentará al alumno como pregunta.
- *Grading Information:* contiene información para realizar la corrección de la pregunta.
- *Correct Answer Feedback:* contiene información que se le presentará al usuario cuando acierte esta pregunta (opcional).
- *Incorrect Answer Feedback:* contiene información que se le presentará al usuario cuando falle esta pregunta (opcional).

8.2.1.2 Edición de preguntas de elección múltiple

Página de edición de preguntas de elección múltiple.

Figura 8 Edición de una pregunta de elección múltiple

En esta página se muestran los campos necesarios para editar una pregunta:

- *Question Presentation:* contiene la cadena de texto que se le presentará al alumno como pregunta.
- *Answer definitions:* contiene una lista de posibles respuestas para la pregunta. Se pueden añadir tantas opciones como se quiera pulsando el botón *Add more possible responses*.
- *Grading information:* contiene información para determinar cual es la respuesta correcta para la pregunta.

8.2.2 Exámenes

Se pueden realizar exámenes con preguntas aleatorias disponibles para la aplicación.

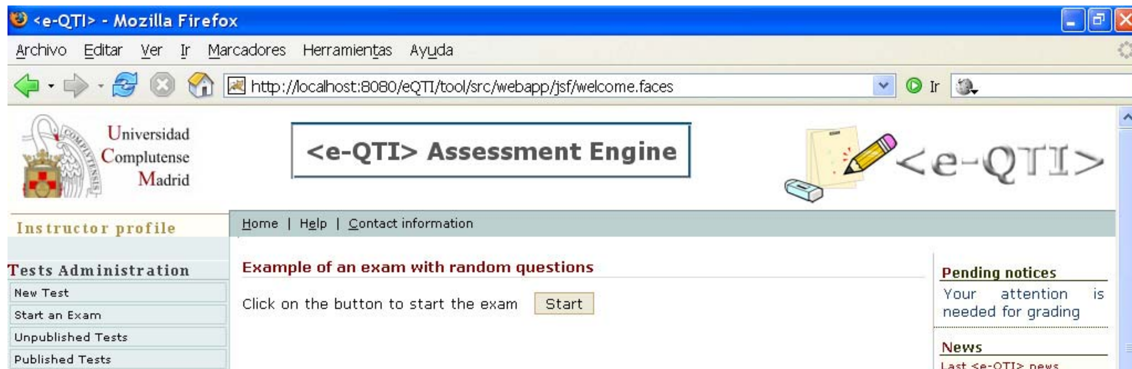


Figura 9 Página de inicio de un examen

Las cuestiones tienen un enunciado y varias respuestas posibles a elegir, aunque la presentación de una pregunta puede variar dependiendo del tipo. A continuación se muestra un ejemplo de una pregunta:

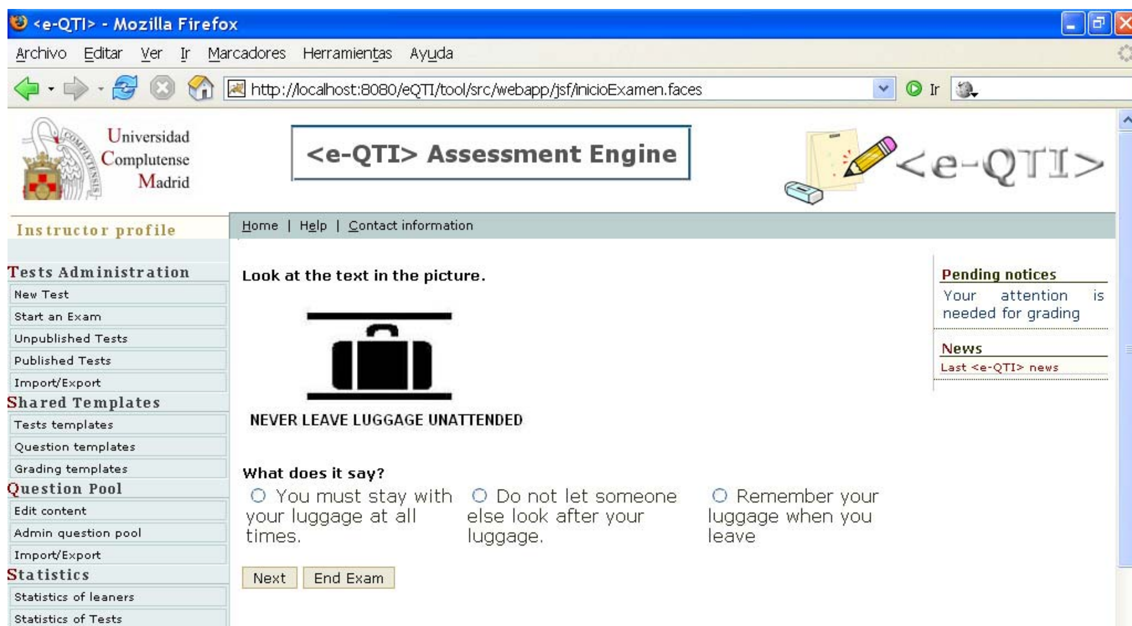


Figura 10 Ejemplo de una pregunta

En cada pregunta hay dos botones: *Next,* que pasa a la siguiente pregunta, y *End Exam,* que termina el examen. Una vez se ha acabado, la aplicación procesa las respuestas del usuario y se muestran los resultados en la siguiente página:

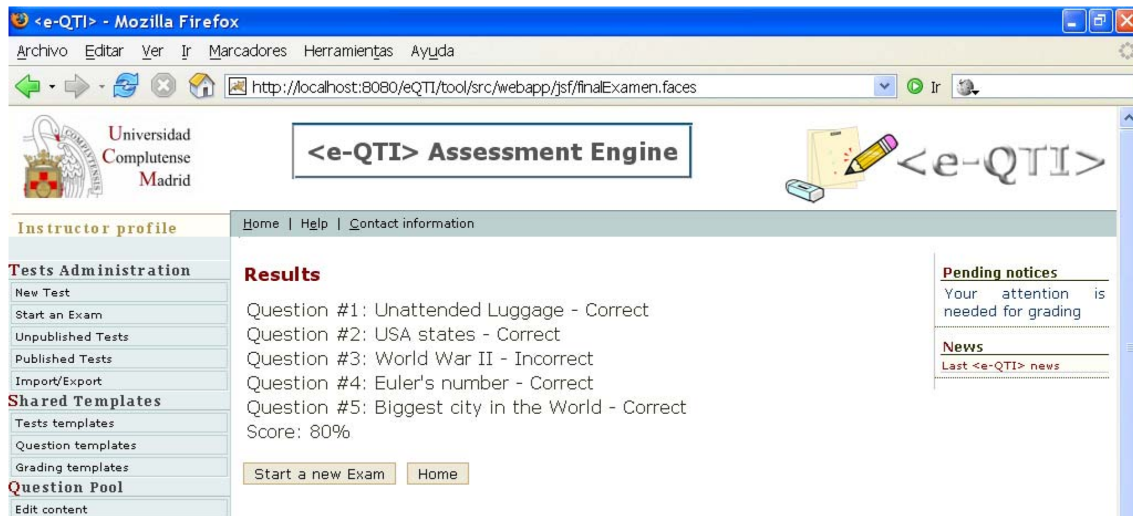


Figura 11 Resultados de un examen

En esta página se muestran las correcciones para las preguntas que el usuario ha contestado así como la puntuación total del examen.

9. Módulos de la aplicación

9.1. Módulo de persistencia

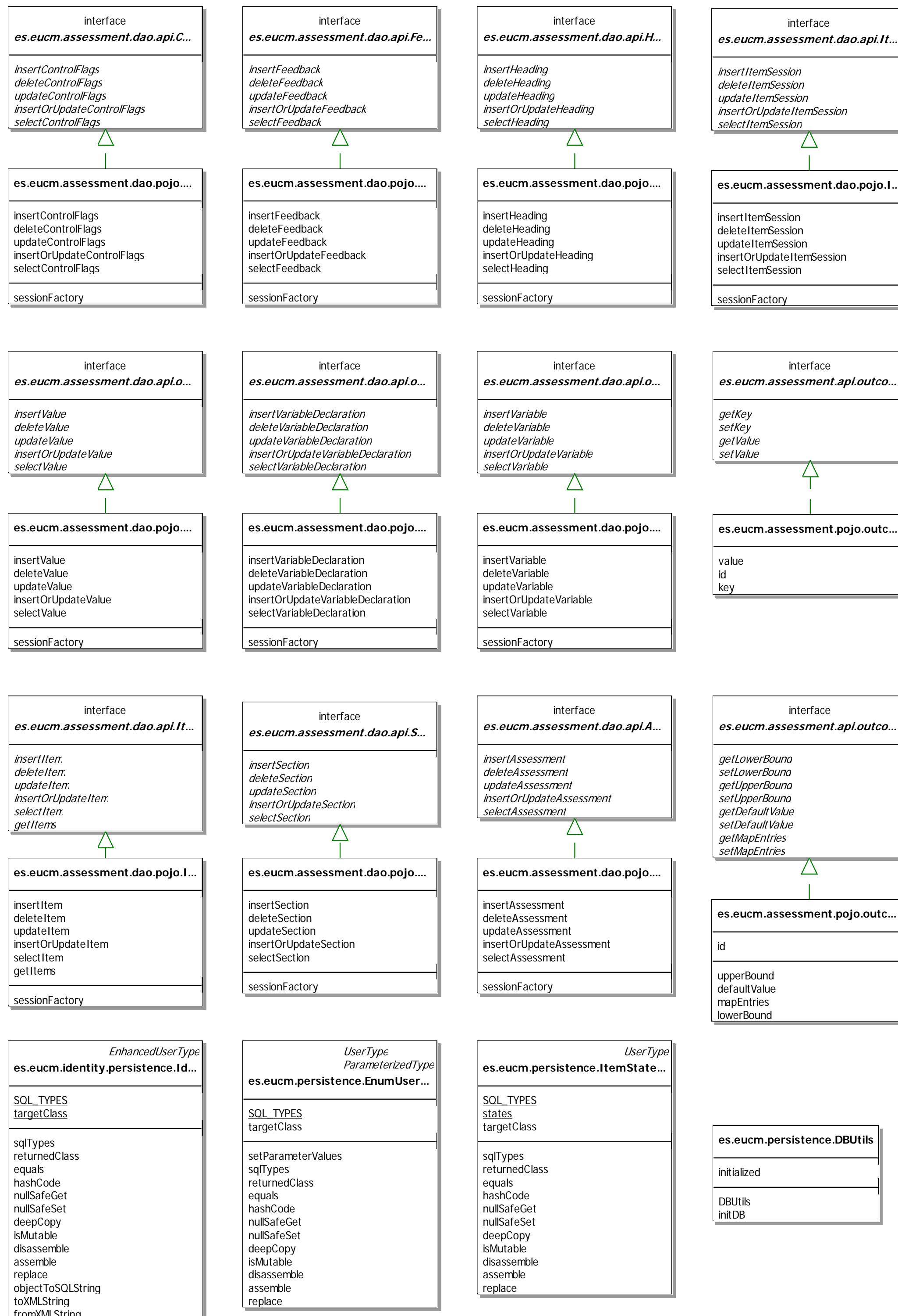


Figura 12 Módulo de persistencia

El diagrama UML de persistencia muestra la estructura de clases que se encargan de realizar las consultas e interacciones con la base de datos del sistema, logrando así el almacenamiento, transferencia y recuperación del estado de la aplicación.

Se aplica la persistencia a *Assessment*, *Section*, *Item*, *ItemSession*, *Variable*, *VariableDeclaration*, *Value*, *Heading*, *Feedback* y *ControlFlags*. Por lo tanto necesitamos unas clases determinadas que realicen las acciones pertinentes para la persistencia de las clases anteriormente mencionadas. También existe la clase *DBUtils* la cual inicializa la base de datos.

Del mismo modo este diagrama muestra las clases necesarias para realizar persistencia a otros tipos de datos. *IdentifierUserType* proporciona persistencia a *Identifier*, *EnumUserType* permite almacenar valores que son instancias de enumeraciones, *ItemStateUserType* ofrece persistencia para la clase *ItemState*. *MapEntry* y *Mapping* utilizadas para el mapeo y persistencia de *ResponseDeclaration*.

9.2. Módulo de la interfaz gráfica

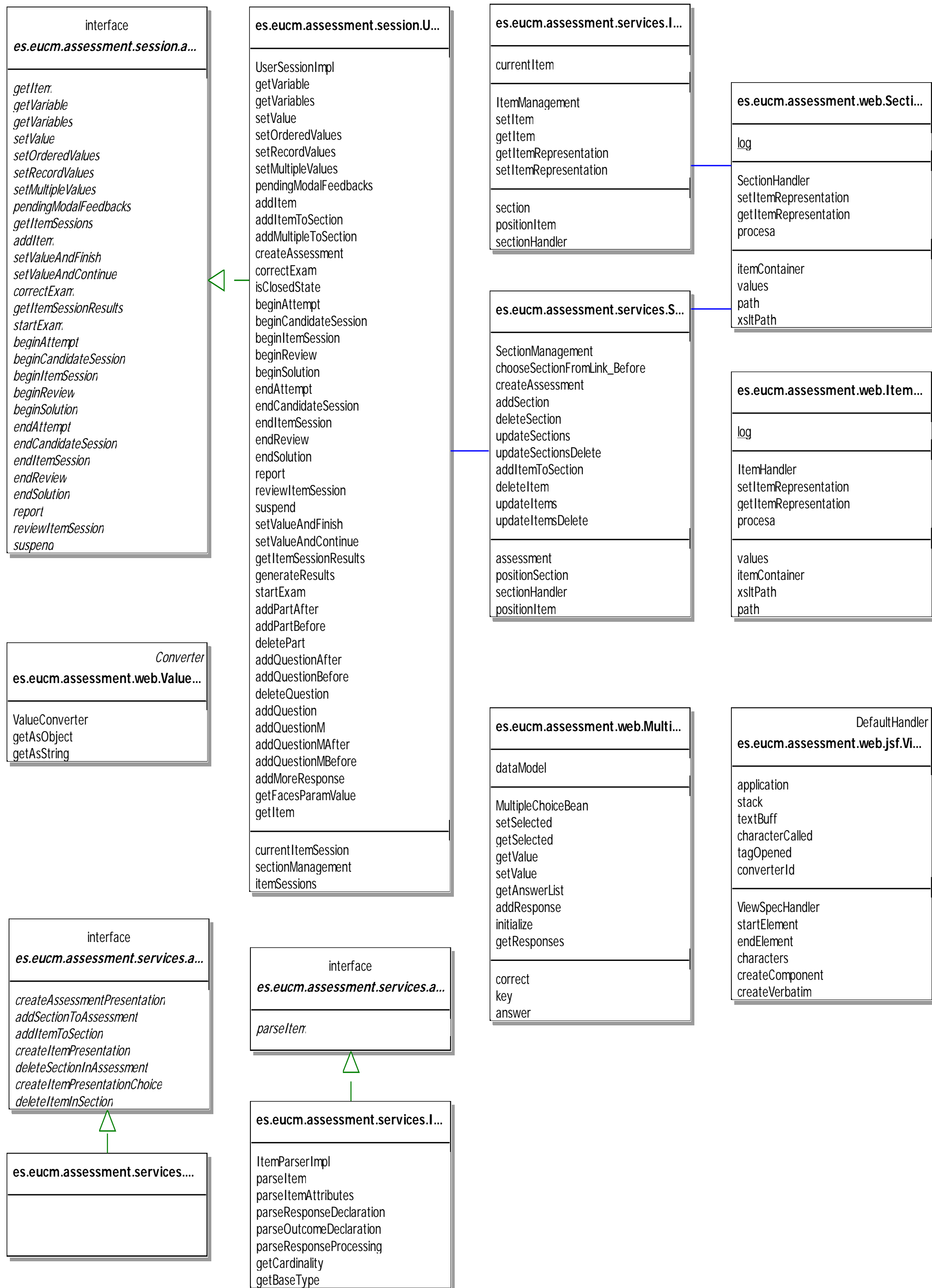


Figura 13 Módulo de la interfaz gráfica

Este módulo muestra las clases necesarias para la presentación de la aplicación mediante JSF. El sistema posee la clase *UserSession* que proporciona una fachada para la conexión entre la lógica de negocios y la presentación de la funcionalidad al cliente. *SectionManagement* e *ItemManagement* realiza las acciones necesarias para generar una nueva evaluación, secciones e ítems.

SectionHandler, *ItemHandler* y *ViewSpecHandler* proporciona las transformaciones XSL desde ficheros xml a una estructura de componentes JSF. *AssessmentParser* crea a partir de los datos e información almacenada en las evaluaciones las presentaciones XML e *ItemParser* es el parser de los archivo xml a la construcción de ítems.

MultipleChoiceBean almacena mediante *DataModel* y *UIData* los valores introducidos en la aplicación por el cliente, proporcionando interacción y soporte a los JSF tags. Por último *ValueConverter* convierte el valor introducido por el usuario *String* a un tipo no predefinido *Value*.

9.3. Módulo de tipos

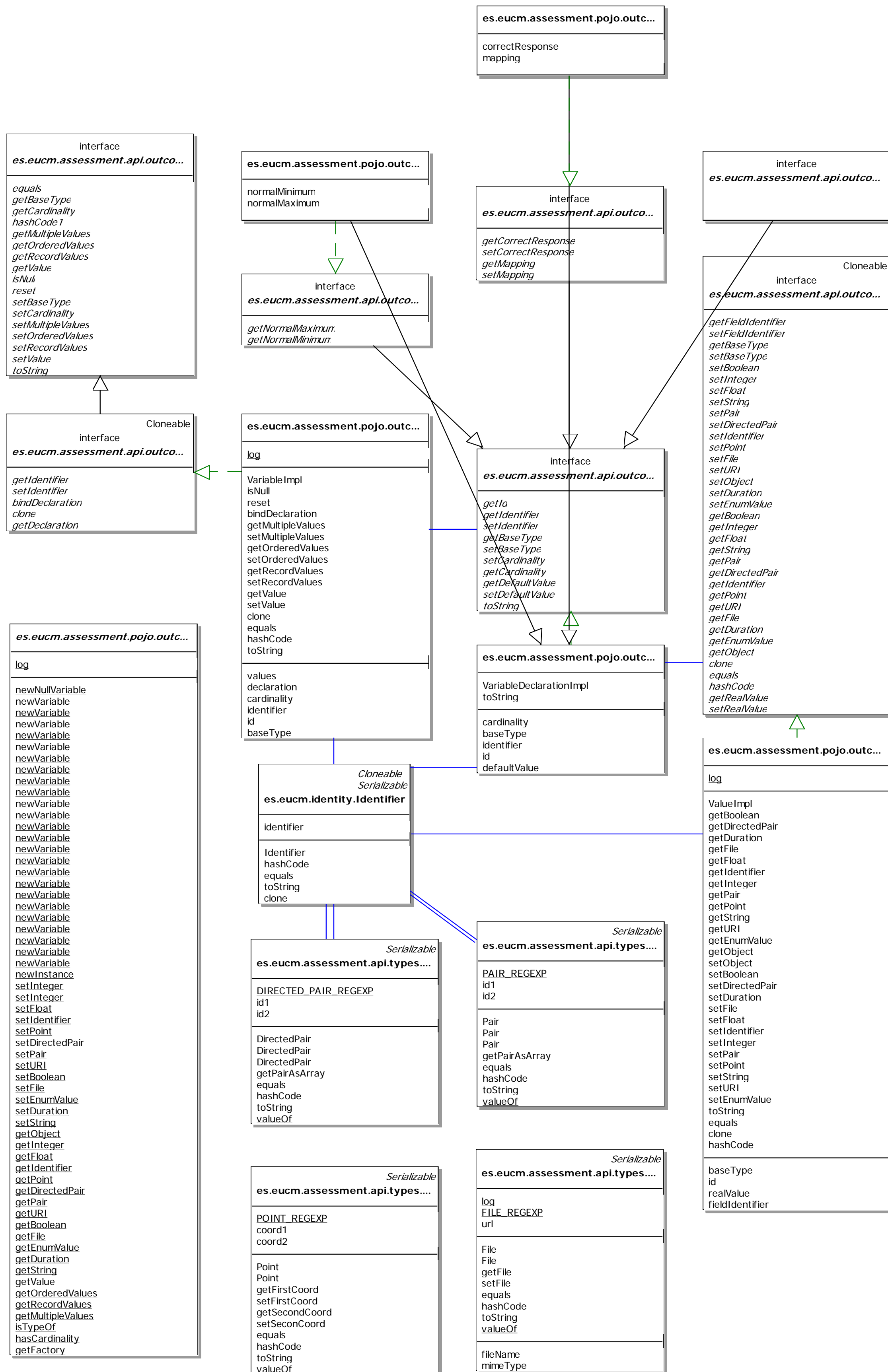


Figura 14 Módulo de tipos

Este módulo expone la estructura de las clases *Variable*, *VariableDeclaration* (*OutcomeDeclaration, ResponseDeclaration, TemplateVariable*) y *Value*. Estas clases almacenan los valores que poseerán los ítems de las evaluaciones dentro de la aplicación. Este diagrama UML también expone las clases de los tipos definidos necesarios para la aplicación: *Identifier*, *Directed_Pair*, *Pair*, *File*, *Point*.

9.4. Módulo de tipos Enum

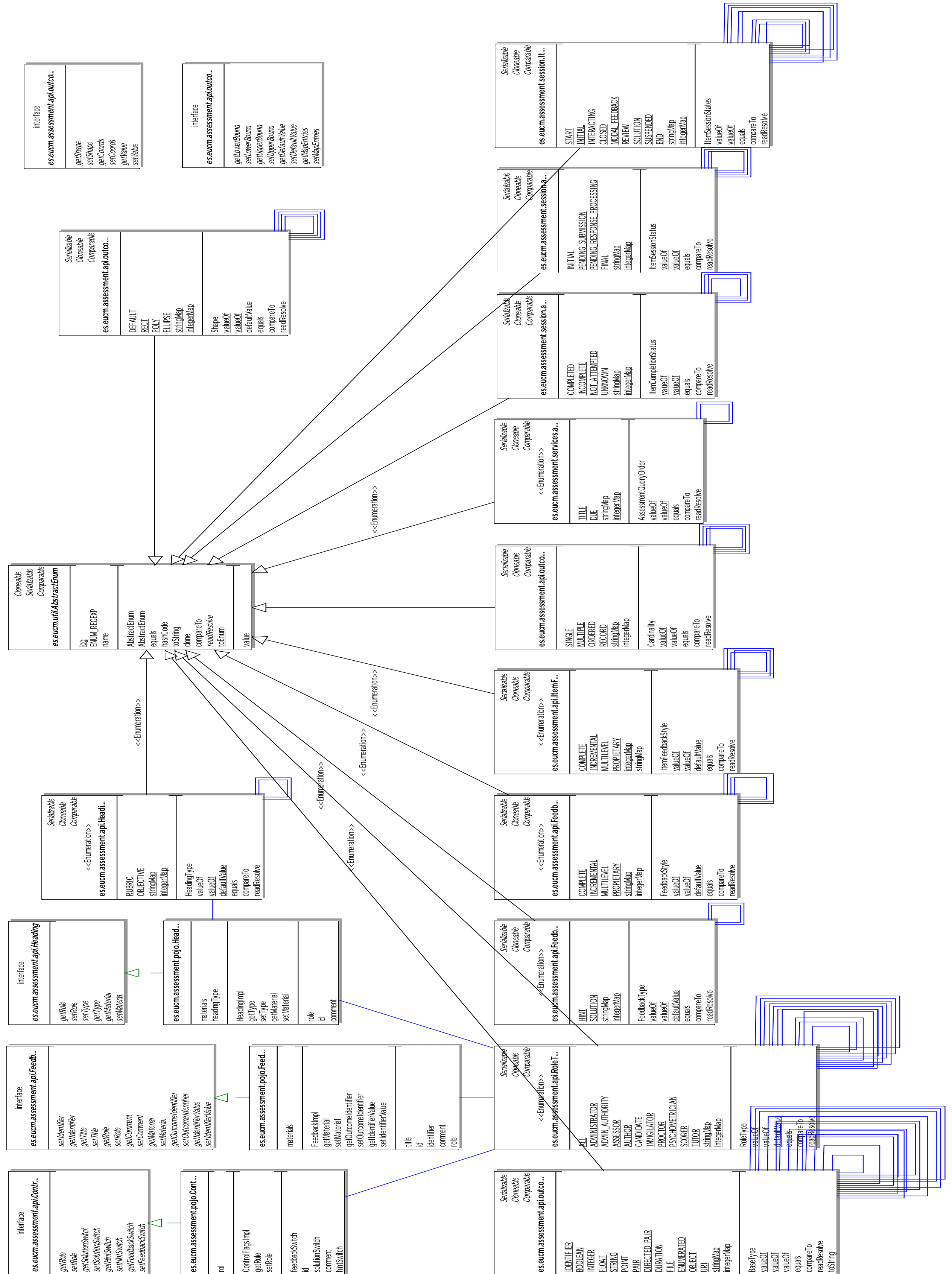


Figura 15 Módulo de tipos Enumerados

Enum indica aquellas clases que son de tipo enumerado: *RoleType*, *FeedbackType*, *HeadingType*, *BaseType*, *Cardinality*, *ItemSessionStates*, *ItemSessionStatus*, *FeedBackStyle*, *Shape*, *AssessmentQueryOrder*, *ItemCompletionStatus*, *ItemSessionStatus*. Todas ellas extienden de la clase *AbstractEnum*

9.5. Módulo de datos ASI

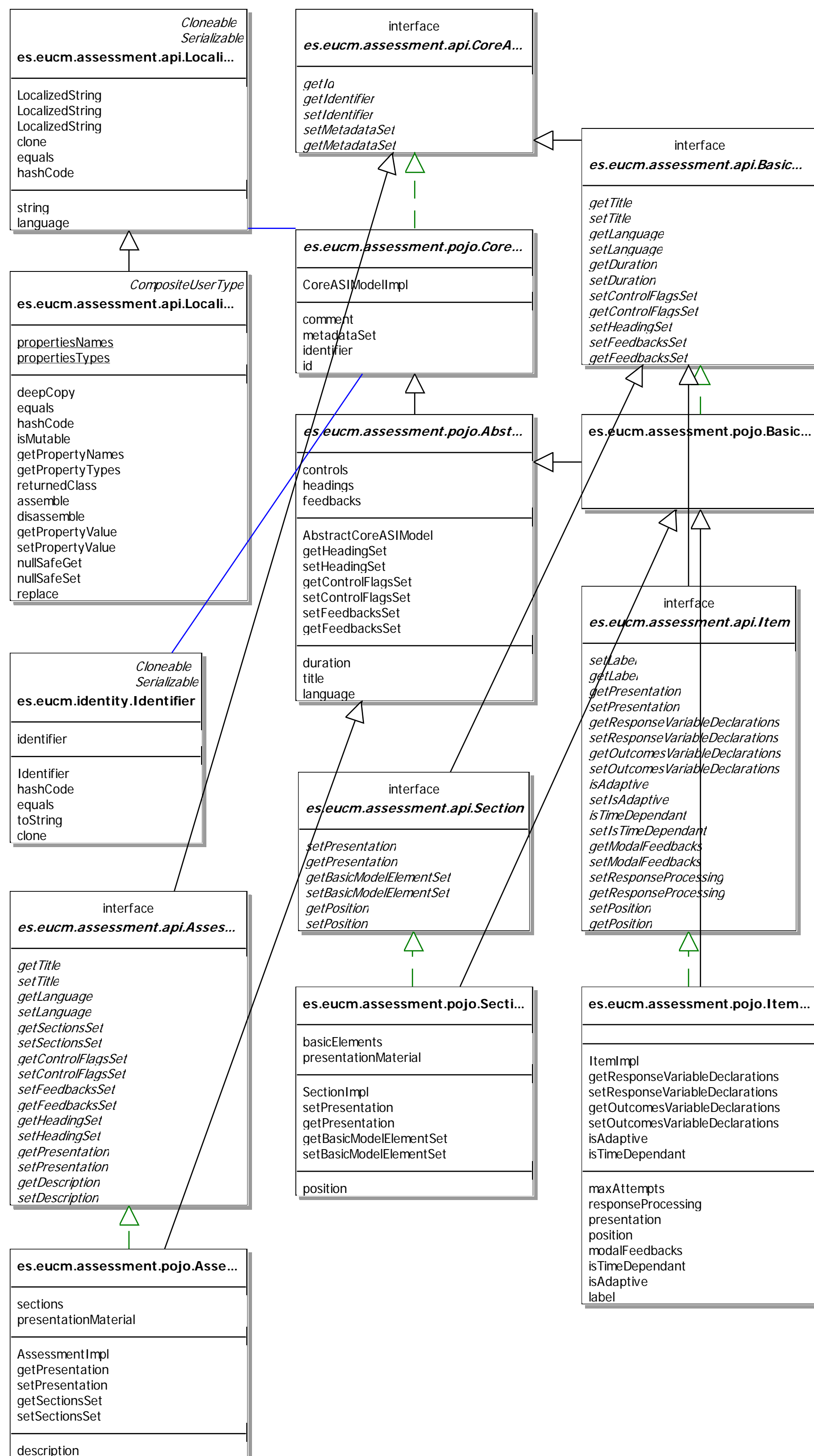


Figura 16 Módulo de datos ASI

Datos ASI muestra la jerarquía de clases para el núcleo de ASI Model, es decir la estructura de los datos ASI. Estos pueden consistir en un grupo complejo basado en múltiples *assessments* y/o múltiples secciones recursivas y/o múltiples ítems. Para ello se encuentra en el más alto nivel de la jerarquía la clase *CoreASIModel* y *CoreASIModelElement*, lo que permite una estabilidad estructural y una composición flexible entre los diferentes datos ASI (*Assessment*, *Section* e Ítems).

9.6. Módulo de datos ItemSession

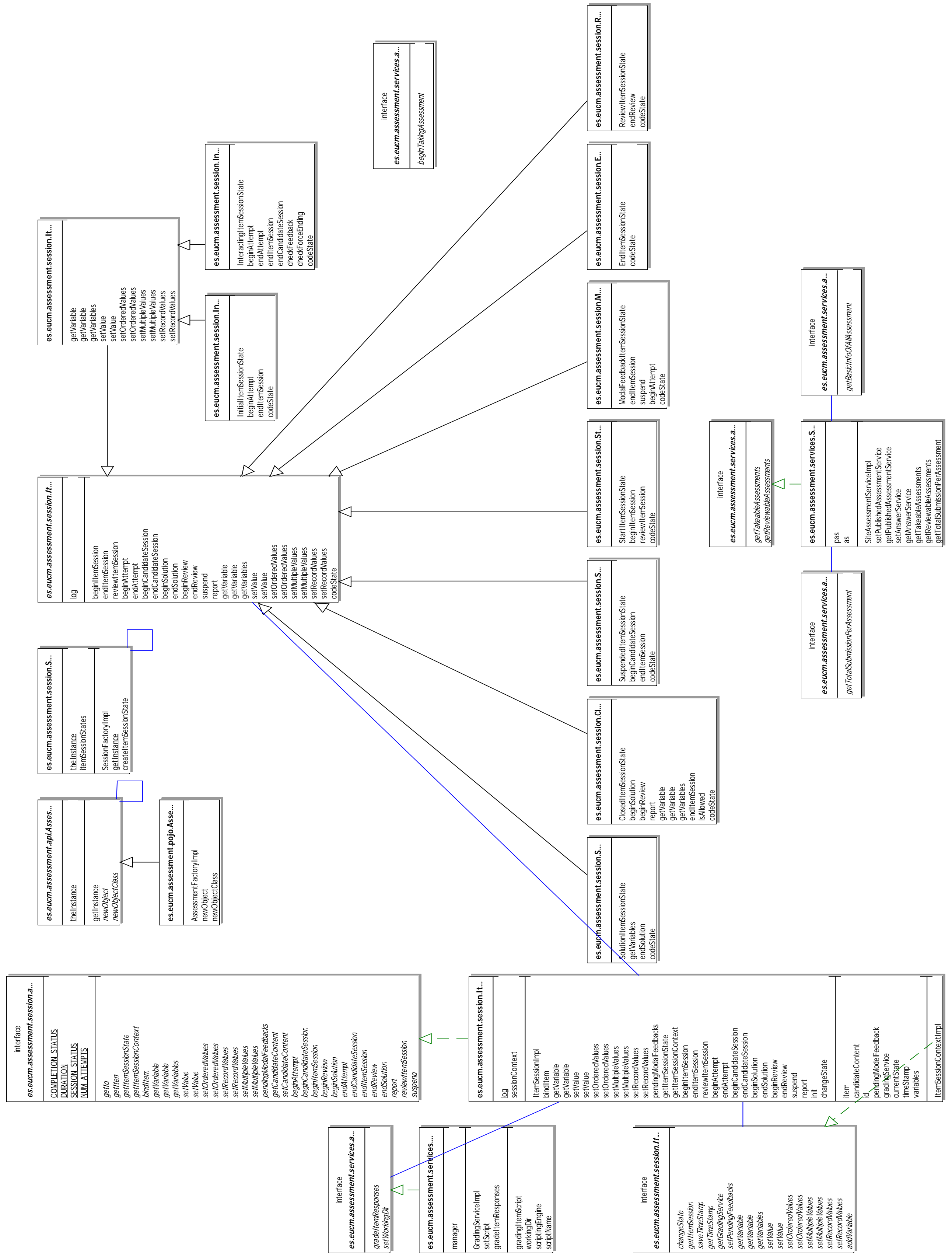


Figura 17 Módulo de datos ItemSession

El diagrama UML ItemSession muestra la jerarquía de clases relacionadas a la sesión de un Item. *ItemSession* contiene *GradingService* para realizar las correcciones del ítem en particular.

ItemSessionContext proporciona información sobre el usuario, cuando la sesión se realiza y ejecuta las operaciones oportunas para las transiciones y almacenamiento de datos de la sesión del ítem. *ItemSessionState almacena* el estado actual de la session, puede ser *Start, Inicial, Interacting, Modal Feedback, Suspended, Closed, Solution, Review, End*. Esta estructura implementa el diagrama de transiciones de *ItemSession* del estándar IMS QTI Information Model.

El diagrama UML *ItemSession* muestra la jerarquía de clases relacionadas a la sesión de un Item. *ItemSession* contiene *GradingService* para realizar las correcciones del ítem en particular.

ItemSessionContext proporciona información sobre el usuario, cuando la sesión se realiza y ejecuta las operaciones oportunas para las transiciones y almacenamiento de datos de la sesión del ítem. *ItemSessionState almacena* el estado actual de la session, puede ser *Start, Inicial, Interacting, Modal Feedback, Suspended, Closed, Solution, Review, End*. Esta estructura implementa el diagrama de transiciones de *ItemSession* del estándar IMS QTI Information Model.

9.7. Módulo de excepciones

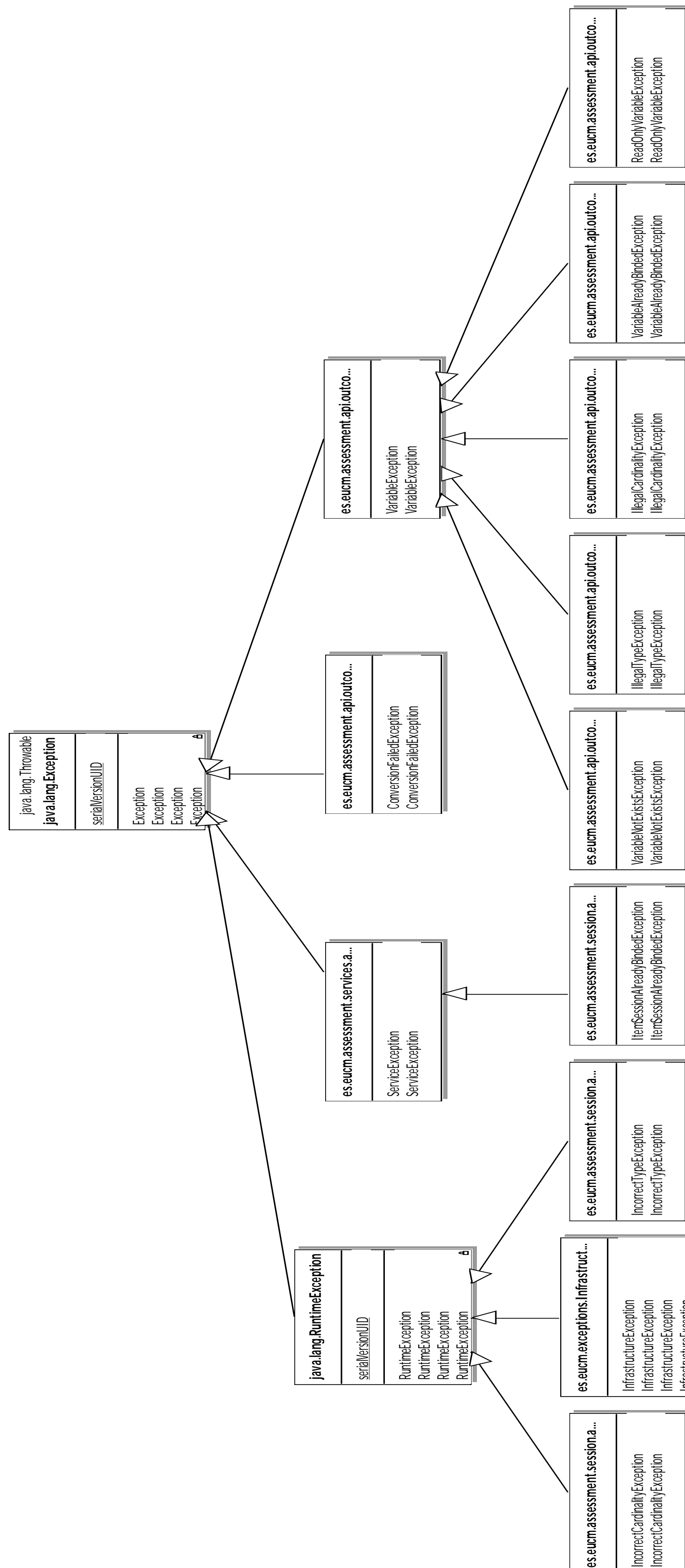


Figura 18 Módulo de excepciones

El diagrama UML *Exceptions* muestra la jerarquía de clases utilizadas para la captura y manejo de excepciones. Esta estructura engloba excepciones tales como *ConversionFailedException, RuntimeExcepcion, ServiceException, VariableException*, que a su vez contiene *IllegalType*, *VariableAlreadyBinding*, *VariableNoExists*, *IllegalCardinality*, *ReadOnlyVariable*.

9.8. Módulo de datos CorrectResponse

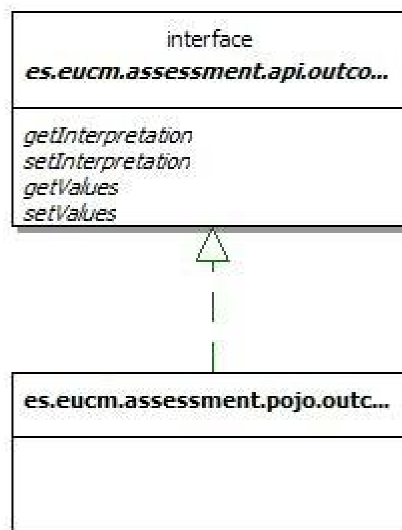


Figura 19 Módulo de datos CorrectResponse

Este diagrama muestra *CorrectResponse* que almacena las soluciones correctas de los ítems. Esta clase será utilizada para la corrección del item en particular.

10. QTI

10.1. Introducción a QTI

La especificación IMS de Interoperabilidad de Preguntas y Tests (*Questions & Test Interoperability, QTI*) describe la estructura básica para la representación de preguntas (ítems) y tests (evaluaciones) y sus correspondientes informes de resultados. Por lo tanto, la especificación permite el intercambio de ítems, evaluaciones y resultados entre distintos sistemas de gestión de aprendizaje (*Learning Management Systems, LMS*), así como librerías de contenido y colecciones.

La especificación QTI está definida en XML para promover la máxima aceptación posible. XML es un lenguaje de marcado estándar muy flexible y potente, lo que permite que la especificación QTI sea extensible y personalizable, proporcionando una adaptación rápida incluso en sistemas especializados o propietarios.
La especificación QTI no limita el diseño de los productos especificando interfaces de usuario, paradigmas pedagógicos o estableciendo una tecnología que restrinja la innovación, la interoperabilidad o la reutilización.

10.2. Especificación de Casos de Uso

Los componentes de procesos (círculos), estructuras de datos (rectángulos) y los participantes (muñecos) de la arquitectura del sistema QTI se muestra en la siguiente figura:

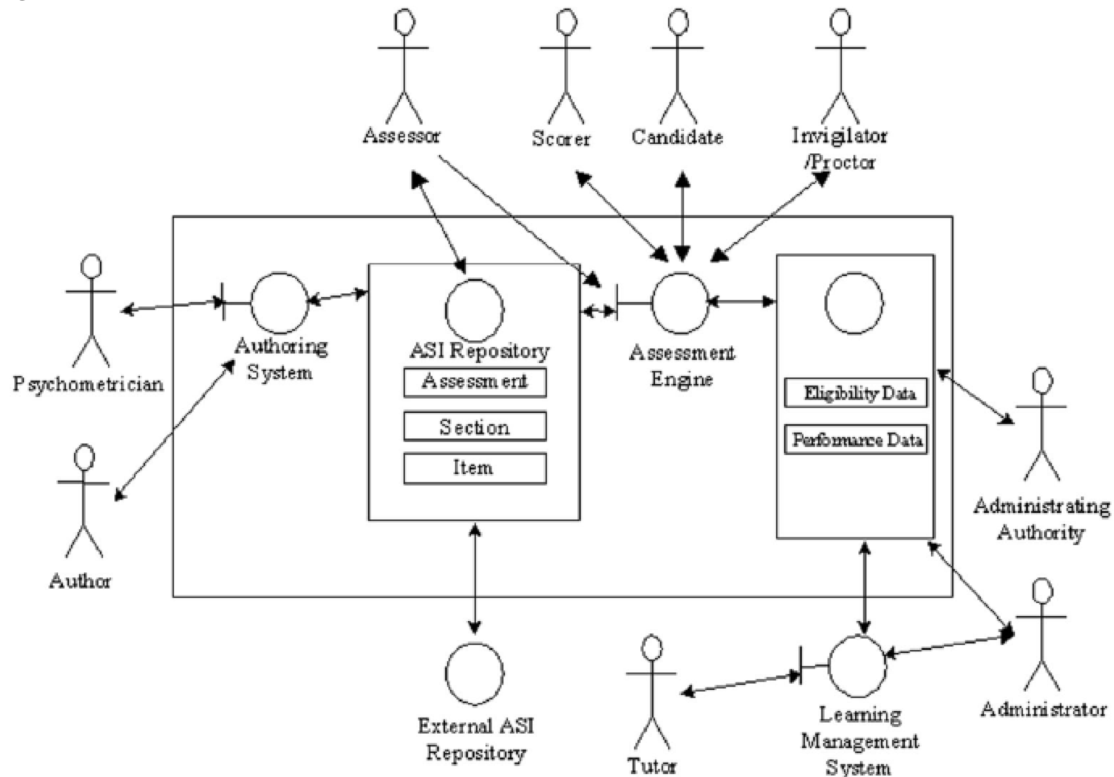


Figura 20 Representación de un sistema de Evaluación

Componentes clave de un sistema de evaluación:

- *Authoring system* (Sistema de Autoría) – proceso que soporta la creación y edición de las evaluaciones, secciones e ítems (ASIs).
- *Assessment engine* (Motor de Evaluación) – proceso que soporta la evaluación de las respuestas en términos de producción de ASIs relacionados con puntuaciones, evaluaciones y feedback.
- *Learning Management System* (Sistema de Gestión de Aprendizaje) – proceso/sistema que es responsable de la gestión de la arquitectura de aprendizaje.
- *Candidate data repository* (Repositorio de datos del candidato) – base de datos con la información específica del candidato.
- *ASI repository* (Repositorio de ASIs) – base de datos de los ASIs locales.
- *External ASI repository* (Repositorio externo de ASIs) – base de datos externa de ASIs que se importarán usando la especificación QTI.

Es posible un amplio rango de casos de uso pero sólo tres se presentan como ejemplo para esta especificación:

- Autoría: creación y edición de ASIs.
- Evaluación de alta participación: examen de candidatos.
- Evaluación de baja participación: apoyo del tutor usando ASIs.
- Evaluación basada en el contenido: contenido interactivo basado en QTI-XML.

10.2.1 Caso de uso de Autoría

La secuencia de procesamiento respecto a la estructura de datos ASI es la siguiente:

- El autor inicia el sistema de autoría.
- El autor crea o modifica ítems, secciones y/o evaluaciones. Éstos se exportan después usando la especificación QTI y se almacenan en una base de datos externa. Las estructuras de datos ASIs pueden consistir en un grupo complejo basado en múltiples evaluaciones y/o múltiples secciones recursivas y/o múltiples ítems.
- El autor puede importar ASIs que se usarán para crear nuevos ASIs. Estos ASIs importados también se ajustan a la especificación QTI.
- Una de las responsabilidades más importantes del autor es determinar el tipo de respuesta y asociar ésta a un tipo de presentación adecuado. Esta asociación dependerá del objetivo educativo del ítem. De la misma manera, el agrupamiento, selección y orden de secciones y/o ítems se basará en los objetivos educativos de la unidad ASI. El autor es responsable también de proveer la información específica de la vista del actor, que será importante porque ayudará a los usuarios a entender como se debe usar el material.

10.2.2 Caso de uso de Evaluación

10.2.2.1 Evaluación de alta participación

El proceso del motor de evaluación es el encargado de realizar esta actividad. Es importante notar que la operación interna del motor de evaluación está más allá del ámbito de esta especificación. Este caso se incluye porque justifica alguno de los componentes estructurales que se deben definir dentro de los ASIs. La secuencia de procesamiento del motor de evaluación es la siguiente:

- El evaluador (*assessor*) construye/selecciona los ASIs que se van a usar a lo largo del procedimiento de evaluación. Estos ASIs serán almacenados en una base de datos interna, por lo que la información de secuencia dinámica debe almacenarse en los propios ASIs.
- La evaluación se activa por el candidato y esta actividad la monitoriza el supervisor de exámenes (*proctor*). Las soluciones de los ASIs del candidato generan un conjunto de respuestas, de nuevo almacenadas internamente. Estas respuestas son un conjunto de identificadores de los ítems junto con información asociada que caracteriza exactamente la respuesta.

- De forma síncrona o asíncrona, cada solución es evaluada por el proceso de respuestas para construir la puntuación inicial (la información de la puntuación es parte de la estructura de datos del ítem). Esta calificación requiere el uso de unas reglas de prueba (*evidence rules*) que se usan para definir los parámetros claves a través de los cuales las respuestas se van a evaluar. Esta evaluación del resultado del ítem se almacena en la estructura de datos de salida. Si un ítem se va a reutilizar en dos evaluaciones diferentes (por ejemplo, selección de alta participación o autoría de baja participación), entonces se puede usar el mismo contenido con diferente procesamiento de las respuestas y acumulación. En este caso, los sistemas de autoría serían los responsables de cambiar la descripción de la salida asociada y el procesamiento de las respuestas, así como los datos de acumulación y los parámetros.
- El proceso de acumulación analiza y recopila las salidas en términos de peso, etc., definido como parte de la estructura de datos de la sección. Esta información se almacena como parte del documento de evaluación.
- La fase final del proceso de evaluación es el proceso acumulado de evaluación en el que el documento de evaluación es procesado completamente usando las instrucciones de nivel de la estructura de datos de la evaluación.
- El paso final de este proceso es el feedback del documento de evaluación a la selección de actividad que puede tener como consecuencia una modificación de los ASIs presentados al candidato.

10.2.2.2 Evaluación de baja participación

El caso de uso del tutor es similar al de evaluación. Las diferencias son que el candidato recibirá feedbacks, incluyendo pistas y una o más posibles soluciones. La secuencia de procesamiento del motor de evaluación del tutor es la siguiente:

- El tutor construye/selecciona los ASIs que se van a usar a lo largo del procedimiento de tutoría. Estos ASIs serán almacenados en una base de datos interna, por lo que la información de secuencia dinámica debe almacenarse en los propios ASIs. Los candidatos podrían actuar con su propio tutor con parte de control sobre la actividad de selección.
- La sesión del tutor es activada por el candidato. El candidato responde a los ASIs y genera un conjunto de respuestas, de nuevo almacenadas internamente. Estas respuestas son un conjunto de identificadores de los ítems junto con información asociada que caracteriza exactamente cada respuesta.
- Cada respuesta es evaluada por el proceso de respuestas para construir la puntuación del ítem. Esta calificación requiere el uso de unas reglas de prueba que se usan para definir los parámetros claves a través de los cuales las respuestas se van a evaluar. Esta evaluación del resultado del ítem se almacena en la estructura de datos de salida. Esta información es usada después para generar feedbacks, por ejemplo, pistas o revelar un parte de la solución.

10.2.3 Caso de uso interactivo basado en el contenido

Es posible usar la especificación QTI para realizar cualquier tipo de material de aprendizaje. El contenido no tiene por que ser usado para un determinado tipo de evaluación. El proceso para desarrollar este contenido es el siguiente:

- Una herramienta de autoría se usa para construir el contenido y el diseño adecuados. Un asistente para la autoría se debería usar siempre que se requiera un formulario basado en preguntas, como por ejemplo una pregunta con respuestas múltiples. El autor debe crear el ítem por completo, incluyendo material de presentación, procesado de reacciones, feedbacks y descripciones de meta datos. Este material luego se exportará a su instancia equivalente QTI.
- Para el intercambio de contenidos interactivos entre LMS y el motor de presentación de contenidos se define el correspondiente modelo de interacción. Los diferentes componentes de QTI se asocian ahora en este modelo de transacción.

10.3. Preguntas, ítems y respuestas

10.3.1. Taxonomía de Respuestas

Las respuestas pueden ser clasificadas por tres tipos diferentes: Básicos, Compuestos y Patentado:

- Básico – es aquel que contiene un único tipo de respuesta.
- Compuesto – actúa como un contenedor de respuestas, es decir sería una combinación de tipos de respuestas básicas.
- Patentado – es un tipo alternativo soportado por la especificación QTI.

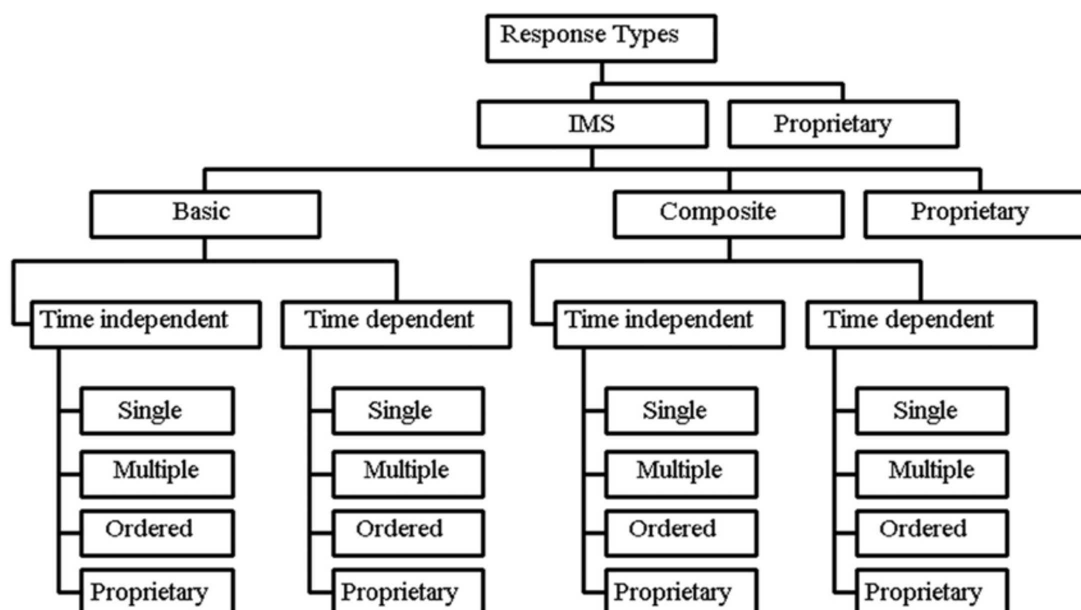


Figura 21 Taxonomía de Tipos de Respuestas

Dentro de cada una de las clasificaciones, el tipo básico o compuesto se puede categorizar en dependiente del tiempo o no (*time dependent*/ *time independent*), dependiendo de si es importante o no el tiempo que se ha tomado el usuario desde que se le presento la pregunta hasta que se ha dado una respuesta.

Si es dependiente debe registrarse dicho tiempo para poder ser usado por los distintos tipos de respuestas y así establecer o definir una secuencia de eventos que el usuario ha de completar en un periodo de tiempo predefinido.

El último nivel de clasificación en la figura 21 esta basada en el número de acciones requeridas por el usuario.

- **Básico**:
    - Único – una única respuesta por parte del usuario.
    - Múltiple – una o mas respuestas del usuario.
    - Ordenado – una o más respuestas del usuario y el orden de las selecciones es significativo.
- **Compuesto**:
    - Único – una única respuesta por parte del usuario por cada ítem que forma el tipo de respuesta compuesto.
    - Múltiple – una o más respuestas por cada ítem que forma el tipo de respuesta compuesto.
    - Ordenado – una o más respuestas por cada ítem que forma el tipo de respuesta compuesto y el orden de las selecciones es significativo.

El siguiente nivel de taxonomía es mostrado en la figura 22

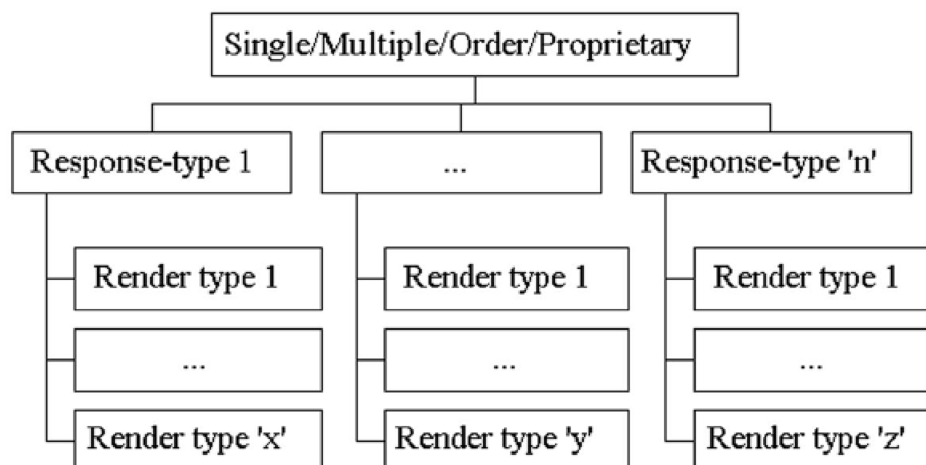


Figura 22 Relación entre tipos de respuestas y representación.

Esta taxonomía muestra la relación existente entre los tipos de respuesta y los diferentes formatos de presentación. Por cada tipo de respuesta hay una o más formas de representar la pregunta, la respuesta y su selección. Por ejemplo puede ser usado simplemente una lista de opciones en formato texto, o una imagen con botones o huecos que pueden ser seleccionados.

Estas dos representaciones requieren la misma acción por parte del usuario, la identificación de la correcta información dentro de varias posibles opciones.

10.3.2. Tipos de ítems

10.3.2.1. Tipos de ítems básicos

Identificador lógico

Verdadero/falso estándar (opciones basadas en texto)

Interpretación basada en la elección.
Típica cuestión verdadero/falso de elección múltiple donde las posibles respuestas son formateadas de diferentes modos. El usuario debe seleccionar una de las opciones (verdadero-falso, si-no).

Figura 23 Verdadero/falso estándar

Elección múltiple estándar (opciones basadas en texto)

Interpretación basada en la elección.
Típica cuestión de elección múltiple basada en texto. Se espera que el usuario elija una de las opciones disponibles pulsando el botón apropiado.

Figura 24 Elección múltiple estándar basada en texto

Elección múltiple estándar (opciones basadas en imagen)

Interpretación basada en la elección.
Cuestión de elección múltiple basada en imagen. El usuario debe seleccionar una de las opciones pulsando en el botón apropiado.

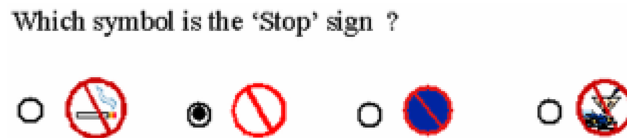


Figura 25 Elección múltiple estándar basada en imágenes

Elección múltiple estándar (opciones basadas en audio)

Interpretación basada en la elección.
Cuestión de elección múltiple basada en audio. Se espera que el usuario haga su elección seleccionando el botón apropiado pero el audio solamente será activado pulsando en cada símbolo de la fuente de sonido. Los iconos usados para denotar los archivos de sonido dependen de la interpretación del sistema.

Identify the VDU.

Figura 26 Elección múltiple basada en audio

Respuesta múltiple estándar (opciones basadas en texto)

Interpretación basada en la elección.
Cuestión de respuesta múltiple típica. El usuario debe seleccionar cada una de las soluciones correctas usando los botones apropiados.

Which of the following elements are used to form water ?

☒ Hydrogen
☐ Helium
☐ Carbon
☒ Oxygen
☐ Nitrogen
☐ Chlorine

Figura 27 Respuesta múltiple estándar basada en texto

Elección múltiple con una sola imagen (opciones basadas en imagen)

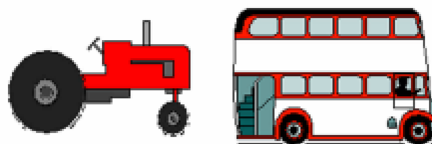
Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de elección múltiple usando puntos seleccionables. Se espera que el usuario seleccione el botón apropiado.

What *city* is the capital of *France* ?

Figura 28 Elección múltiple basada en imagen

Respuesta múltiple con múltiples imágenes (opciones basadas en imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de respuesta múltiple usando varias imágenes con puntos calientes. El usuario debe usar el ratón para pulsar en este caso en las cuatro ruedas.



Identify all of the wheels on the vehicles displayed.

Figura 29 Respuestas múltiples basadas en imagen

Elección múltiple (opciones basadas en barra de desplazamiento)

Interpretación basada en el deslizamiento.
Cuestión de elección múltiple usando puntos seleccionables con barra de desplazamiento. Se espera que el usuario mueva la barra de desplazamiento para señalar el valor correcto.

What is the value of 2 * 3 ?

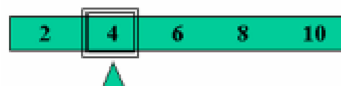


Figura 30 Elección múltiple basada

Ordenar objetos estándar (objetos basados en texto)

Interpretación basada en los objetos.
Cuestión típica de objetos basados en el orden del texto. El usuario debe pulsar en cada objeto de texto y colocarlo en el orden correcto.

Figura 31 Ordenación de objetos estándar basados en texto

Ordenar objetos estándar (objetos basados en imagen)

Interpretación basada en los objetos.
Típico ordenamiento de objetos basados en imagen. Se espera que el usuario mueva cada objeto usando el ratón y moviéndolos alrededor de la pantalla.

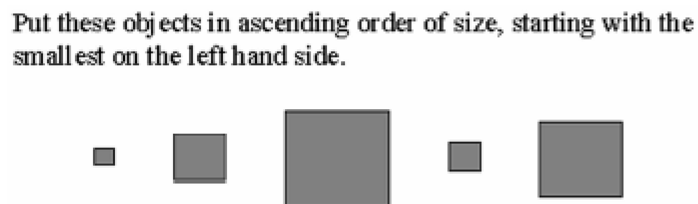


Figura 32 Ordenación de objetos basado en imágenes

Unir los puntos (basado en imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de conectar puntos. El usuario debe pulsar en las áreas apropiadas en la imagen y dibujar la correspondiente figura.

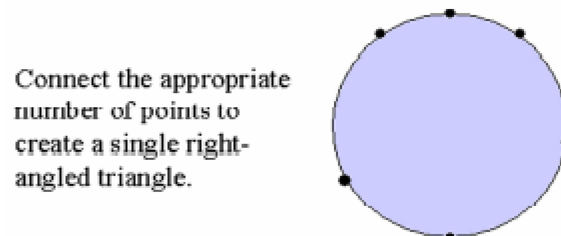


Figura 33 Unión de puntos basado en imágenes

Coordenadas XY

Imagen con zona seleccionable estándar (única imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de imagen con punto caliente. Se espera que el usuario pulse en el área apropiada de la imagen.

Figura 34 Imagen con zonas seleccionables

Unir los puntos (basado en imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de conectar puntos. El usuario debe pulsar en el área apropiada en la imagen y dibujar la correspondiente figura.

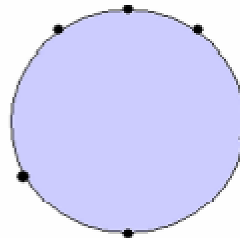


Figura 35 Unión de los puntos basado en imágenes

Cadenas

Rellenar un solo espacio en blanco estándar

Interpretación basada en rellenar huecos.
Típico texto de rellenar los huecos de un texto. Se espera que el usuario teclee la respuesta en el espacio destinado.

Complete the sequence:

Winter, Spring, Summer, _ _ _ _ _ _ .

Figura 36 Rellenar un solo espacio en blanco estándar

Rellenar múltiples espacios en blanco estándar

Interpretación basada en rellenar huecos.
Texto para rellenar huecos con varias entradas. El usuario debe teclear las respuestas en los espacios propuestos para ello.

Fill-in-the blanks in this text from
Richard III:

Now is the ______ of our
discontent made glorious ______
by these sons of ____.

Figura 37 Rellenar múltiples espacios en blanco estándar

Respuestas cortas estándar (requerido texto)

Interpretación basada en rellenar huecos.
Típica cuestión de respuesta corta. Se espera que el usuario teclee el texto en el espacio suministrado.

In less than 100 words describe how you start a car.

Figura 38 Respuesta corta estándar

Números

Rellenar huecos con números enteros o reales estándar

Interpretación basada en rellenar huecos.
Típica cuestión de rellenar huecos con números. El usuario debe escribir el número apropiado en la casilla suministrada.

Give the value of π to three
decimal places:

What is 13 x 13 ? ***

Figura 39 Rellenar huecos con números

Entrada numérica con barra de desplazamiento

Interpretación basada en barra de desplazamiento.
Cuestión para rellenar huecos numéricos con deslizador. Los usuarios deben mover la barra de desplazamiento hasta visualizar la respuesta necesaria.

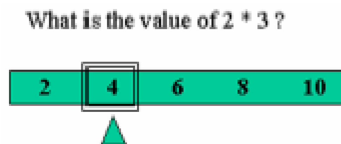


Figura 40 Entrada numérica con barra de desplazamiento

Grupos lógicos

Arrastrar y soltar estándar (imágenes múltiples)

Interpretación basada en objetos.
Cuestión típica de arrastrar y soltar. Se espera que el usuario pulse en la imagen de la respuesta apropiada y la desplace al lugar de respuesta apropiado.

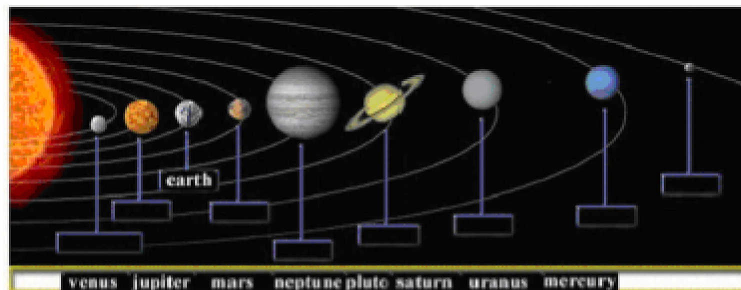


Figura 41 Arrastrar y soltar estándar basado en imágenes

10.3.2.2. Tipos de ítems compuestos

Elección múltiple con rellenar huecos

Elección múltiple con una pregunta adicional de rellenar huecos.
La respuesta basada en cadenas requiere la intervención humana para su corrección.

Which *city* is the capital of *England* and name another city in England ?

Sheffield
London
Manchester
Edinburgh

Another city:

Figura 42 Elección múltiple con rellenar huecos

Respuesta múltiple basada en matriz

Cuestión compuesta de múltiples elecciones, ordenado en una matriz. Solamente se permite una respuesta por fila. El usuario debe elegir una opción por fila.

Which of the following are used to describe the passage of time ?

Hour Gallon Mile
Metre Dozen Decade
Tonne Century Score

Figura 43 Respuesta múltiple basada en matriz

10.3.2.3. Extensiones propietarias

Un requisito clave para esta especificación es su suporte para tipos de respuesta y representaciones propietarias. La manera de cómo se ajusta las extensiones propietarias con la taxonomía de tipos de respuestas esta mostrada en la figura 21.

Las extensiones propietarias pueden usarse como una alternativa al conjunto de los tipos IMS, a las clasificaciones básica o compuesta, o a las clasificaciones Única/Múltiple/Ordenada.

Las extensiones también se pueden encontrar en los tipos de representaciones y formatos mostrados en la figura 22.

10.4. Descripción general del Modelo de Datos.

10.4.1. Información básica.

El modelo de datos de QTI se muestra en la siguiente figura:

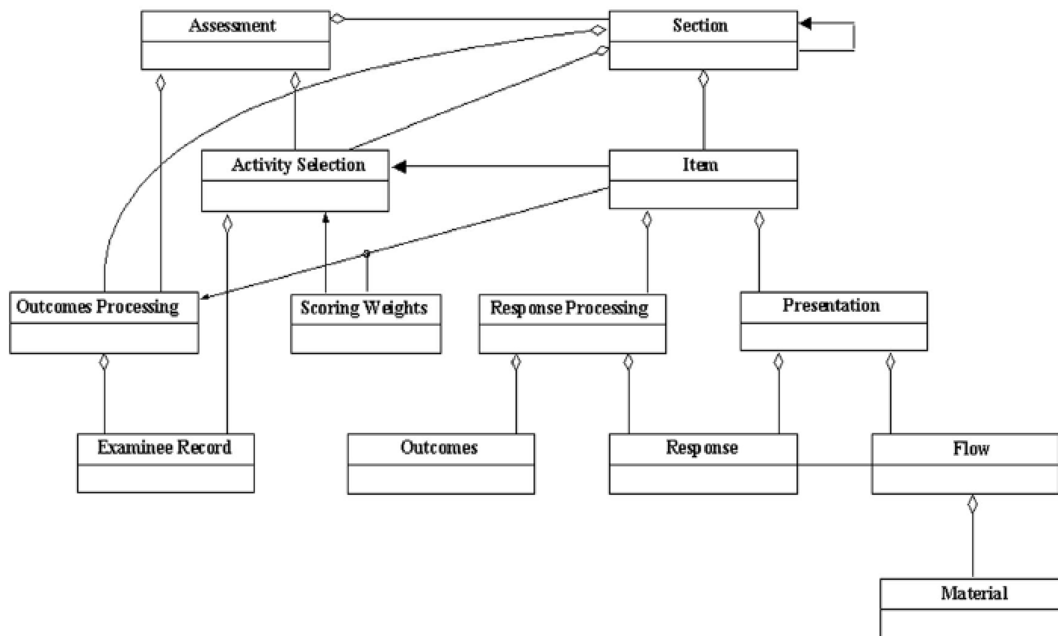


Figura 44 Modelo de objetos de datos de QTI

Los objetos en este modelo y sus claves de comportamiento son:

- *Assessment* (evaluación) – objeto que representa la estructura de datos *Assessment*.
- *Section* (sección) – objeto que representa la estructura de datos *Section*.
- *ítem* – objeto que representa la estructura de datos *ítem*.
- *Activity Selection* (selección de actividad) – selecciona la próxima actividad determinada por el progreso y los resultados obtenidos en el momento de seleccionar la actividad.
- *Outcomes Processing* (procesamiento de salidas) – procesa todas las evaluaciones de salida producidas para conseguir una evaluación general.
- *Scoring Weights* (Pesos de puntuación) – las puntuaciones ponderadas son asignadas a los resultados producidos en el procesamiento de la respuesta.
- *Response Processing* (procesamiento de respuestas) – procesamiento y evaluación de las respuestas de los usuarios.
- *Presentation* (presentación) – interpretación del contenido y posibles respuestas;
- *Examinee Record* (registro del examinado) – conjunto de resultados recopilados que son producidos durante el proceso completo. Esto es un registro permanente en el cual esta contenido el progreso histórico de los individuos.
- *Outcomes* (resultados) – conjunto de resultados que van a ser evaluados en la respuesta del objeto procesado. Esto determina las métricas de puntuación que van a ser aplicadas a las evaluaciones de respuesta.

- *Response* (respuesta) – respuestas que son suministradas por el usuario de los ítems, es decir, la entrada seleccionada por el usuario.
- *Flow* (flujo) – estructura de representación subyacente que define el bloque de relaciones entre los distintos componentes materiales;
- *Material* (material) – contenido que va a ser mostrado.

Esta estructura muestra la relación entre los tres objetos de datos centrales, Ítems, Secciones y Evaluaciones.

La siguiente figura muestra los tipos de objetos que pueden ser intercambiados:

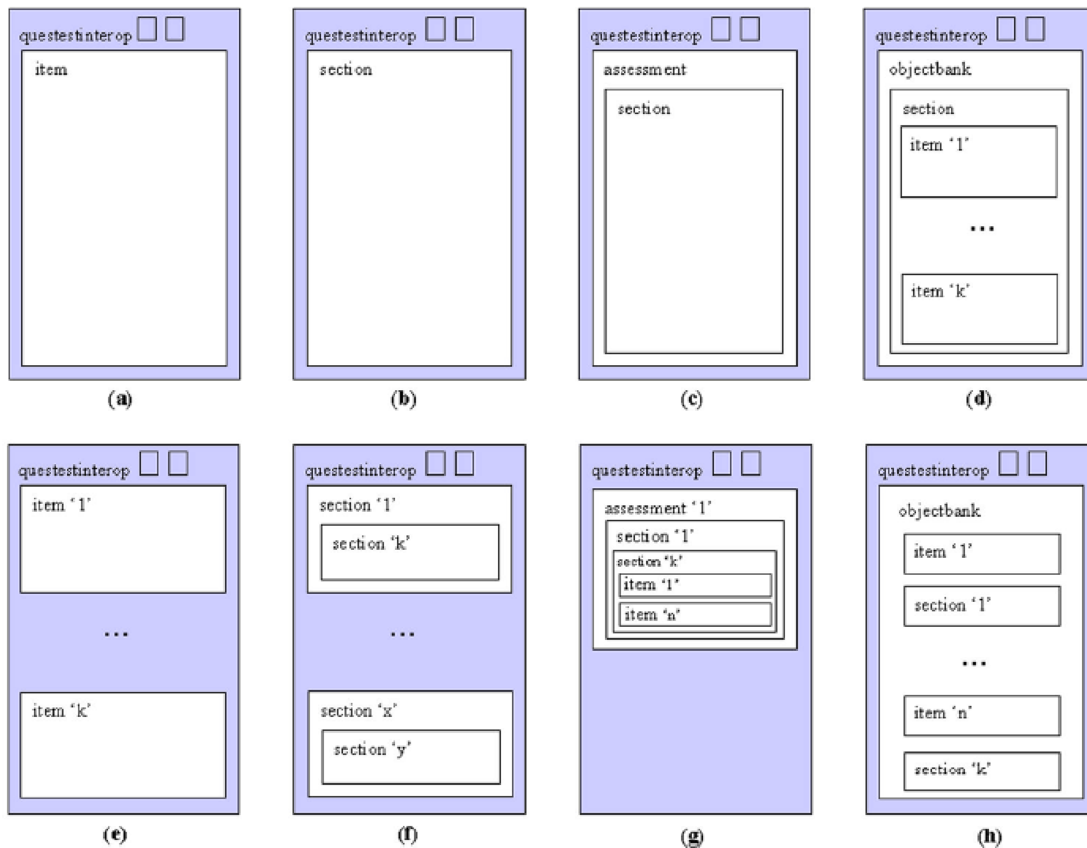


Figura 45 Principio de intercambio de objetos de datos en QTI

Las relaciones entre los distintos objetos de datos se resumen de la siguiente forma:

- Una evaluación contiene al menos una sección (c)
- Una sección puede contener otras secciones (b) y (f)
- Un sección puede contener uno o más ítems (d) y (h)
- Un banco de objetos puede contener solamente ítems, solamente secciones o una mezcla de ítems y secciones.

La definición de la estructura de datos básica a nivel raíz es simple y completamente flexible. La estructura de datos puede usarse para importar/exportar estructuras de datos que posean:

- Una evaluación solamente (c) y (g)
- Una o más secciones solamente (b) y (f)
- Uno o más ítems solamente (a) y (e)
- Una evaluación puede o no contener más de una sección (c) y (g)
- Una sección puede o no contener ítems (b), (c), (f), y (g)
- Un banco de objetos (d) y (h)

Los bancos de objetos (*Object-banks*) son intercambiables mediante la definición de un tipo de paquete QTI, es decir, un conjunto de objetos de datos semejantes contenidos dentro del elemento <*questioninterop*>.

La siguiente figura muestra esquema en árbol genérico de XML:

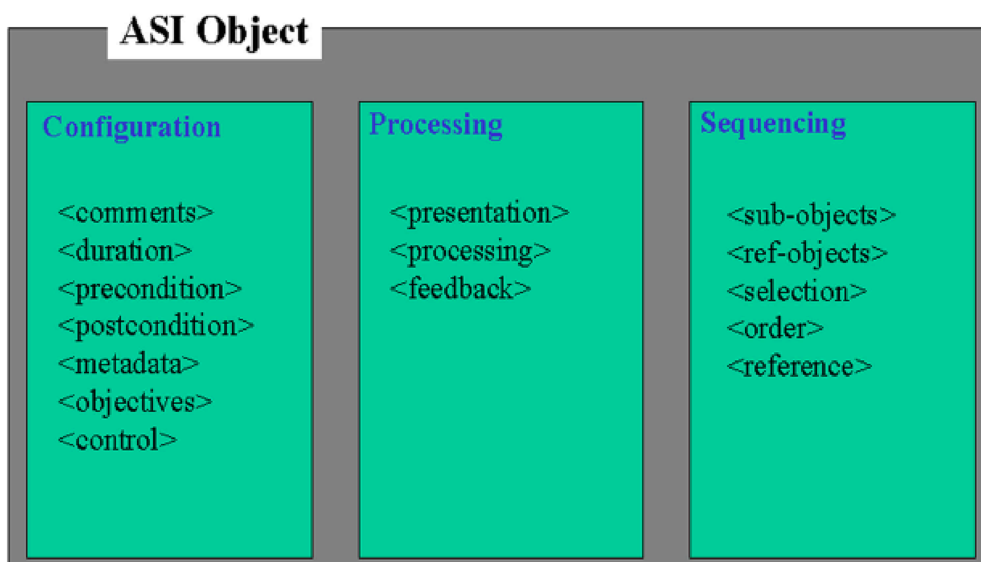


Figura 46 Estructura genérica del esquema en árbol XML

Esta representación refleja la estructura de Ítem, Sección y Evaluación. Esta estructura tiene los tres componentes principales:

- *Configuration* (configuración) – generación del entorno apropiado para la correcta interpretación de la información contenida dentro del objeto.
- *Processing* (procesamiento) – procesamiento actual representado por el objeto, es decir, la representación de una cuestión y el correspondiente procesamiento de respuesta y correcciones.
- *Sequencing* (secuencia) – enlace, vínculo a los objetos referidos y la selección y secuenciamiento del próximo en ser procesado.

10.4.2. Evaluaciones, secciones, ítems y bancos de objetos.

10.4.2.1 Ítems

La siguiente figura muestra el árbol de esquema XML para las estructuras de datos *ítem*:

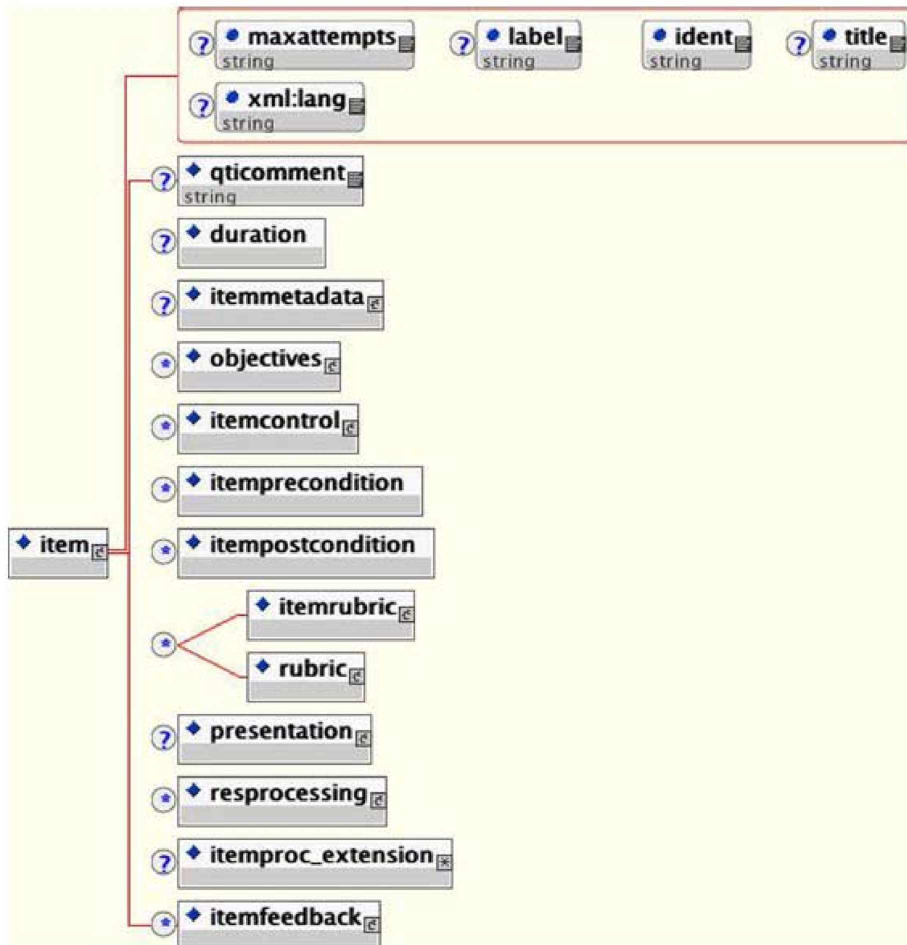


Figura 47 Árbol de esquema XML de ítem

Los árboles de esquema XML correspondientes a los elementos de <*itemmetadata*>, <*presentation*>, <*response_lid*> y <*resprocesing*> son mostrados a continuación:

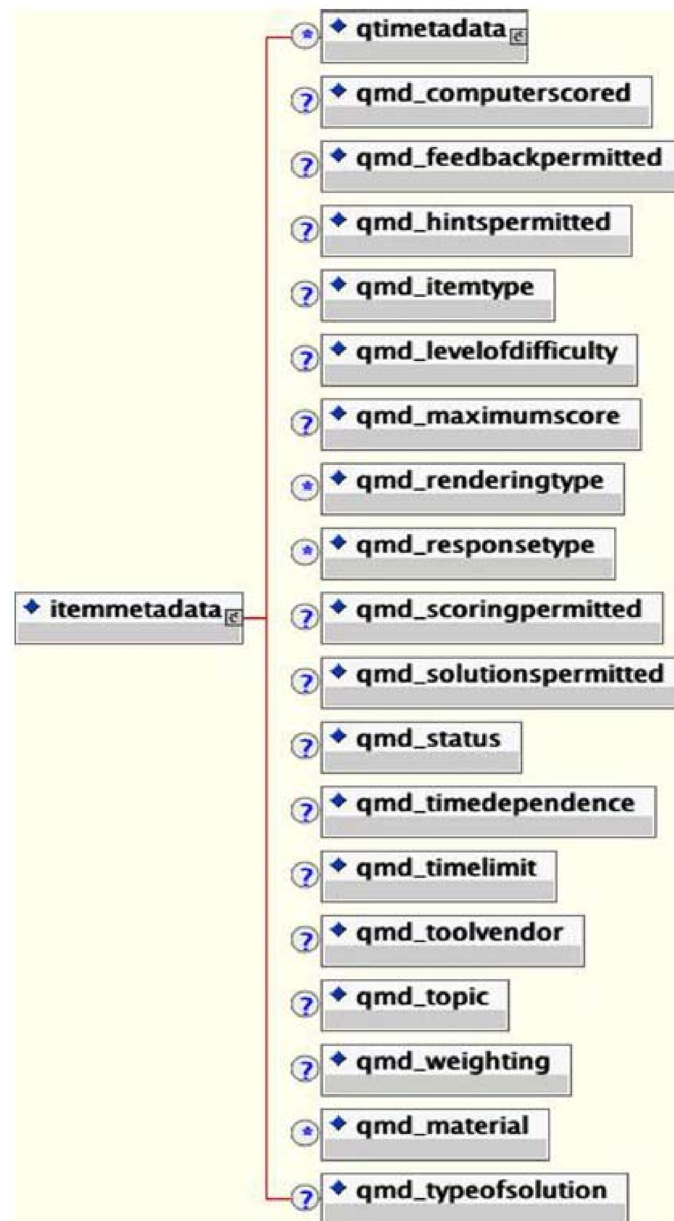


Figura 48 Árbol de esquema XML del elemento Itemmetadata

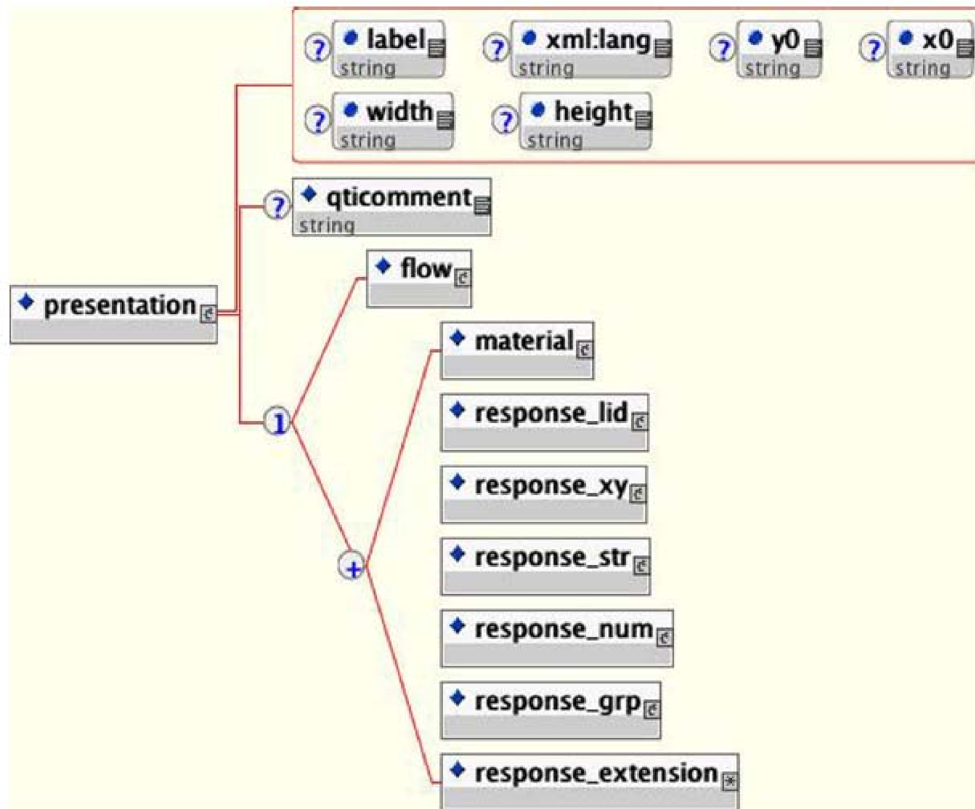


Figura 49 Árbol de esquema XML del elemento Presentation

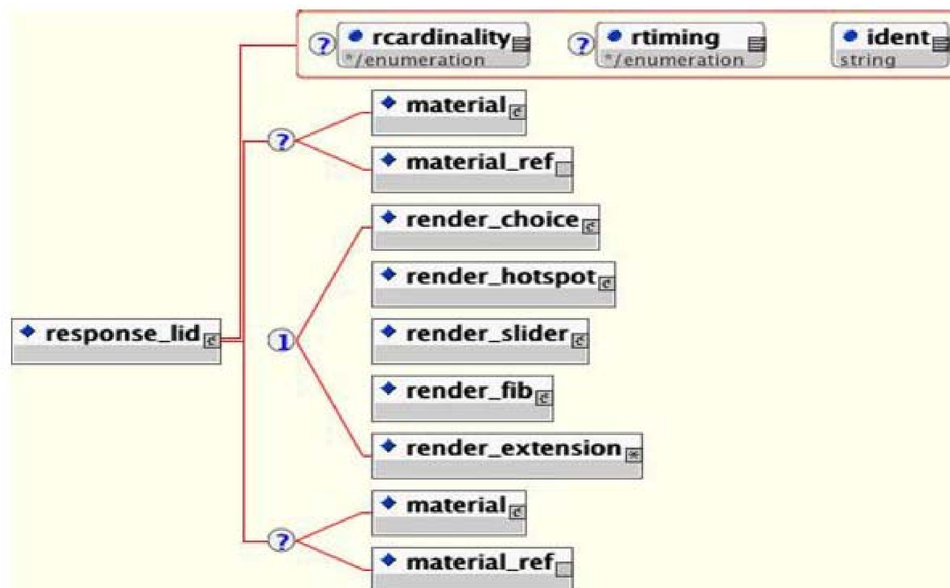


Figura 50 Árbol de esquema XML del elemento Response_lid

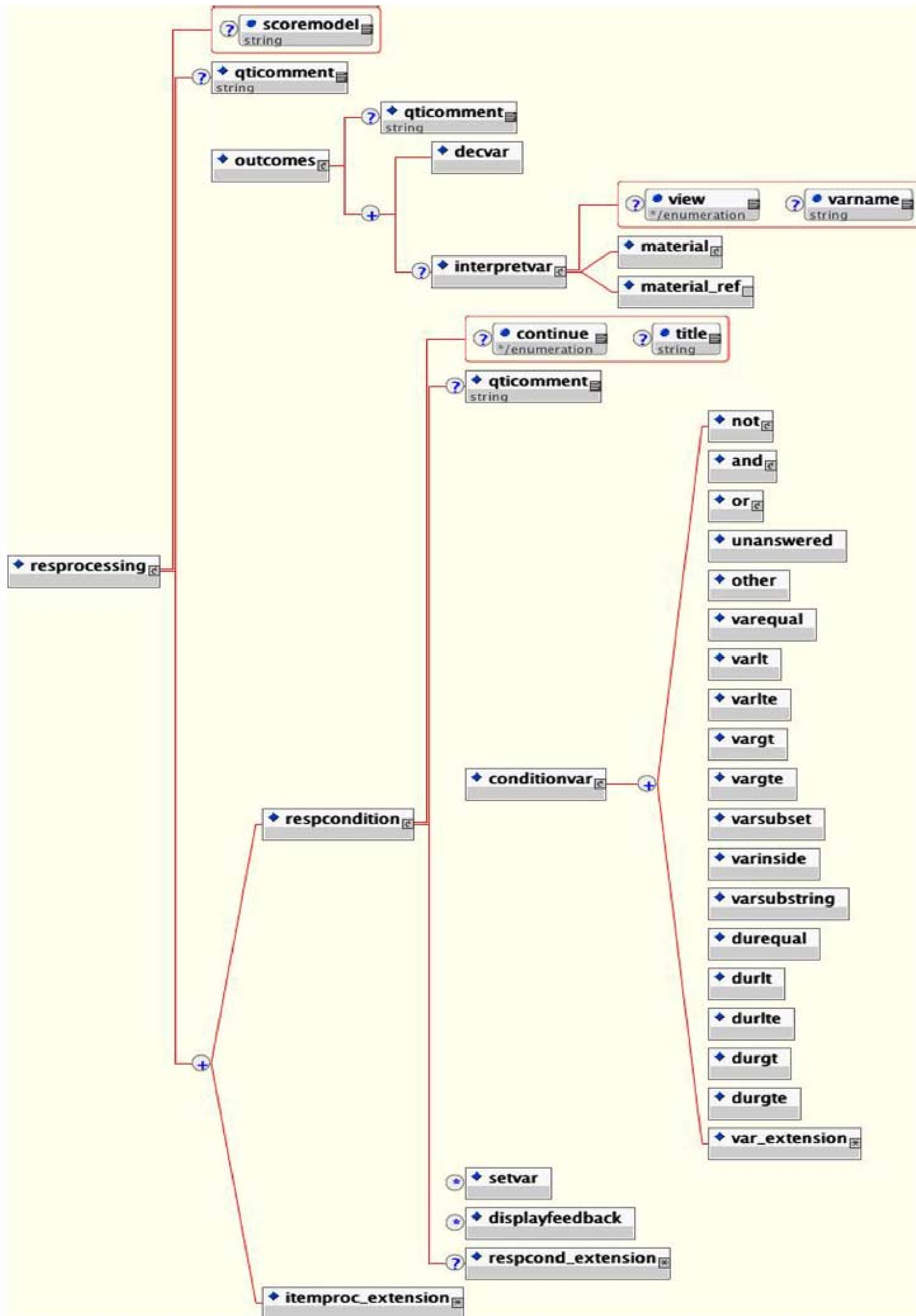


Figura 51 Árbol de esquema XML del elemento Resprocessing

Árbol de esquema XML correspondientes a <*material*> se muestra a continuación:

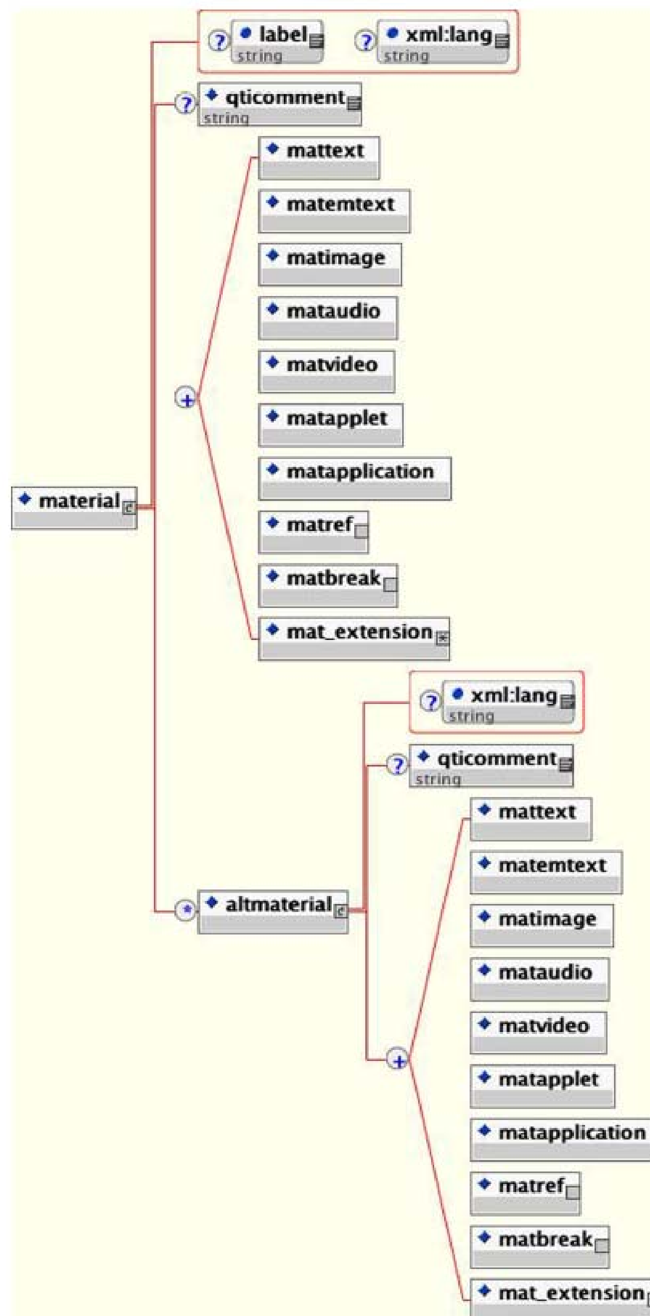


Figura 52 Árbol de esquema XML del elemento Material

Elementos y atributos

- Solamente el elemento *qtimetadata* dentro del elemento *itemmetadata* se debe usar. Esto significa que la especificación IMS QTI sobre la información sobre los propios datos (meta datos) para los ítems debería ser asignada utilizando el vocabulario básico apropiado.

- El elemento *objectives* debe ser usado para definir las objetivos del ítem para cada uno de los actores disponibles. Los objetivos no pueden incluir ningún tipo de contenido y así pueden ser presentados en un amplio rango de formas. El mecanismo por el cual los objetivos son visualizados va más allá del alcance de la especificación IMS QTI.
- El elemento *rubric* debe usarse para presentar material contextual que es aplicado a un conjunto de ítems contenidas. Estas descripciones pueden ser suministradas para cada vista que es soportada. El elemento <*itemrubric*> es una alternativa soportada. El título para las vistas *All* y *Participant* debe ser siempre visualizado. En los casos donde hay demasiada información para ser visualizada se debería usar algún mecanismo de visualización por partes para permitir al participante acceder a la información ya lista, sin importar que pagina está viendo.
- El elemento *Itemcontrol* debería ser usado para definir las condiciones por defecto para la exposición de diferentes tipos de corrección a los usuarios. Las definiciones del nivel *ítem* de *feedbackswitch*, *hintswitch* y *solutionswitch* tienen preferencia sobre todos los demás niveles de definición.

Grupos de elementos

- Tipos de respuesta e interpretación:

Hay cinco tipos básicos de respuesta. Es importante darse cuenta que el tipo de respuesta está determinado por la manera en la que es procesado internamente. Un único punto puede tener más de un tipo de respuesta, es decir, crear un tipo de respuesta compuesto. Si un nuevo tipo de respuesta es definido entonces puede ser añadido usando la extensión apropiada *response_extension*. El tipo clásico de pregunta de elección múltiple y respuesta múltiple puede ser soportado usando el elemento *response_lid*.

Hay cuatro tipos de interpretación: *render_choice, render_hotspot, render_slider* y *render_fib*. Es importante notar que el tipo de interpretación está impuesto sólo indirectamente por el tipo de respuesta pero está muy ligado al objetivo educacional de la pregunta. Si se identifica un nuevo tipo, entonces se puede añadir usando el elemento de extensión apropiado *render_extension*. Los tipos clásicos de preguntas de elección múltiple o de respuesta múltiple pueden ser soportados usando *render_choice*.

El conjunto de tipos de respuesta e interpretación hace posible 20 combinaciones distintas. La inclusión de los atributos de cardinalidad y de tiempo hace que haya un total de 140 posibles combinaciones. No es posible examinar el tipo de respuesta, interpretación, cardinalidad y medición para asegurar automáticamente el tipo de la pregunta (si el tipo de la pregunta es una información requerida entonces debe ser suministrada usando el campo IMS QTI con la meta-data especifica *qmd_questiontype*). Esta ingeniería inversa no es posible porque algunas combinaciones pueden ser usadas para dar soporte a más de un tipo de preguntas.

- *Flows* (Flujos)

Hay tres elementos que dan soporte a las capacidades de párrafos y bloques en QTI, llamado flujo, *flow_label* (etiqueta de flujo) y *flow_mat* (etiqueta de material).
La manera en la que el párrafo/bloque se presenta depende del motor de presentación pero se asume que estos motores de presentación serán capaces de procesar cada uno de estos elementos en los diferentes modos en que son combinados de un modo consistente. Se espera que este motor sea capaz de procesar flujos desde el contenido que no hace uso de ellos; no debería asumirse que todo motor sea capaz de tales diferenciaciones.

- *Itemfeedback* (Corrección de Ítems)

El elemento *itemfeedback* contiene los elementos de pista y solución y sus contenidos se disparan usando el elemento *itemfeedback*. Este elemento puede contener varios conjuntos de consejos (pistas), soluciones y respuestas de corrección estándar. La diferenciación entre éstos se realiza usado el atributo *feedbacktype* y el *feedbackstyle*. El primero define el tipo de corrección que será mostrada (pista, solución, o respuesta) mientras que el último indica como el material de corrección será revelado, es decir, incrementalmente, etc.

- *Variable Manipulation* (Manipulación de Variables)

La manipulación de las variables de puntuación declaradas en la combinación *outcomes*/*decvar* está contenida dentro del elemento *conditionvar*. La comparación de variables se hacen individualmente usando los elementos definidos como *varequeal*, etc. y el estado de estas comparaciones se puede invertir usando el elemento lógico NOT. El análisis del periodo de respuestas es soportado usando los elementos *durequal*, etc. (no se asume una variable por defecto asociada). La combinación de los elementos individuales *varequal*, etc. es posible usando dos técnicas:

  - Implícita – la secuencia de elementos *varequal*, etc. dentro de un elemento *conditionvar* es por definición con una condición AND. El uso de múltiples elementos *conditionvar* es también tratado como una condición AND. La condición OR es conseguida a través del uso de múltiples elementos *respcondition*. La secuencia de estos es equivalente a una condición lógica inclusive OR;
  - Explicita – el uso de los elementos lógicos AND y OR que combina los resultados de cada comparación por separado y los combina en una declaración de estado consolidado para el elemento *conditionvar*.

Es recomendado que el enfoque implícito se use siempre que sea posible. Este enfoque proporciona un código más interoperable.

Las variables de procesamiento de respuesta se declaran usando el elemento *decvar*. Cada implementación del IMS QTI debe generar una variable entera por defecto llamada SCORE cuyo valor por defecto es cero. Esta variable se usa siempre que un test condicional se aplica y el correspondiente *setvar* no incluye un nombre particular de variable. Cuando se soporta el procesamiento de una respuesta, hay dos condiciones especiales que necesitan ser tratadas:

  - Cuando la respuesta no ha sido contestada – esto puede ser soportado usando dos técnicas, a saber, el elemento *unanswered* o el elemento *response_na*. El elemento *unanswered* es colocado dentro de

conditionvar y es activado siempre que la respuesta no se haya intentado. El elemento *response_na* es una facilidad para la extensión propietaria que esta presente en cada una de la interpretación de elementos, es decir, *render_choise, render_hotspot,* etc.

- Cuando ninguna de las condiciones en el elemento *conditionvar* se satisface, se necesita declarar un estado por defecto. Este estado es capturado usando el elemento *other*. Cuando está incluido en *conditionvar* entonces se devolverá *verdadero* si ninguna otra condición se ha invocado.

10.4.2.2. Section

La siguiente figura muestra el árbol de esquema XML para las estructuras de datos *Section*:

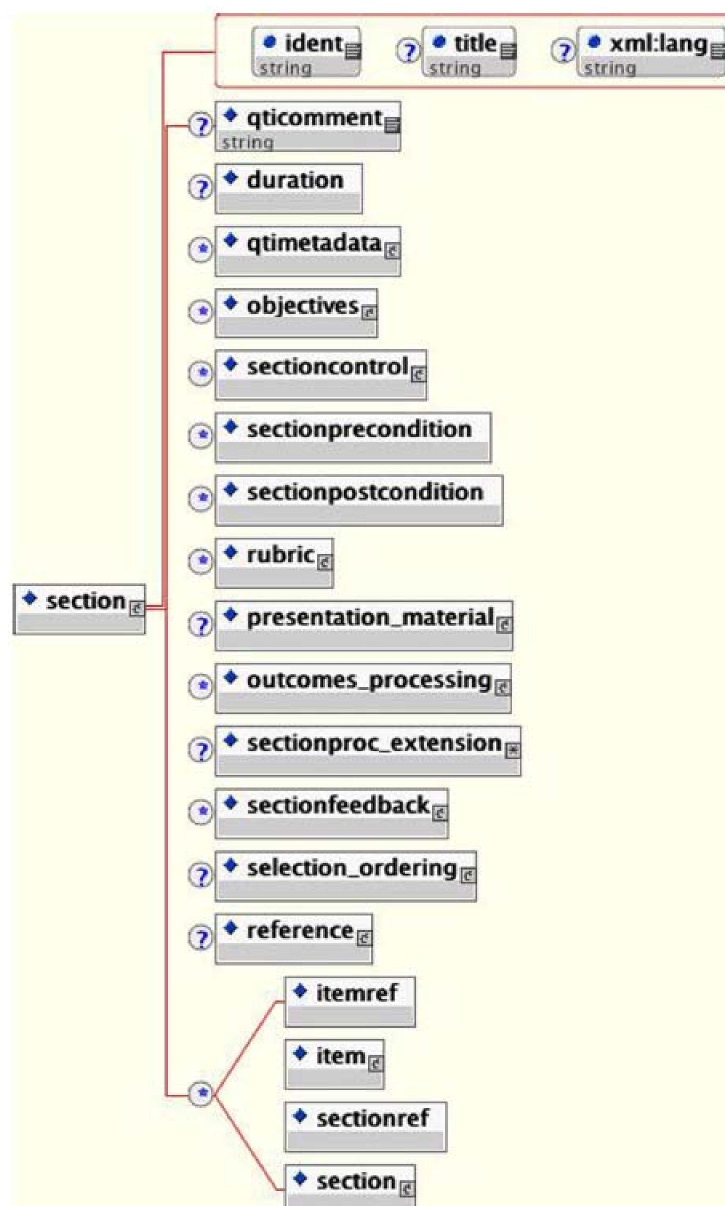


Figura 53 Árbol de esquema XML del elemento section

Elementos y atributos

- La especificación IMS QTI de *metadata* (es decir la especificación sobre los propios datos) para secciones se debe designar usando el vocabulario básico apropiado.
- El elemento *objectives* debe ser usado para definir las objetivos de la sección para cada uno de los actores disponibles. Los objetivos no pueden incluir ningún tipo de contenido y así pueden ser presentados en un amplio rango de formas. El mecanismo por el cual los objetivos son visualizados va más allá del alcance de la especificación IMS QTI.
- El elemento *rubric* debe ser usado para presentar material contextual que se aplica a un conjunto de secciones/ítems contenidas – esta información no debería ser usada para contener una cuestión actual o un componente de una cuestión. Estas descripciones pueden ser suministradas para cada vista que es soportada. El título para las vistas *All* y *Participant* debe ser siempre visualizado. En los casos donde hay demasiada información para ser presentada se debería usar algún mecanismo de visualización por partes para permitir al participante acceder a la información ya lista, sin importar que pagina está viendo.
- El elemento *Sectioncontrol* debería ser usado para definir las condiciones por defecto para la exposición de tipos de correcciones de los usuarios. Las definiciones del nivel *Section* de *feedbackswitch*, *hintswitch* y *solutionswitch* tienen prioridad si no se encuentran definiciones en un nivel más bajo, es decir, dentro de un *ítem*. En caso de conflicto con una definición del nivel *Assessment,* las definiciones de nivel *Section* tienen prioridad.
- *Presentation_material* – el material contenido en este elemento debe usarse para almacenar información que sea relevante para cada uno de los objetos de evaluación contenidos en la sección. Este material debe ser visualizado para todos los participantes. No debería usarse para contener información como título y objetivos, puesto que hay elementos separados para este tipo de información.
- *Outcomes_processing* – ver http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_outv1p2.html.
- El elemento *Sectionfeedback* puede almacenar contenidos que se usan para informar al participante de las calificaciones de sus respuestas, es decir, no puede ser usado para contener pistas y soluciones definidas dentro del contexto IMS QTI. La corrección se realiza a través del elemento *Outcomes_processing*.
- *Reference* – el material que es común en muchos objetos de evaluación dentro de objetos de evaluación, como por ejemplo un fragmento de texto, una imagen, etc. se almacena en este elemento. Este material es entonces referenciado desde dentro del elemento *material* apropiado. El material en este contenedor puede que no se visualice a menos que el elemento que referencia sea el mismo que se dispara.

Grupos de elementos

- *Selection & Ordering* – La selección y secuenciamiento de secciones se soporta por medio de un mecanismo de preselección usando el elemento *selection_ordering* que define las secciones/ítems seleccionados desde el conjunto de Secciones disponibles y el orden en cada una de estas secciones/ítems se activan respectivamente. Ver http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_saov1p2.html.
- *Variable Manipulation* – se refiere al proceso de agregación de puntuaciones para la sección. Ver http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_outv1p2.html.

10.4.2.3 Assessment

La siguiente figura muestra el árbol XML para las estructuras de datos *Assessment*:

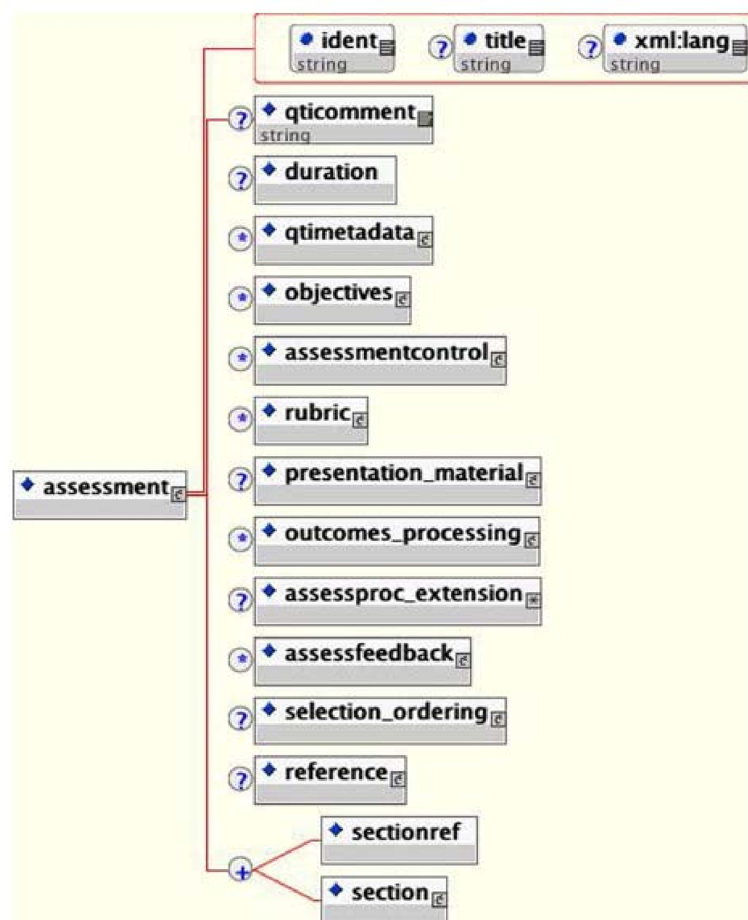


Figura 54 Árbol de esquema XML del elemento assessment

Elementos y atributos

- La especificación IMS QTI de *metadata* para la evaluación (*assessment*) se debe designar usando el vocabulario básico apropiado.
- El elemento *objectives* debe usarse para definir los objetivos de la evaluación para cada uno de los actores disponibles. Los objetivos no pueden incluir ningún tipo de contenido, para que así se pueden presentar en un amplio rango de formas. El mecanismo por el cual los objetivos son visualizados va más allá del alcance de la especificación IMS QTI.
- El elemento *rubric* debe ser usado para presentar material contextual que es aplicado a un conjunto de secciones/ítems contenidas – esta información no debería ser usada para contener una cuestión actual o un componente de una cuestión. Estas descripciones pueden ser suministradas para cada vista que es soportada. El título para las vistas *All* y *Participant* debe ser siempre visualizado. En los casos donde hay demasiada información para ser visualizada se debería usar algún mecanismo de visualización por partes para permitir al participante acceder a la información ya lista, sin importar que pagina está viendo.
- El elemento *assessmentcontrol* debería ser usado para definir las condiciones por defecto para la presentación de tipos de corrección de los usuarios. Las definiciones del nivel *Assessment* de *backswitch*, *hintswitch* y *solutionswitch* tienen prioridad si no se encuentran definiciones en un nivel más bajo, es decir, dentro de una sección o un ítem. Esto significa que el nivel de definición *Assessment* actúa como estado por defecto.
- *Presentation_material* – El contenedor material en este elemento es usado para contener información que es relevante para cada uno de los objetos de evaluación contenidos en *Assessment*. Este material se debe visualizar para todos los participantes. No se debería usar para contener información como el título o los objetivos puesto que ya hay elementos separados para este tipo de información.
- *Outcomes_processing* – ver http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_outv1p2.html.
- El elemento *assessfeedback* puede almacenar contenidos que se usan para informar al participante de las calificaciones de sus respuestas, es decir, no puede ser usado para contener pistas y soluciones definidas dentro del contexto IMS QTI. La corrección se realiza a través del elemento *Outcomes_processing*.
- *Reference* – el material que es común en muchos objetos de evaluación dentro de objetos de evaluación, como por ejemplo un fragmento de texto, una imagen, etc. se almacena en este elemento. Este material es entonces referenciado desde dentro del elemento *material* apropiado. El material en este contenedor puede que no se visualice a menos que el elemento que referencia sea el mismo que se dispara.

Grupos de elementos

- *Selection & Ordering* – La selección y secuenciamiento de evaluaciones se soporta por medio de un mecanismo de preselección usando el elemento *selection_ordering* que define las secciones/ítems seleccionados desde el conjunto de secciones disponibles y el orden en cada una de estas secciones/ítems se activa respectivamente.
- *Variable Manipulation* – se refiere al proceso de agregación de puntuaciones para la Assessment. Ver http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_outv1p2.html.

10.4.2.4 Object Banks

Un banco de objetos es un conjunto de objetos de evaluación que están agrupados juntos e intercambiados en un paquete QTI etiquetado como *Banco de Objetos*. Este etiquetado se consigue asignándole un identificador de objetos banco usando el atributo IDENT. Sólo se puede referenciar el objeto banco entero (desde dentro del elemento *selection_ordering*) y no un componente individual del banco. Este mecanismo permite referenciar fácilmente un gran grupo de elementos agrupándolos como un objeto banco. Actualmente un objeto banco puede consistir en un combinación de secciones y/o ítems (elementos). Un objeto banco puede también tener meta datos asignados mediante el elemento *qtimetadata*.

10.4.3. Contenidos

10.4.3.1. Flujos

El siguiente ejemplo muestra la importancia de los flujos a través de la visualización de bloques múltiples de <*material*>.

Fill-in-the blanks in this text from
Richard III:

Now is the _ _ _ _ _ _ of our
discontent made glorious _ _ _ _ _ _
by these sons of _ _ _ _ .

Figura 55 Flujos

En este ejemplo hay tres respuestas FIB (*fill in blank*) contenidas en un único ítem. Este texto y el texto de la pregunta introductoria son contenidos en varios elementos <*material*>. El problema es que no está clara la semántica de bloque definida para el elemento <*material*> y por tanto es confuso como la primera sentencia debe ser definida como un párrafo separado. Para resolver esta cuestión se introduce el concepto de flujo.

Un flujo es definido como un conjunto de contenido para ser gestionado por el motor de visualización como un bloque lógico o un párrafo. Un flujo puede contener otros flujos y por tanto se puede construir un sistema complejo de flujos jerárquicos. En el caso del ejemplo anterior tendremos dos flujos o bloques.

Cuando se utilizan estos flujos se ha de indicar mediante el elemento <*flow*> colocado dentro del elemento <*presentation*>. Con cualquier construcción de flujo se ha de usar <*flow*> dentro del elemento <*presentation*>. Los tres elementos que soportan flujos son:

- <*flow*> – indica el nivel superior de flujo dentro del elemento <*presentation*>
- <*flow_label*> – encapsula el elemento <*response_label*>
- <*flow_mat*> – encapsula el elemento <*material*>

Cualquiera de los tres casos los elementos puede ser recursivo, por ejemplo, <*flow_mat*> dentro de <*flow_mat*>, y las reglas de bloque deben ser definidas e implementadas consistentemente.

El uso de flujos permite controlar las cuestiones de diseño como por ejemplo:

- Párrafos para texto
- Mezclar texto y espacios para respuestas, como el tipo de respuesta FIB múltiple para un único ítem.
- Alineación de listas, incluyendo la alineación horizontal y vertical de las opciones disponibles en una pregunta múltiple elección.

Es recomendable que los ítems utilicen el enfoque de flujo. Permite una semántica más clara para controlar el diseño de los ítems y una mayor facilidad de soporte de técnicas como XML *Style-sheets*. Sin embargo, esto no debe sobrecargar las cuestiones de estilo y que se oponga a nuestro principal objetivo de interoperabilidad funcional. Nuestra intención es permitir al autor influenciar en el diseño sin crear complicaciones en el motor de visualización, por ejemplo, diferentes fuentes de texto dará lugar a complicaciones significativas que han de ser evitadas permitiendo al motor de visualización usar su propio conjunto de valores por defecto.

10.4.3.2. Texto

La mayor cantidad de contenido a visualizar es de tipo texto. La especificación de QTI posee características para gestionar el contenido basado en texto, llamadas:

- *Mime type* – el valor por defecto es *text/plain*
- Conjunto de caracteres – indica al sistema la naturaleza del texto. El valor por defecto es *us-ascii*. En XML existe dos formatos de codificación: UTF-8 (por defecto) y UTF-16.
- Idioma – permite que el texto este disponible en una variedad de diferentes idiomas. Este mecanismo proporciona el contenido de un mismo ítem en uno o mas lenguajes utilizando <*almaterial*> por cada lenguaje dentro del ítem.
- Gestor de espacios en blanco – para ello ha de utilizarse el atributo *xml:space*. Por defecto no preserva los espacios en blanco.

- Énfasis – distingue partes de concretas de otras, se utiliza el elemento <*matemtext*>. La manera por la cual es obtenido, por ejemplo negrita, cursiva... se deja al motor de visualización.
- Párrafos – utiliza la estructura de flujo, el elemento <*matbreak*> que coloca una pausa en el material.
- Posición del texto – puede ser controlado por los atributos x_0, y_0 para indica la esquina superior izquierda del bloque. La altura y ancho del texto es controlado por los atributos *Height* y *Width*.

Es importante remarcar que las cuestiones estilísticas pertenecientes a la fuente del texto se encuentren fuera del alcance de la especificación. Estas cuestiones han de ser controladas usando hojas de estilo XSL, ficheros referenciados externamente como HTML, etc.

10.4.3.3. Imágenes

La presentación de imágenes requiere la especificación de un punto de referencia (*anchor point*). Este punto está definido por las coordenadas de la esquina superior izquierda en términos x_0, y_0. También es necesario otros dos atributos: *pixel-height* (altura) y *pixel-width* (anchura) de la imagen.

Un ejemplo de imágenes múltiples se encuentra en la siguiente figura. Cada imagen posee su propio tamaño y localización definido por x_0, y_0, anchura y altura. Los valores de los puntos x_0, y_0 se encuentra definidos respecto a la esquina superior de la pantalla. Los puntos seleccionables de cada imagen se especifican de la misma manera. En el caso de coincidir las imágenes el orden de precedencia esta marcado por el orden de los elementos *response_label* (posee mayor precedencia el que se haya declarado antes).

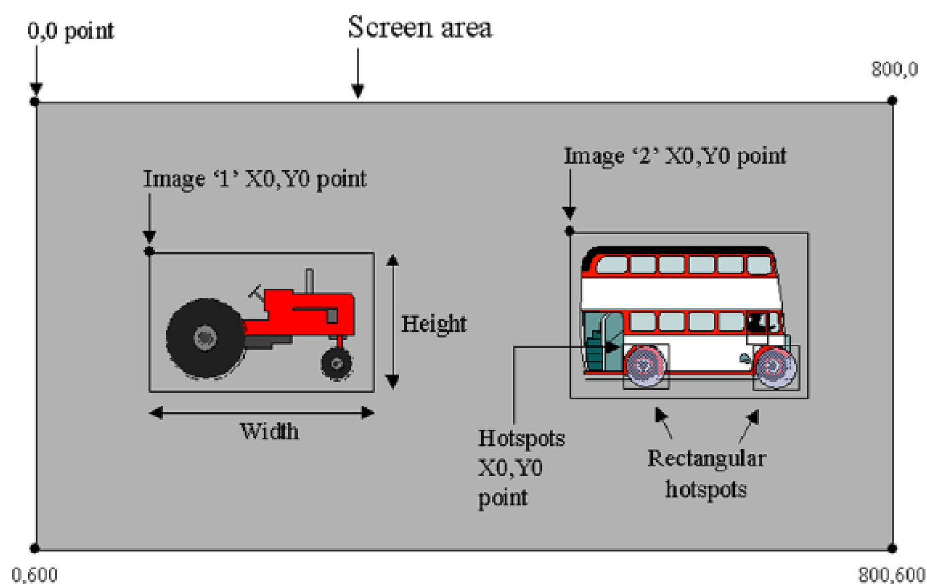


Figura 56 Diseño de imágenes múltiples.

10.4.3.4. Contenido referenciado

Se puede almacenar material y después ser referenciado en diferentes lugares dentro de Evaluación, Sección e Ítem. El contenido del material ha de colocarse dentro de un objeto <reference> (disponible para evaluaciones y secciones). Si está disponible el material a un nivel, todos sus hijos también pueden referenciar a dicho material, es decir, si está definido a nivel de sección, también lo estará para todos las secciones y/o ítems dentro de él.

Para referenciar al contenido especificado en *<reference>* se puede utilizar tanto el elemento *<matref>* como *<material_ref>*. El elemento *<matref>* se utiliza para identificar un componente concreto del contenido, como por ejemplo *<mattext>*, *<matimage>*... En cambio el elemento *<material_ref>* referencia por completo la estructura *<material>*.

10.5. La sesión de un ítem

La clase que representa la sesión de un ítem (*itemSession*) es una clase abstracta que ayuda a ilustrar los requisitos de los procesos de reparto cuando se entreguen los ítems de los candidatos.

La sesión contiene el atributo *completionStatus* que almacena el estado de dicha sesión. Inicialmente contiene el valor "*not_attempted*" (el usuario no ha realizado ningún intento). Una vez que se ha producido el primer intento *completionStatus* toma el valor de "*unknown*" (valor desconocido). Permanece con este valor hasta que la sesión del ítem finalice o es modificado por *setOutcomeValue* durante el procesamiento de la respuesta. Hay cuatro posibles valores permitidos: *completed* (completado), *incomplete* (incompleto), *not_attempted* (ningún intento) y *unknown* (desconocido). La variable *completionStatus* puede cambiar a cualquiera de estos cuatro valores durante el procesamiento de la respuesta.

Si un ítem de tipo adaptativo asigna a la variable *completionStatus* el valor "*complete*" entonces la sesión ha de encontrarse en el estado *closed* (cerrado). Sin embargo, una sesión no tiene que esperar a la señal de *complete* para interrumpir la sesión, por ejemplo puede finalizar como respuesta a una petición directa del candidato, superar el tiempo máximo o por cualquier otra circunstancia excepcional. Los ítems de tipo No-adaptativo no requieren colocar un valor para *completionStatus*, mientras que los ítems de tipo Adaptivo deben mantener un valor adecuado y deberían establecer en *completionStatus* el valor "*complete*" para indicar cuando el ciclo de interacción, el procesamiento de las respuestas y los resultados de las evaluaciones deben parar.

La sesión almacena los valores que han sido asignados a todas las variables del ítem. Los valores de *completionStatus* y *duration* son tratados como variables de ítem especiales. Ellas comparten el mismo espacio de nombres como las variables de ítem que son explícitamente declaradas a través de las declaraciones de variable (*variableDeclarations*).

La sesión está asociada a un contexto (*sessionContext*) que proporciona información sobre el candidato, cuando y donde la sesión toma lugar, etc.

El siguiente diagrama ilustra los estados de la sesión de un ítem. No todos los estados pueden ser aplicados a cada escenario, por ejemplo el estado *Feedback* (corrección) quizás no exista para un ítem o no es permitido en un contexto donde el ítem esta siendo utilizado. Igualmente, el candidato puede que no tenga permiso para revisar sus respuestas y/o examinar una solución. En la práctica, quizás los sistemas solamente soportan un limitado número de transiciones de estados y/o soporten otras transiciones que no se encuentren aquí.

Un primer paso para los desarrolladores de sistemas en determinar que requisitos poseen su aplicación es identificar cuales de los estados son soportados en el sistema y como encajan las transiciones de estados indicadas en el diagrama con su propio modelo de eventos.

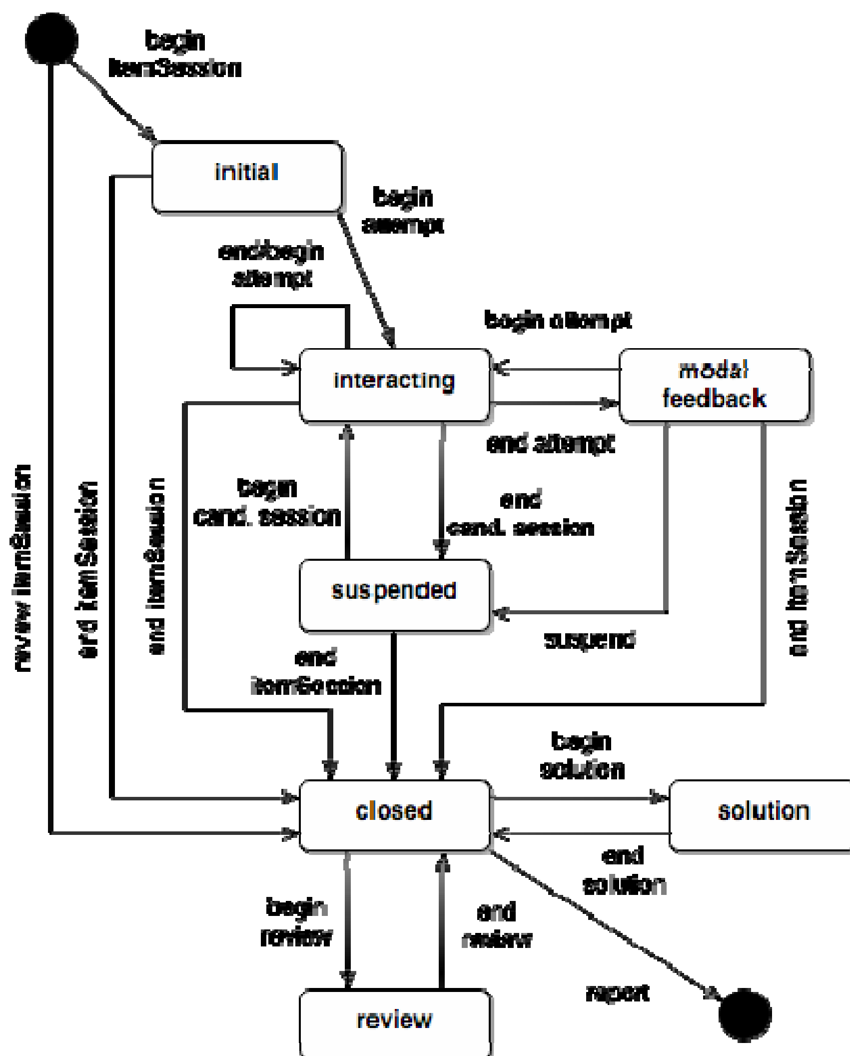


Figura 57 Ciclo de vida de la sesión de un ítem (*itemSession*)

El proceso de reparto crea una instancia de una sesión una vez que el ítem es apto para ser entregado al candidato. Entonces el estado de la sesión se mantiene y se actualiza con respecto a las acciones del candidato hasta que la sesión finalice. En este punto el estado cambia a informe de sesión. El proceso de reparto también puede permitir un

informe desde una sesión ya anterior para ser utilizado en la re-creación de dicha sesión y así permitir que las respuestas de un candidato sean vistas en el contexto del mismo ítem (y posiblemente compararlas con la solución).

El estado inicial de una sesión de un ítem representa el estado posterior a que se haya decidido que el ítem sea entregado al candidato pero anterior a dicha entrega.
En una evaluación no-adaptativa los ítems son seleccionados con antelación y el informe de las interacciones del candidato con todos los ítems se realiza al final de la evaluación, independientemente de cuando el candidato haya realizado los intentos para todos los ítems. En realidad, se crean las sesiones para todos los ítems en el estado inicial y se mantienen en paralelo.

En una evaluación de tipo adaptativa los ítems que estarán presentes son seleccionados durante la sesión basándose en las respuestas y salidas asociadas a los ítems presentados hasta el momento. Los ítems son seleccionados de un gran conjunto y el proceso de reparto solamente informa de las interacciones de los candidatos con los ítems que realmente si han sido seleccionados.

La interacción de un candidato con un ítem puede ser cero o más intentos. Durante cada uno de los intentos el candidato interactúa con el ítem a través de una o más sesiones. Al final de la sesión del candidato el ítem es colocado en un estado suspendido y preparado para la próxima sesión del candidato. Durante la sesión del candidato la sesión del ítem se encuentra en el estado de interacción. Una vez que el intento haya finalizado se realiza el procesamiento de la respuesta, después puede iniciarse un nuevo intento.

Para ítems no-adaptativos el procesamiento de la respuesta solamente puede ser invocado por un número limitado de veces, normalmente una sola vez. Para ítems adaptativos tal límite no es necesario porque el procesamiento de la respuesta adapta los valores que asigna a las variables de salida basando en una ruta a través del ítem. En ambos casos, cada invocación del procesamiento de respuestas indica el fin del intento. La apariencia del cuerpo del ítem, y cualquiera que sea la solución mostrada, esta determinado por los valores de las variables de salida (*outcomeVariables*).

Cuando no se permitan más intentos la sesión del ítem pasa al estado de *closed* (cerrado). Una vez situada en este estado los valores de las variables respuestas son fijados. Un sistema de reparto o de informe todavía podrá permitir al ítem ser presentado después de alcanzar el estado de *closed*. Este tipo de presentación se realiza en el estado *review* (revisión), las correcciones también pueden ser visibles en este punto si el procesamiento de respuestas haya tenido lugar y establece unas variables de salida adecuadas.

Por último, la sesión del ítem podrá pasar al estado *solution* (solución) para los sistemas que soportarán la visualización de las soluciones. En este estado, se reemplazará temporalmente las respuestas del candidato por los valores correctos obtenidos en las declaraciones de la respuesta (*responseDeclarations*) correspondientes o por el valor NULL si ninguna ha sido declarada.

10.6. El Procesamiento de la respuesta

El procesamiento de respuesta es aquel proceso por el cual el sistema de entregas asigna los resultados a la sesión del ítem basado en las respuestas del candidato. Los resultados pueden ser utilizados para proveer la corrección al candidato. La corrección puede ser entregada inmediatamente después de que finalice el intento del candidato o un tiempo más tarde, quizás como parte del informe sobre la sesión del ítem.

El fin de un intento, y por tanto del procesamiento de la respuesta, debe únicamente realizarse en una respuesta directa a una acción del usuario o en una respuesta a algún evento esperado, tales como el fin de una evaluación. Una sesión que entra al estado de suspendido quizás tenga valores para las variables de las respuestas que tienen todavía que ser enviadas para el procesamiento de la respuesta.

Para un ítem de tipo no-adaptativo se restaura los valores de las variables de salida a sus valores por defecto (o Null si no existe) antes de cada invocación al procesamiento de respuesta. Sin embargo, aunque el proceso de entrega puede invocar varias veces el procesamiento de la respuesta para un ítem no adaptativo únicamente debe informar sobre el primer conjunto de resultados producidos o limitar el número de intentos a algún límite predefinido acordado fuera del ámbito de esta especificación.

Para un ítem de tipo adaptativo no se restaura los valores de las variables de salida a sus valores por defecto. Un proceso de entrega que soporta los ítems adaptativos debe permitir al candidato revisar y entregar sus respuestas al procesamiento de corrección y debe solamente informar del último conjunto de resultados producidos. Además, debe presentar todas las correcciones aplicables e integradas al candidato. El procesamiento de respuesta posterior puede considerar que la corrección es conocida por el candidato cuando los resultados de la sesión son actualizados. Un ítem de tipo adaptativo puede indicar al proceso de entrega que el candidato ha completado la interacción y no se permiten mas intentos mediante asignando a la variable de salida *completionStatus* el valor complete (completo).

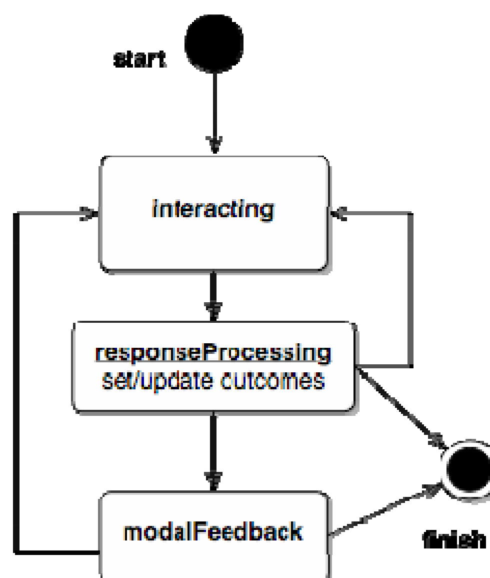


Figura 58 Diagrama de estados Feedback

11. Tecnologías utilizadas

11.1. Ant y Maven

11.1.1.- Introducción

Antes de comparar Ant con Maven, hay que decir que mientras que Ant es una herramienta de construcción, Maven es una herramienta de gestión de código fuente y que, por tanto, abarca una problemática mucho mayor. Es por eso que Ant, siendo una herramienta de construcción de proyectos, parece tan pobre al lado de Maven.

En general, compensa mucho más Maven que Ant, salvo en algún caso excepcional. Esto es debido a que Ant está mucho más centrado en la construcción del software, mientras que Maven está enfocado hacia el trabajo en equipo. Ant puede compensar en aquellos casos en los que se quiera tener control absoluto del proceso de construcción (y, aún así, se puede integrar una tarea Ant como si fuera un goal Maven).

Ant permite hilar mucho más fino a la hora de hacer el despliegue de la aplicación, pero a costa de ficheros de construcción muchísimo más grandes. Maven, por el contrario, simplifica enormemente el proceso de construcción y despliegue a cambio de seguir una estandarización en el modo de desarrollo. No hay nada que no se puede hacer con Maven que no se pueda hacer con Ant y viceversa, pero en el caso de Ant lleva mucho más trabajo por delante.

11.1.2.- Ant

11.1.2.1. Introducción

Ant es una herramienta de construcción de dominio público. Hoy por hoy es la herramienta J2EE de construcción más usada y cualquier IDE que se precie se integra nativamente con este programa. Las ventajas más importantes son que:

- es multiplataforma, ya que está escrito en Java,
- utiliza ficheros de construcción escritos en XML (*build.xml*), frente a la sintaxis críptica de *make*,
- puede extenderse fácilmente creando nuevas tareas a medida.

El hecho de ser multiplataforma hace que Ant añada una primera capa de abstracción que aísla al desarrollador respecto a la máquina en que esté trabajando. Un mismo proyecto Ant se comportará igual en el ordenador de un desarrollador que en el entorno de producción: no hay que cambiar las rutas de ficheros, ni depender de un entorno de desarrollo concreto para montar una aplicación, ni hacer quinientas cosas para realizar el proceso de construcción (se realiza todo con un solo clic...).

Esta independencia se ve reforzada por el hecho de que el fichero de configuración está escrito en XML, lo que facilita muchísimo su legibilidad y por tanto el mantenimiento y la extensibilidad de las tareas del proyecto.

11.1.2.2. El fichero de construcción en Ant

Ant busca por defecto el fichero *build.xml* en el directorio desde donde se ha invocado (puede cambiarse con la opción *–buildfile*). Este archivo contiene toda la información referente a las acciones que se realizan para la construcción de un proyecto: compilar, crear directorios, eliminarlos, hacer despliegues de aplicaciones, etc. Dichas acciones en el ámbito de Ant se conocen como tareas. Las tareas se agrupan en el fichero *build.xml* en bloques funcionales, llamados *targets*, que son los que se le pide a Ant que ejecute.

El fichero *build.xml* también indica las dependencias de unas acciones con otras, así como los pasos a seguir cuando se ejecuta una tarea. Cada fichero de construcción contiene un único proyecto, que se descompone en uno o más *targets* y éstas en cero o más tareas. Un ejemplo de dicho fichero puede ser éste:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<project basedir="." default="all" name="test">
        <property environment="env" />

        <!-- borra el directorio build -->
        <target name="clean">
                <delete dir="build"/>
        </target>

        <!-- crea los directorios build/classes y build/lib -->
        <target name="prepare">
                <mkdir dir="build/classes" />
                <mkdir dir="build/lib" />
        </target>

        <!-- define un grupo de archivos que pueden ser usados en el resto de tareas -->
        <path id="jars">
                <fileset dir="${env.CATALINA_HOME}/common/lib" includes="servlet.jar" />
        </path>

        <!-- compila las clases de la carpeta src y las deja en el directorio
build/classes;
        posteriormente monta un jar con todos los archivos .class que hay en
build/classes y
        lo deja en build/lib -->
        <target name="javac">
                <javac classpathref="jars" destdir="build/classes" srcdir="src"/>
                <jar destfile="build/lib/microf.jar" basedir="build/classes"
                includes="microf/**/*.class" />
        </target>

        <!-- ejecuta el resto de targets en el orden indicado -->
        <target depends="clean, prepare, javac" name="all">
        </target>
</project>
```

11.1.2.3. Extendiendo Ant

Para ello se creará una clase que extienda la clase *org.apache.tools.ant.Task*. La ejecución de la tarea se implementa mediante un método *execute* que lanza una excepción *BuildException* en caso de error.

Por cada atributo de la tarea se escribe un método *setter*. El tipo de atributo puede ser un tipo primitivo, *String*, *File*, *Class* o cualquier otro tipo que contenga un constructor con un único parámetro de tipo *String*. A continuación se muestra un ejemplo muy sencillo de una tarea Ant creada a medida:

```
import org.apache.tools.ant.Task;
public class MyEcho extends Task {
	private String message;
	public void setMessage(String m) {
		this.message= m;
	}
	public void execute() throws BuildException {
		log(message);
	}
}
```

Por último se añade al sistema la nueva tarea, que puede hacerse o bien añadiendo un elemento <*taskdef*> al proyecto, o bien de manera permanente, añadiendo la tarea al fichero *defaults.properties* en el paquete *org.apache.tools.ant.taskdefs*.

11.1.3.- Maven

11.1.3.1. Qué no es Maven

Características principales de Maven:
- Maven es una herramienta de *sites* y documentación.
- Maven extiende Ant para permitirte bajar dependencias.
- Maven son una serie de *scripts* reusables para gestionar dependencias.

Aunque Maven hace todas estas cosas, no son las únicas características que tiene y sus objetivos son bastante diferentes.

Maven promueve las *best practices* pero algunos proyectos no cuadran bien con estos ideales por razones históricas. Mientras que Maven está diseñado para ser flexible, hasta cierto punto, en estas situaciones y a las necesidades de distintos proyectos, no puede recoger cada situación sin comprometer la integridad de sus objetivos.

Si te decides a usar Maven y tienes una estructura de construcción inusual que no puedes reorganizar, tal vez tengas que olvidarte de algunas características de Maven o incluso del uso del mismo Maven.

11.1.3.2. De Ant a Maven

Ant añade sobre otras plataformas de construcción el hecho de ser multiplataforma y que su sintaxis está escrita en XML, siendo muy sencilla de entender (al contrario que por ejemplo en *make*). Permite hacer construcciones multi-plataforma de un modo estándar y elegante. Pero no todo es perfecto:
- Hay que encajar librerías externas al programa (dependencias), saber poner los distintos descriptores, *properties*, *jars*, etc. en el directorio adecuado... Esto se traduce en que el fichero *build.xml* empieza a crecer de manera

desproporcionada, con tareas que son un refrito de otras y que luego se modifican.

- También está el problema del *classpath*: diferentes versiones de diferentes librerías, en diferentes sitios, en diferentes estaciones de trabajo. Y después se oyen cosas del estilo de "¡pero si en mi ordenador compila!"

Maven adopta un punto de vista distinto, ya que fuerza a trabajar de tal modo que se desarrollen módulos (conocidos en el ámbito de Maven como *artifacts*: *WARs*, *JARs*, etc.) que dependan de versiones concretas de librerías, que a su vez se encuentran disponibles en un repositorio común; del mismo modo, estos *artifacts* se publican en el repositorio, de tal manera que estén disponibles para el resto de desarrolladores. Se acabó el problema del *classpath* y se acabó el problema del uso de versiones distintas de la misma librería.

El primer concepto básico de desarrollo en Maven es el proyecto. Un proyecto es cualquier directorio que contenga un archivo *project.xml*. Este archivo se conoce con el nombre de POM (*Project Object Model*), y contiene toda la información y estructura acerca del proyecto (nombre del proyecto, tipo, versión, autor, dependencias, etc.). Un fichero *project.xml* tiene una pinta parecida a esta:

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
        <pomVersion>3</pomVersion>
        <id>maven_dummy_project</id>
        <artifactId>maven_dummy_project_demo</artifactId>
        <name>maven dummy project</name>
        <groupId>metaware</groupId>

        <currentVersion>0.0.1</currentVersion>
        <organization>
                <name>Metaware Inc</name>
                <url>http://metaware-inc.wiki.mailxmail.com/</url>
                <logo>http://maven.apache.org/images/jakarta-logo-blue.gif</logo>
        </organization>

        <inceptionYear>2002</inceptionYear>
        <package>com.metaware.inc</package>
        <logo>http://maven.apache.org/images/maven.jpg</logo>

        <description>A collection of example projects showing how to use maven in
        different situations</description>
        <shortDescription>How to use maven in different situations</shortDescription>
        <url>http://maven.apache.org/reference/plugins/examples/</url>
        <issueTrackingUrl>http://nagoya.apache.org/scarab/servlet/scarab/</issueTrackingUrl>

        <siteAddress>jakarta.apache.org</siteAddress>
        <siteDirectory>/www/maven.apache.org/reference/plugins/examples/</siteDirectory>
        <distributionDirectory>/www/maven.apache.org/builds/</distributionDirectory>

        <repository>
                <connection>scm:cvs:pserver:anoncvs@cvs.apache.org:/home/cvspublic:
                mavenplugins/ examples</connection>
                <url>http://cvs.apache.org/viewcvs/maven-plugins/examples/</url>
        </repository>
        <mailingLists/>
        <developers/>
        <dependencies>
                <dependency>
                        <groupId>commons-beanutils</groupId>
                        <artifactId>commons-beanutils</artifactId>
                        <version>1.6.1</version>
                        <type>jar</type>
                        <properties>
                                <war.bundle>true</war.bundle>
                        </properties>
                </dependency>
```

```
	</dependencies>

	<build>
		<nagEmailAddress>turbine-maven-dev@jakarta.apache.org</nagEmailAddress>
		<sourceDirectory>src/java</sourceDirectory>

		<unitTestSourceDirectory>src/test</unitTestSourceDirectory>
		<unitTest>
			<includes>
				<include>**/*Test.java</include>
			</includes>
			<excludes>
				<exclude>**/NaughtyTest.java</exclude>
			</excludes>
		</unitTest>

		<resources>
			<resource>
				<directory>src/conf</directory>
				<includes>
					<include>*.properties</include>
				</includes>
				<filtering>false</filtering>
			</resource>
		</resources>

	</build>
</project>
```

Maven debe ser visto como un *framework*: es un entorno de trabajo para la gestión de proyectos y despliegue de aplicaciones. Consta de un núcleo que ejecuta distintas acciones o *goals*, el equivalente de los *targets* en Ant. Dichos *goals* se encuentran distribuidos en *plugins*. Por ejemplo, cuando ejecutamos *maven jar:compile*, estamos ejecutando el *goal compile* del *plugin jar*.

La versatilidad de Maven se encuentra en la cantidad ingente de *plugins* que tiene disponible y que permiten hacer casi cualquier cosa. Además de los *plugins* con los que Maven viene "de serie", encontramos gran cantidad de *plugins* de terceros y, si con eso no basta, siempre se puede desarrollar uno que realice las acciones que se necesiten en un momento dado.

El comportamiento de los *plugins* de Maven es personalizable: hay un fichero (opcional dentro de un proyecto Maven) que se usa para conseguir este objetivo, se trata del archivo *maven.xml*. En él se pueden definir los *goals* y añadirles *pre-goals* y *post-goals*. Un ejemplo de dicho fichero puede ser éste:

```
<project default="foobar-dist" xmlns:m="jelly:maven">
	<preGoal name="java:compile">
		<attainGoal name="xdoclet:webdoclet"/>
	</preGoal>

	<goal name="foobar-dist">
		<attainGoal name="war:install" />
	</goal>

	<postGoal name="war:war">
		<mkdir dir="${test.result.dir}"/>
		<echo>Creating directories</echo>
	</postGoal>
</project>
```

El último concepto clave es el de repositorio, que es dónde se guardan los *artifacts* de los cuales depende el proyecto. Este repositorio puede ser local o remoto. La idea es que a través del repositorio los desarrolladores compartan todas las librerías. Cada proyecto genera sus *artifacts* y los publica (instalar en Maven) en el repositorio remoto, al que accede el resto del equipo de desarrollo para recogerlos y poder usarlos de un modo estandarizado.

La estructura funcional de Maven puede representarse gráficamente de este modo:

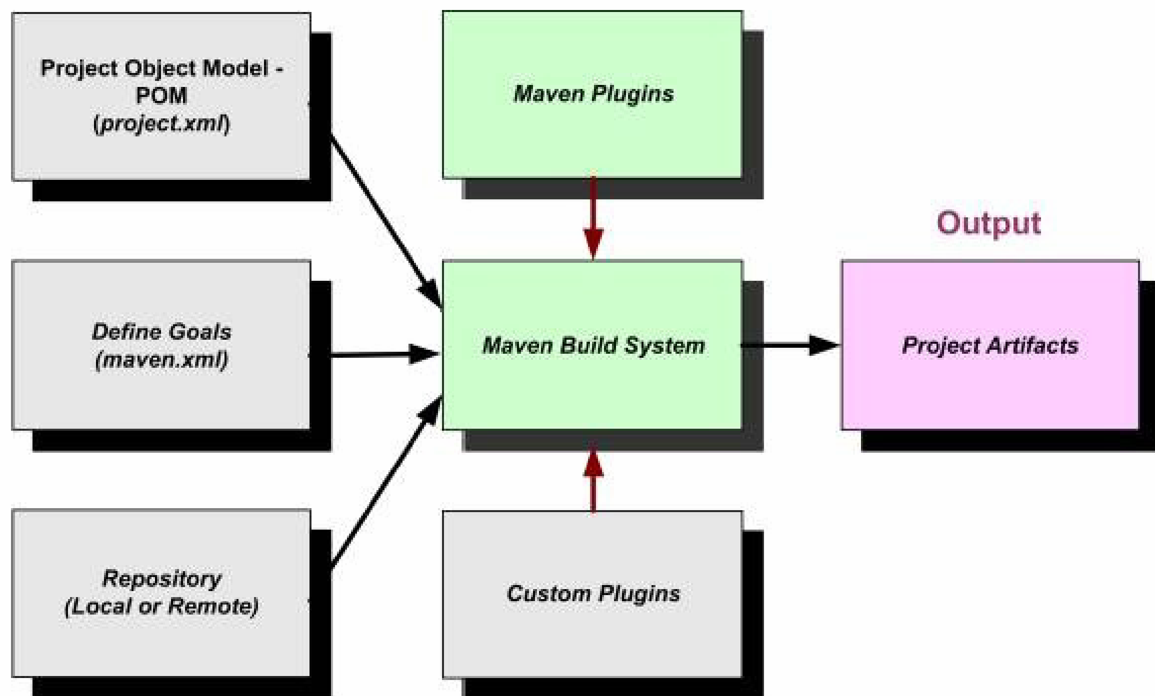


Figura 59 Estructura funcional de Maven

Los cuadros grises son dados por el desarrollador (son obligatorios el POM y el repositorio, los otros dos son opcionales), los verdes por Maven y el rosa es el resultado final del proyecto Maven, los *artifacts* generados, uno por proyecto Maven.

Maven por tanto está orientado al trabajo en equipo, consigue que:

- el problema de las distintas dependencias entre proyectos desaparezca
- un mismo proyecto funcione en distintos entornos de desarrollo y/o sistemas operativos sin tocar el código (como mucho ajustando propiedades en un fichero de configuración, y aún así es raro tener que tocar este fichero más de una vez para configurar todos los proyectos)
- el trabajo de un desarrollador se integre de modo transparente con el del resto
- se maximice la cohesión del código y se minimice su acoplamiento, debido al uso intensivo del repositorio Maven
- se facilite la reutilización del código

11.1.3.3. Maven y las metodologías ágiles

Los proyectos en Maven soportan herencia, de tal modo que es posible definir un proyecto "padre", que haga de plantilla del resto de subproyectos definidos dentro de éste. De esta manera, cada subproyecto hereda las propiedades del proyecto raíz. Siguiendo esta filosofía, en Maven está el concepto de Multi-Project y la herramienta *Maven Reactor*, que se encarga de, dado un conjunto de *project.xml*, calcular las dependencias entre ellos y de ejecutarlos en el orden correcto. Además también es posible parametrizar el *Reactor* vía el fichero *maven.xml*, con lo que se pueden hacer auténticas virguerías.

Y aún hay más sobre la potencia de Maven en la integración entre distintos proyectos, dado que se integra con herramientas de integración continua como *Cruise-Control* mediante el *plugin* adecuado. La integración continua es una práctica que forma parte de las llamadas metodologías ágiles. Propuesta por Martin Fowler, consiste en hacer integraciones (compilación y ejecución de tests) automáticas de un proyecto lo más a menudo posible, para así poder detectar los fallos cuanto antes.

El hecho de poder trabajar con varios proyectos independientes pero relacionados entre sí favorece enormemente la refactorización del código, así como poder orientar el desarrollo cara a poder cumplir las llamadas "*small releases*". Además, dentro del fichero *project.xml* se pueden definir dónde están las baterías de tests unitarios (ya sean a través de JUnit o de Cactus) que se ejecutarán nada más terminar de compilar el proyecto.

11.1.3.4. Extendiendo Maven 1.0

Maven 1.0.x se extiende a través de nuevos *plugins* que son escritos en Jelly, un lenguaje de *scripting* basado en xml. Con dicho lenguaje escribimos nuevos *goals* en el fichero *maven.xml*. Desde este fichero también se pueden invocar tanto tareas Ant como métodos de clases que hayamos escrito para tal efecto. Ejecutando el *goal Maven plugin:install* tendremos disponible el *plugin* para todos los proyectos. Ejecutando *maven site* se genera toda la documentación asociada al sitio Un ejemplo de un pequeño *plugin* Maven puede ser éste:

```
<project
	xmlns:j="jelly:core"
	xmlns:m="jelly:maven"
	xmlns:ant="jelly:ant"
	xmlns:util="jelly:util"
	xmlns:artifact="artifact">

	<goal name="war-distribution:pack-depedencies">
		<!--Remove previous libs-->
		<delete dir="${maven.build.dir}/dependencies/lib" />

		<!--Copy dependencies to bundle-->
		<j:forEach var="lib" items="${pom.artifacts}">
			<j:set var="dependency" value="${lib.dependency}"/>
			<j:if test="${dependency.getProperty('war.bundle') != null}">
				<ant:copy file="${lib.path}"
					toDir="${maven.build.dir}/dependencies/lib/${depend
					ency. getProperty('war.bundle')}" />
			</j:if>
		</j:forEach>
```

```
            <!--Build tar-->
            <tar destfile="${maven.build.dir}/${pom.artifactId}-
            ${pom.currentVersion}.tar"
                   basedir="${maven.build.dir}/dependencies" />

            <echo>Created tar holding dependencies</echo>
        </goal>
</project>
```

11.1.4.- Comparación de características de Ant y Maven

	Ant	***Maven***
Instalación	Muy fácil	Muy fácil
Tiempo para empezar un proyecto nuevo	5 minutos	1 minuto (mediante el *plugin genipa* en Maven 1.0 o mediante el *plugin archetype* en Maven 2.0)
Tiempo que se tarda en añadir una nueva funcionalidad	10 minutos para añadir un *target* nuevo	2 minutos en usar un *goal* nuevo
Tiempo de aprendizaje de la herramienta	30 minutos	2 horas
Estructura estándar en los proyectos	No	Sí
Soporte para Multi-Projects	Sí, pero es complicado de implementar	Sí, está orientado a ello
Generación de documentación	No, pero hay muchas herramientas disponibles para ello	Sí
Integración con IDEs	Sí, en casi todos, por no decir en todos (o por lo menos en cualquier IDE que se precie)	A través del *plugin Mevenide*, en Eclipse, Netbeans, JBuilder e IntelliJ IDEA

Tabla 11.1.1 Ant vs Maven

Como se comentó al principio del artículo, una comparación de Maven con Ant nunca será del todo acertada, ya que Maven abarca una funcionalidad mucho mayor que la que pretende Ant. Pero dependiendo del caso puede ser mejor utilizar uno u otro.

11.2. HSQLDB

11.2.1. Introducción

HipersonicSQLDB es un completísimo gestor de bases de datos relacionales, escrito 100% en puro Java y de código abierto(software libre -licencia BSD), pequeño, funcional y muy estable, por lo que es esta usando ya en muchos proyectos grandes. http://hsqldb.sourceforge.net/. Características:

- Completo gestor de bases de datos relacional.
- Escrito totalmente en Java, por lo que es portable.
- Rápido; tiempo de arranque mínimo y gran velocidad en las operaciones de selección(*SELECT*), inserción (*INSERT*), actualización (*UPDATE*) y borrado (*DELETE*).
- Sintaxis SQL estándar.
- Integridad referencial con claves foráneas: propiedad gracias a la cual se garantiza que una entidad, fila o registro, siempre se relaciona con otras entidades válidas, es decir, que existen en la base de datos.
- Se puede escribir operaciones en cascada y procedimientos almacenados(*stored procedures*) utilizando como lenguaje Java.
- Restricciones (*constraints*) y disparadores (*triggers*) que son eventos que se ejecutan cuando se cumple una condición establecida al realizar una operación de inserción (*INSERT*), actualización (*UPDATE*) o borrado (*DELETE*).
- Tablas en memoria o en disco de hasta 8GB.
- Tiene un controlador(*driver*) JDBC que incluso se puede usar de forma embebida dentro de la aplicación.
- Se puede operar de varios modos, que se pueden agrupar en En-Proceso (*In-process*) y Cliente-Servidor (*Client-Server*). En el primer caso solamente puede tener una conexión abierta dentro de la misma máquina virtual, y en el segundo, que a su vez tiene varias alternativas, sirve los datos a través de un puerto 9001 de forma predeterminada y admite n conexiones.
- Puede ser un servlet en un servidor tomcat,... o un applet en el navegador. Tiene un modo Servidor-Web en el cual puedes acceder a ella a través de los puertos http o https, y en la ultima versión incluso puede hacer transacciones de esta manera, de modo que puedes tener el servidor detrás de un cortafuego (*firewall*) y los programas clientes que hacen transacciones en Internet.
- También se compacta para optimizar su rendimiento lo que se puede hacer al cerrar la BD, utilizando "*Shutdown Compact*" (en vez del simple "*Shutdown*").
- En el mismo archivo .jar vienen algunas herramientas administrativas, una de las más útiles es el gestor de bases de datos hecha con AWT y otra hecha con Swing. Mediante la cual se pueden enviar sentencias SQL al motor para modificar la base de datos o recuperar registros.

11.2.2. Componentes del paquete jar HSQLBD

El paquete *Hsqldb.jar* contiene varios componentes y programas:

- *HSQLDB RDBMS*
- *HSQLDB JDBC Driver*
- *Database Manager*(versiones *Swing* y *Ant*)
- *Query Tool*(*AWT*)
- *SQL Tool*(línea de comandos)

El *HSQLDB RDBMS* y el *JDBC Driver* proporcionan la funcionalidad principal. El resto son herramientas de bases de datos de propósito general que pueden ser usadas con cualquier motor de base de datos que tenga un controlador *JDBC*. Diferentes comandos son usados para ejecutar cada programa.

11.2.2.1. Ejecutando herramientas

Todas las herramientas pueden ser ejecutadas de manera estándar para clases Java archivadas. En el siguiente ejemplo el *hsqldb.jar* de la versión AWT del gestor de base de datos(*Database Manager*) está localizado en el directorio ../lib relativo al directorio actual.

```
java –cp ../lib/hsqldb.jar org.hsqldb.util.DatabaseManager
```

Si el *hsqldb.jar* esta en el directorio actual, el comando debería cambiar a:

```
java –cp hsqldb.jar org.hsqldb.util.DatabaseManager
```

11.2.2.2. Clases principales para las herramientas Hsqldb

- *org.hsqldb.util.DatabaseManager*
- *org.hsqldb.util.DatabaseManagerSwing*
- *org.hsqldb.util.Transfer*
- *org.hsqldb.util.QueryTool*
- *org.hsqldb.util.SqlTool*

Algunas herramientas, tal como el Gestor de Bases de datos (*Database Manager*) o la herramienta SQL (*SQL Tool*), pueden usar argumentos en línea de comandos o únicamente contar con ellos. Puedes añadir el argumento –? en línea de comandos para obtener una lista de argumentos disponibles para esas herramientas. Características del gestor de base de datos (*Database Manager*) son una interfaz grafica de usuario y que puede ser explorado interactivamente.

11.2.3. Ejecutando HSQLDB

HSQLDB puede ser ejecutada de diferentes maneras. En general son divididas en modos servidor (*Server Modes*) y en modo en-proceso (*In-Process Mode*) también llamada modo autónomo (*Standalone Mode*). Un subprograma diferente desde el jar es usado para ejecutar HSQLDB en cada modo.

Cada base de datos HSQDB consta de entre dos a cinco archivos, todas llamados igual pero con diferentes extensiones, localizados en el mismo directorio. Por ejemplo, una base de datos llamada "*test*" consta de los siguientes archivos:

- *test.properties*
- *test.script*
- *test.log*
- *test.data*
- *test.backup*

El archivo propiedades (*properties*) contiene la configuración general de la base de datos. El archivo guión (*script*) contiene la definición de tablas y otros objetos de la base de datos, más los datos para tablas no cacheadas. El archivo registro (*log*) contiene cambios recientes de la base de datos. El archivo datos (*data*) contiene los datos para tablas cacheadas y el archivo copia de seguridad (*backup*) es una copia de seguridad comprimida del ultimo estado consistente conocido del archivo de datos. Todos estos archivos son esenciales y nunca deberían ser borrados. Si la base de datos tiene tablas no cacheadas, los archivos test.data y test.backup no estarán presentes. Además de estos archivos, la base de datos HSQLDB puede tener cualquier archivo de texto formateado, tal como listas CSV, en cualquier parte en el disco.

Mientras la base de datos "*test*" es operacional, un archivo *test.log* es usado para escribir los cambios hechos en los datos. Este archivo es eliminado en un cierre (*shutdown*) normal. De lo contrario, con un cierre anormal, este archivo es usado en el próximo arranque (*startup*) para rehacer los cambios. Un archivo *test.lck* es también usado para grabar el hecho que la base de datos es abierta. Este es eliminado en un cierre normal. En algunas circunstancias, un *test.data.old* es creado y eliminado después.

Cuando el motor cierra la base de datos en un cierre, ello crea archivos temporales con la extensión *.new* cualquiera de ellos luego son renombrados a los antes enumerados.

11.2.4. Modos Servidor (Server Modes)

Los modos servidor proporcionan la máxima accesibilidad. El motor de la base de datos se ejecuta en una *JVM* y están atentos a conexiones desde programas en el mismo ordenador u otros ordenadores en la red. Varios programas diferentes pueden conectarse al servidor y recuperar o actualizar información. Los programas clientes conectados al servidor usan el controlador *HSQLDB JDBC*. En muchos modos servidor, el servidor puede servir a diez bases de datos que son especificadas en el tiempo de ejecución del servidor.

Hay tres modos servidor, basados en el protocolo usado para comunicaciones entre cliente y servidor.

11.2.4.1. Servidor HSQLDB (HSQLDB Server)

Este es el modo preferido para ejecutar un servidor de base de datos y uno de los más rápidos. Un protocolo de comunicaciones propietario es usado para este modo. Un comando similar a estos es usado para ejecutar herramientas y lo arriba descrito es usado para ejecutar el servidor. El siguiente ejemplo del comando para arrancar el servidor comienza el servidor con una base de datos por defecto con los archivos llamados "*mydb.**".

```
java -cp ../lib/hsqldb.jar org.hsqldb.Server -database.0 mydb -dbname.0 xdb
```

El argumento en línea de comandos -? puede ser usado para obtener una lista de los argumentos disponibles.

11.2.4.2. Servidor Web HSQLDB (HSQLDB Web Server)

Este modo es usado cuando el acceso al ordenador anfitrión(*hosting*) del servidor de la base de datos esta restringido al protocolo http. La única razón para usar el modo Servidor Web(*Web Server*) es imponer restricciones por cortafuegos(*firewalls*) en las maquinas cliente o servidor y no debería ser usado donde no hay restricciones. El Servidor Web(*Web Server*) HSQLDB es un servidor Web especial que permite a los clientes JDBC conectarse vía http. Desde la versión 1.7.2 este modo también permite transacciones.

Para ejecutar un servidor Web, reemplazamos la clase principal del servidor en la línea de comandos del ejemplo anterior con lo siguiente:

```
org.hsqldb.WebServer
```

El argumento en línea de comandos -? puede ser usado para obtener una lista de los argumentos disponibles.

11.2.4.3. HSQLDB Servlet

Este usa el mismo protocolo que el Servidor Web (*Web Server*). Se usa cuando un motor servlet separado, o servidor de aplicación, tal como Tomcat o Resin proporciona acceso a la base de datos. El modo servlet (*Servlet Mode*) no puede ser arrancado independientemente desde el motor servlet. La clase hsqlServlet, en el jar HSQLDB, debería ser instalada en el servidor aplicación para proporcionar la conexión. La base de datos se especifica usando un servidor aplicación propietario.

Ambos modos servidor Web (*Web Server*) y servidor Servlet (*Servlet Server*) solamente pueden ser accedidos usando el controlador JDBC en el cliente final. Ellos no proporcionan una interfaz a la base de datos. El modo Servlet puede servir solamente una única base de datos.

Hay que tener en cuenta que normalmente no se usa este modo si estas usando el motor de la base de datos en un servidor de aplicación.

Conectando a una base de datos ejecutándose como un servidor

Una vez que el servidor HSQLDB esta ejecutándose, los programas clientes pueden conectarse a él usando el controlador HSQLDB JDBC (*HSQLDB JDBC Driver*) contenido en el hsqldb.jar. Toda la información de cómo conectar a un servidor esta suministrada en la documentación de Java para jdbcConnection [../src/org/hsqldb/jdbcConnection.html]. Un ejemplo común es la conexión al puerto por defecto (9001) usada por el protocolo hsql en la misma maquina:

```
try {
Class.forName("org.hsqldb.jdbcDriver" );
} catch (Exception e) {
System.out.println("ERROR: failed to load HSQLDB JDBC driver.");
e.printStackTrace();
return;
}
Connection c = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb",
```

En algunas circunstancias, puedes usar la siguiente línea para obtener el controlador.

```
Class.forName("org.hsqldb.jdbcDriver").newInstance();
```

Nota que en la anterior conexión URL, no hay mención al archivo de la base de datos, esto fue así especificado cuando ejecutamos el servidor. En su lugar, usamos el valor definido por dbname.0.

Consideraciones de seguridad

Cuando HSQLDB esta corriendo como un servidor, el acceso a la red debería ser protegido adecuadamente. Las direcciones IP originales-fuentes deberían ser restringidas mediante el uso del filtrado TCP, programas cortafuegos (*firewall*), o cortafuegos independientes. Si el trafico atravesara una red desprotegida, tal como Internet, la corriente debería ser encriptada; por ejemplo con VPN, haciendo un túnel ssh, o TLS usando el HSQLDB permitido y variantes https del servidor y modos servidor Web. Solamente contraseñas (*passwords*) seguras deberían ser usadas—lo más importante, la contraseña por defecto del usuario del sistema debería ser cambiada de la cadena vacía por defecto. Si decididamente estas proporcionando datos al público, entonces la conexión a la red pública ancho-abierto debería ser usada exclusivamente para acceder a los datos públicos vía cuentas de solo lectura. (Es decir, tampoco datos seguros ni cuentas privilegiadas deberían usar esta conexión). Estas consideraciones también se aplican a los servidores HSQLDB ejecutándose con el protocolo http.

11.2.4.4. Modo En-proceso, Autónomo(In-Process, Standalone Mode)

Este modo ejecuta el motor de la base de datos como parte de tu programa aplicación en la misma JVM. Para la mayoría de las aplicaciones este modo puede ser más rápido, ya que los datos no son transformados y enviados por la red. El principal inconveniente es que por defecto no es posible conectarse a la base de datos desde fuera de tu aplicación. Como resultado no puedes chequear los contenidos de la base de datos con herramientas externas tal como gestor de base de datos (*Database Manager*) mientras tu aplicación se está ejecutando. En 1.8.0, tu puedes ejecutar una instancia del servidor en un hilo (*thread*) desde la misma maquina virtual como desde tu aplicación y proporcionar accesos externos a tu base de datos en-proceso.

La forma recomendada de usar el modo en-proceso en una aplicación es usar una instancia del servidor HSQLDB (*HSQLDB Server*) para la base de datos mientas desarrollamos la aplicación y entonces cambiar al modo en-proceso (*In-Process*) para desarrollar.

Una base de datos en modo en-proceso (In-Process Mode) es arrancada desde JDBC, con la ruta del archivo de la base de datos especificada en la conexión URL. Por ejemplo, si el nombre de la base de datos es testdb y ese archivo esta localizado en el mismo directorio donde fue hecho el comando para ejecutar tu aplicación, así el siguiente código se usa para la conexión:

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:file:testdb", "sa", "");
```

El formato de la ruta del archivo de la base de datos puede ser especificado usando hacia cortada en Windows hosts tanto como en Linux hosts. Así las rutas relativas o rutas que referencian al mismo directorio en el mismo conducto puede ser idéntica. Por ejemplo si la ruta de tu base de datos en Linux es /opt/db/testad y tu creas una estructura de directorio idéntico en C:conducto de un host Windows, tu puedes usar la misma URL en ambos Windows y Linux:

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:file:/opt/db/testdb", "
```

Cuando usamos rutas relativas, estas rutas serán puestas relativas al directorio en el cual el comando shell arranca la JVM que fue ejecutada.

11.2.4.5. Bases de datos de solo memoria(Memory-Only Databases)

Es posible ejecutar HSQLDB de manera que la base de datos no sea persistente y exista completamente en memoria de acceso aleatorio. Así la información no es escrita en disco, este modo debería ser usado solamente para procesamientos internos de los datos de la aplicación, en *applet* o en ciertas aplicaciones especiales. Este modo es especificado en el protocolo mem:

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:mem:aname", "sa", "");
```

También puedes ejecutar una instancia de un servidor solo en memoria especificando la misma URL en el server.properties. Este uso no es común y es limitado para aplicaciones especiales donde el servidor de la base de datos se usa solamente para intercambiar información entre clientes, o para datos no persistentes.

11.2.5. Características generales de la Base de Datos

11.2.5.1. Cerrar la base de datos

Todas las bases de datos ejecutándose en diferentes modos pueden ser cerradas con el comando *SHUTDOWN*, distribuidas como una consulta SQL. Para la versión 1.7.2, las bases de datos en-proceso no son mas largas de cerrar cuando la ultima conexión a la base de datos es específicamente cerrada vía JDBC, un *SHUTDOWN* es requerido. En la 1.8.0, una propiedad de conexión *SHUTDOWN=TRUE*, puede ser especificada en la primera conexión de la base de datos, la conexión que abre la base de datos, para forzar un *SHUTDOWN* cuando la ultima conexión cierra.

Cuando se hace *SHUTDOWN*, todas las transacciones activas son reducidas. Una forma especial de cerrar la base de datos es mediante el comando *SHUTDOWN COMPACT*. Este comando rescribe el archivo.data que contiene la información almacenada en las tablas cacheadas (*cached tables*) y compacta su tamaño. Este comando debería ser realizado periódicamente, especialmente cuando muchos inserts, updates o deletes han sido tocados en las tablas cacheadas (*cached tables*). Cambios en la estructura de la base de datos, tal como caídas o modificaciones tablas cacheadas pobladas (*cached tables*) o indexadas también crea grandes cantidades de filas de espacio no usadas que pueden ser reclamadas usando esta comando.

11.2.5.2. Usando múltiples bases de datos en una JVM.

En los ejemplos anteriores cada servidor sirve solamente una base de datos y solamente puede ser creada una base de datos en memoria. Sin embargo, desde la versión 1.7.2, HSQLDB puede servir varias bases de datos en múltiples modos servidores y permitir accesos simultáneos a múltiples bases de datos en-proceso y solo en memoria.

11.2.5.3. Crear una nueva base de datos.

Cuando una instancia de un servidor es arrancada, o cuando una conexión es hecha en una base de datos in-process, una nueva, base de datos vacía se crea si la base de datos no existe en la ruta dada.

Esta característica tiene un efecto lateral que puede confundir a nuevos usuarios. Si se produce un error especificando la ruta para la conexión para una base de datos existente, una conexión es establecida para una nueva base de datos. Para propósitos mediadores, tu puedes especificar una propiedad de conexión IFEXISTS=TRUE permite conexión a una base de datos existente solamente y permitir crear una nueva base de datos. En este caso, si la base de datos no existe, el método *getConnection()* lanzara una excepción.

11.2.6. Usar el motor de bases de datos

Una vez que una conexión esta establecida a una base de datos en un determinado modo, los métodos JDBC son usados para interactuar con la base de datos.
El Javadoc para jdbcConnection [../src/org/hsqldb/jdbc/jdbcConnection.html], jdbcDriver [../src/org/hsqldb/jdbcDriver.html], jdbcDatabaseMetadata [../src/org/hsqldb/jdbc/jdbcDatabaseMetaData.html], jdbcResultSet [../src/org/hsqldb/jdbc/jdbcResultSet.html], jdbcStatement [../src/org/hsqldb/jdbc/jdbcStatement.html], y jdbcPreparedStatement [../src/org/hsqldb/jdbc/jdbcPreparedStatement.html] enumera todos los métodos JDBC suportados junto con la información que es específica para HSQLDB.
Los métodos JDBC son generalmente divididos en: métodos relacionados con conexión, métodos meta-datos y métodos de acceso a la base de datos. Los métodos de acceso a la base de datos usan comandos SQL para hacer acciones en la base de datos y devolver los resultados también como un tipo primitivo Java o como una instancia de la clase java.sql.ResultSet.

Puedes usar el gestor de base de datos (*Database Manager*) u otra herramienta de acceso a la base de datos java para explorar tu base de datos y actualizarla con comandos SQL. Estos programas usan JDBC internamente para presentar tus comandos al motor de la base de datos y mostrar el resultado en un formato legible.

El dialecto SQL usado en HSQLDB es cercano a los estándares SQL92 y Sql200n así ha sido posible lograr extender tan lejos un motor de base de datos de pequeña-huella.

11.2.6.1. Diferentes tipos de tablas

HSQLDB soporta tablas temporales (*Temp tables*) y tres tipos de tablas persistentes.

Las tablas temporales (*Temp tables*) no son escritas en disco y duran solamente el tiempo de vida del objeto de conexión (*Connection object*). El contenido de cada tabla temporal (*Temp table*) es solamente visible desde la conexión (*Connection*) que fue usada para poblarla, otras conexiones concurrentes a la base de datos tendrá acceso a sus propias copias de la tabla. Desde la versión 1.8.0 la definición de tablas temporales (*Temp tables*) se ajusta al tipo global temporal (*Global Temporary*) en el estándar SQL. La definición de la tabla persiste pero cada conexión nueva tiene su propia copia de la tabla, la cual esta vacía al principio. Cuando la conexión hace commit, los contenidos de la tabla son limpiados por defecto. Si la sentencia de definición de la tabla incluye *ON COMMIT PRESERVE ROWS*, entonces los contenidos son guardados cuando un commit tenga lugar.

Los tres tipos de tablas persistentes con tablas en memoria (*Memory tables*), tablas cacheadas (*Cached tables*) y tablas de texto (*Text tables*).

Las tablas de memoria (*Memory tables*) son el tipo por defecto cuando el comando *CREATE TABLE* es usado. Sus datos son enteramente sostenidos en memoria pero cualquier cambio a su estructura o contenidos es escrito al archivo <dbname>.script. El archivo script es leído la próxima vez que la base de datos es abierta, y las tablas de

memoria (*Memory tables*) son recreadas con todos sus contenidos. Esto es diferente de las tablas temporales (*Temp tables*), por defecto, las tablas de memoria (*Memory tables*) son persistentes.

Las tablas cacheadas (*Cached tables*) son creadas con el comando *CREATE CACHED TABLE*. Solamente parte de sus datos o índices es guardado en memoria, permitiendo grandes tablas que sino deberían ocupar varios cientos de megabytes de memoria. Otra ventaja de las tablas cacheadas es que el motor de la base de datos tarda menos tiempo en arrancar cuando la tabla cacheada esta usando grandes cantidades de datos. El inconveniente de las tablas cacheadas es una reducción de velocidad. No se deberían usar tablas cacheadas si tu conjunto de datos es relativamente pequeño. En una aplicación con alguna tabla pequeña y alguna otra grande, lo mejor es usar el por defecto, modo memoria (*Memory tables*) para tablas pequeñas.

Las tablas de texto (*Text tables*) son soportadas desde la versión 1.7.0 usando un CSV (*Comma Separated Value*) u otro archivo de texto delimitado como el código de sus datos. Tu puedes especificar un archivo CSV existente, como un deposito desde otra base de datos o programa, así el código de una tabla de texto (*Text table*). Alternativamente, tu puedes especificar un archivo vació para ser llenado con datos por el motor de la base de datos. Las tablas de texto (*Text tables*) son eficientes en uso de memoria así ellas cache solamente parte de los datos de texto y todos los índices. El código de los datos de la tabla de texto (*Text table*) puede ser siempre reasignado a un archivo diferente si es necesario.

Con bases de datos de solo memoria, ambas declaraciones tabla de memoria (*Memory table*) y tabla cacheada (*Cached table*) son tratadas como declaraciones para tablas no persistentes en memoria. Las declaraciones de tablas de texto (*Text tables*) no son permitidas en este texto.

11.2.6.2. Restricciones e índices

HSQLDB soporta restricciones *PRIMARY KEY*, *NOT NULL*, *UNIQUE*, *CHECK* y restricciones *FOREIGN KEY*. Además soporta *UNIQUE* o índices ordinarios. Este soporte es justamente comprensible y cubre restricciones e índices multi-columna, mas actualizaciones en cascada y eliminaciones para claves ajenas.

HSQLDB crea índices internamente para soportar restricciones *PRIMARY KEY*, *UNIQUE* y *FOREIGN KEY*: un único índice es creado para cada restricción *PRIMARY KEY* o *UNIQUE*, y índices ordinarios son creados para cada restricción *FOREIGN KEY*. El porque de esto es que tu no deberías crear índices duplicados definidos de usuario en el mismo conjunto de columnas cubiertas por esas restricciones. Esto debería resultar en memoria innecesaria y desbordamientos rápidos.

Los índices son cruciales para una velocidad de consulta adecuada. Cuando las consultas juntan múltiples tablas usadas, debería haber un índice en cada columna juntada de cada tabla. Cuando el rango de condiciones iguales son usadas por ejemplo *SELECT... WHERE* acol >10 *AND* bcol = 0, un índice es requerido en la columna usada en la condición. Índices no tienen efecto en las cláusulas *ORDER BY* o en algunas condiciones *LIKE*.

Como una regla, HSQLDB es capaz del procesamiento interno de consultas de unas 100000 filas por segundo. Cualquier consulta que ejecuta en varios segundos debería ser chequeada e índices deberían ser añadidos a la pertinente columna de la tabla si es necesario.

11.2.6.3. Soporta SQL

La sintaxis SQL es soportada por HSQLDB es esencialmente lo que es especificado en el estándar SQL (*SQL Standard*) 92 y 200n. No todas las características del estándar son soportadas y hay alguna extensión propietaria. En la versión 1.8.0 la conducta del motor esta mas lejos de amoldarse con el estándar que otras versiones más antiguas. Los cambios principales son:

- Tratamiento correcto de valores en unión *NULL* en columna, en restricción *UNIQUE* y en consulta de condiciones
- Procesamiento correcto de selects con *JOIN* y *LEFT OUTER JOIN*
- Procesamiento correcto de funciones agregadas en expresiones o argumentos contenidos en expresiones.

11.2.6.4. Soporta JDBC

Desde la versión 1.7.2, el sopote para JDBC2 ha sido significativamente extendido y algunas características de JDBC3 son también soportadas.

11.3. Hibernate

11.3.1. Mapeador Objeto-Relacional

La programación orientada a objetos y la base de datos relacional corresponde a dos paradigmas muy diferentes. La programación orientada a objetos trata de objetos, sus atributos y asociaciones unos con otros, mientras que la base de datos corresponde a modelos relacionales.

Cuando se desarrolla una aplicación orientada a objetos con base de datos relacional, la manera tradicional de acceder sería a través de una conexión JDBC directa a la base de datos mediante la ejecución de sentencias SQL. Esto sería útil y efectivo cuando no haya gran número de clases de negocio, ya que el mantenimiento del código esta muy relacionado al modelo de datos relacional. Un mínimo cambio en el modelo de datos implica revisión y posibles modificaciones en el código de la aplicación.

Una implementación más avanzada sería la utilización de unas clases de acceso de datos (DAO). Nuestra capa de negocio se comunicaría con la capa DAO y esta realizaría las operaciones sobre la base de datos. Pero sigue habiendo problemas de mantenimiento y portabilidad.

Un Mapeador Objeto-Relacional es una capa de persistencia que separa el código de la aplicación de la realización de nuestras sentencias SQL contra la Base de Datos. Sus funciones van desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.

11.3.2. Características

Hibernate es la herramienta de mapeo objeto-relacional de código abierto más avanzado y maduro que hay actualmente. Te permite diseñar objetos persistentes de una manera muy rápida y optimizada.

La representación de alto nivel para la arquitectura de Hibernate es mostrada en la figura 60. Hibernate es la capa intermediaria entre la aplicación y la Base de Datos y así proporcionar a la aplicación servicios (y objetos) persistentes.

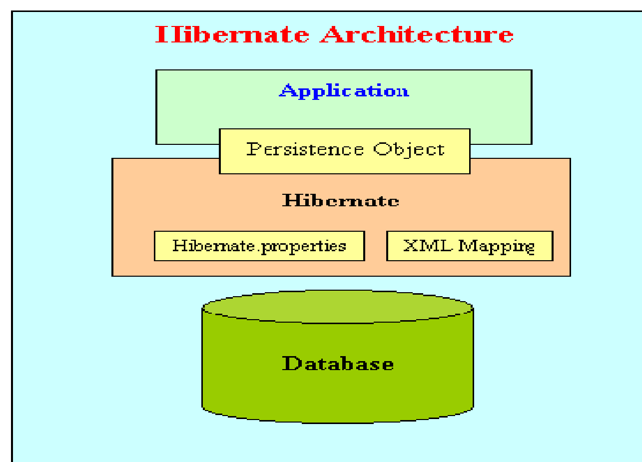


Figura 60 Arquitectura de Hibernate

La arquitectura de Hibernate posee tres grandes componentes:

- Gestión de la Conexión: proporciona una gestión eficiente de las conexiones a la base de datos. La aplicación se comunicará con Hibernate únicamente mediante un objeto llamado Sesión e Hibernate se encargará de crear y comunicarse directamente con la base de datos.

- Gestión de Transacciones: permite que el usuario ejecute una o más transacciones en el mismo tiempo.

- Mapeo Objeto-Relacional: transforma o mapea la representación de los datos de un modelo de objetos a un modelo de datos relacional. Este componente de Hibernate consulta, inserta, actualiza y elimina los registros de las tablas de la BBDD. Cuando la aplicación llama al método Session.save() con el objeto deseado, Hibernate lee el estado de los atributos del objeto y ejecuta las instrucciones o consultas necesarias.

Hibernate soporta la mayoría de los sistemas de bases de datos SQL. El *Hibernate Query Language*, diseñado como una extensión mínima, orientada a objetos, de SQL, proporciona un puente elegante entre los mundos objeto y relacional. Hibernate ofrece facilidades para recuperación y actualización de datos, control de transacciones, repositorios de conexiones a bases de datos, consultas programáticas y declarativas, y un control de relaciones de entidades declarativas.

11.3.2.1. Utilidades de Hibernate

Hibernate mapea objetos java a tablas de la Base de Datos, permitiendo de una manera sencilla la persistencia, actualización, consultas y eliminación de los datos almacenados en la BBDD. La manera de cómo mapear los objetos java es definida en un archivo xml, que ha de ser validado con la DTD de Hibernate.
Un ejemplo de mapeo es el siguiente:

Definimos una clase sencilla Persona, con sus atributos y métodos correspondientes.

```
Public class Persona{
       private String nombre;
       private String apellidos;
       private String email;
       private long id;

       public void setNombre(String n){nombre = n;}
       public void setApellidos(String a){apellidos = a ;}
       public void setEmail(String e){email = e;}
       private void setId(Long i){id = i; }
       public String getNombre(){return nombre;}
       public String getApellidos(){return apellidos;}
       public String getEmail(){return email;}
       public Long getId(){return idl;}
}
```

A continuación construimos el archivo xml donde es definido el mapeo de la clase Persona a una tabla PERSONA de la BBDD.

```
<hibernate-mapping>
<class name="Persona" table="PERSONA">
       <id name="id" type="long" column="ID">
       <generator class="assigned"/>
       </id>
       <property name="nombre">
       <column name="NOMBRE">
       </property>
       <property name="apellidos">
       <column name="APELLIDOS">
       </property>
       <property name="email">
       <column name="EMAIL">
       </property>
</class>
</hibernate-mapping>
```

Hibernate a partir de este archivo construye una tabla en la Base de Datos, asignando por cada atributo de la clase Persona una columna diferente. El atributo Id asignará un identificador único con respecto a los otros objetos persistentes. Este atributo corresponderá con la clave principal de la tabla en la BBDD. Utilizando identificadores de objetos tanto a nivel de código como en BBDD simplificamos mucho la complejidad de nuestra aplicación y podemos programar partes de la misma como código genérico.

Para construir las relaciones de la base de datos también se le ha de indicar a Hibernate mediante un archivo xml. En este archivo ha de indicarse que tipo de relación se trata (*one-to-one, one-to-many, many-to-many*), las tablas correspondientes y así como también las claves por las que se relacionaran las tablas.

Una vez creada estas tablas y sus relaciones, si deseamos almacenar, recuperar, actualizar información de esta BBDD, el desarrollador ha de interactuar con el motor de Hibernate mediante un objeto especial llamado Sesión (clase *Session*). Este objeto ofrece diversos métodos para que el desarrollador pueda comunicarse con la base de datos. Por ejemplo:

- *save(Objet object)*: realiza una actualización del objeto pasado como parámetro.
- *createQuery(String queryString)*: realiza una consulta.
- *beginTransaction()*: comienza una transacción sobre la BBDD.
- *close()*: cierra el objeto Session.

Hibernate puede soportar tres lenguajes de consultas diferentes para realizar la comunicación desde la aplicación a la base de datos relacional. Estos son:

- *Hibernate Query Language* (HQL): es una extensión orientada a objetos de SQL. Permite acceder a la información de la BBDD de diferentes maneras incluyendo queries orientadas a objeto. Ejemplo:

```
List users = session.find("from Person as p
        where p.fullName = ?", "Ana Sanz", Hibernate.STRING );
```

- *Hibernate Criteria Query API*: construye consultas dinámicas. Ejemplo:

```
Criteria criteria = session.createCriteria(Person.class);
criteria.add (Expression.eq ("fullName", "Ana Sanz"));
criteria.setMaxResults(20);
List users = criteria.list();
```

- *Native SQL*: soporte para consultas creadas en SQL. Ejemplo:

```
List users = session.createSQLQuery("SELECT {user.*} FROM USERS AS
        {user}", "user", UserInfo.class).list();
```

11.3.2.2. XDoclet

XDoclet es un motor de generación de código dirigido mediante atributos. No esta solamente limitado a Hibernate, puede generar toda clase de ficheros descriptores basados en XML, tales como EJB o descriptores de desarrollo de servicio Web. En el caso de Hibernate nos permite crear automáticamente los ficheros xml de mapeo mediante la lectura de meta-atributos y tags escritos en Java.

A continuación se explicará los tags más importantes de XDoclet utilizados para la generación de los archivos de mapeo de Hibernate. En el siguiente ejemplo se muestra la clase Usuario y sus atributos que serán mapeados y así obtener la persistencia de los datos deseada.

La clase Usuario es una entidad que posee un identificador, atributos y asociaciones a otras entidades. Primero declaremos el mapeo para la clase Usuario:

```
/**
* @hibernate.class
* table="USUARIOS"
*/
public class Usuario implements Serializable {
...
```

Los tags de XDoclet para Hibernate siempre tiene la sintaxis @hibernate.tagname (opcional) atributos. El tagname esta relacionado a un elemento dentro de las declaraciones de mapeo XML; en el anterior ejemplo, hibernate.class hace referencia al elemento de mapeo <class> del fichero XML. El atributo table es colocado a USUARIOS. Un extracto del fichero de mapeo generado por este tag es el siguiente:

```
<hibernate-mapping>
<class
name="Usuario"
table="USUARIOS">
...
```

Los usuarios son entidades por lo que se necesita un identificador. En el código fuente de la clase todas las propiedades (valores con tipos o atributos asociados a otras entidades) son marcadas por tags XDoclet en los métodos accesores. Para la propiedad id (identificador) añadimos el siguiente tag al método getId():

```
/**
* @hibernate.id
* column="USER_ID"
* unsaved-value="null"
* generator-class="native"
*/
public Long getId() {
return id;
}
```

Los atributos del tag hibernate.id son los mismos atributos para el elemento <id>.
A continuación mostramos los tags para mapear una simple propiedad de la clase, el nombre del usuario:

```
/**
* @hibernate.property
* column="NOMBRE"
* length="16"
* not-null="true"
* unique="true"
* update="false"
*/
public String getNombre() {
return nombre;
}
```

El tag hibernate.property tiene todos los atributos del elemento <property>. Se puede no utilizar este patrón ya que se puede tomar en cuenta los valores por defecto de Hibernate: Si se añade el tag @hibernate.property al método accesor sin ningún atributo, el mapeo sería <property name=”nombre”/>, también se puede utilizar los valores por defecto para todos los otros posibles atributos. Esta técnica permite una mayor facilidad y rapidez de dominio sobre XDoclet.

Otro componente de la clase Usuario es la Dirección:

```
/**
* @hibernate.component
*/
public Direccion getDireccion() {
return direccion;
}
```

Esta vez los valores por defecto de Hibernate son utilizados para la declaración hiberante.component. Además de esta declaración de mapeo de un componente, las propiedades individuales de Dirección son también mapeadas. En el código fuente de Dirección, se añade los tags hibernate.property a los métodos getCalle(), getCodigoPostal(), getCiudad(). La clase Direccion no es mapeada, es decir no se colocan los tags clase para obtener una nueva tabla, solamente es un componente de Usuario y posiblemente de otras entidades y por lo tanto tampoco tiene un identificador asociado. Únicamente los métodos accesores de las propiedades de los componentes serán marcados mediante tags.

A continuación, terminamos la declaración de mapeo de Usuario con tags para mapear las asociaciones entre distintas entidades.

Mapeo de asociaciones entre entidades

El mapeo de asociaciones entre distintas entidades con XDoclet es básicamente la misma que se utiliza para las propiedades con valores con tipo. Se añaden los tags XDoclet a los métodos accesores. Por ejemplo, la asociación entre Usuario a ítem es la siguiente:

```
/**
* @hibernate.set
* inverse="true"
* lazy="true"
* cascade="save-update"
* @hibernate.collection-key
* column="SELLER_ID"
* @hibernate.collection-one-to-many
* class="Item"
*/
public Set getItems() {
return items;
}
```

Lo primero que se diferencia de la propiedad con valores tipados es el número de tags que se necesita para el mapeo. Se mapea el lado “many” de una asociación one-to-many; por lo tanto el tipo es de clase colección. Los atributos para hibernate.set son los mismos de siempre: inversa para el aspecto bi-direccional y por supuesto carga perezosa. Los otros dos tags están también relacionados a los elementos conocidos XML de Hibernate, <key> y <one-to-many>. Se nombra la columna de la foreign key en la tabla de ítem como SELLER_ID (USUARIO_ID podría ser mas obvio pero menos expresivo) y que se ha de nombrar explícitamente la clase de las entidades referenciadas por el objeto Set.

También se ha de mapear el otro lado de esta asociación. En la clase ítem, se mapea el seller:

```
/**
* @hibernate.many-to-one
* column="SELLER_ID"
* cascade="none"
* not-null="true"
*/

public Usuario getSeller() {
return seller;
}
```

Para el lado "one" de la asociación se puede omitir la clase de la entidad referenciada, esta implícito por el tipo de la propiedad. Ahora tenemos los dos lados de la asociación mapeados y podemos por tanto generar automáticamente los ficheros de mapeo XML.

11.4. Spring

11.4.1. Un framework de aplicación

Un *framework* es un término utilizado en computación en general para referirse a un conjunto de bibliotecas, en este caso clases, utilizadas para implementar la estructura estándar de una aplicación. Todo esto se realiza con el propósito de promover la reutilización de código y ahorrarle así trabajo al desarrollador al no tener que rescribir ese código para cada nueva aplicación que desee desarrollar.

Spring es un *framework* indicado para desarrollar aplicaciones escritas en Java, es decir, para el desarrollo de aplicaciones J2EE *(Java 2 Enterprise Edition*), incluyendo EJB (*Enterprise JavaBeans*), Servlets y JSP (*JavaServer Pages*). Spring logra combinar dichas herramientas y otras más en un sólo paquete, proporcionando una estructura más sólida y un mejor soporte para este tipo de aplicaciones. Spring no intenta inventar nada sino integrar las diferentes tecnologías existentes en un único *framework* para el desarrollo más sencillo y eficaz de aplicaciones J2EE portables entre servidores de aplicación.

Otro de los principales enfoques de Spring, por el cual esta ganando dicha popularidad, es que simplifica el desarrollo de aplicaciones J2EE al intentar evitar el uso de EJB. Spring intenta brindar los mismos servicios pero simplificando el modelo de programación, así la complejidad de la aplicación es proporcional a la complejidad del problema que se está resolviendo.

Spring fue creado con las siguientes metas:
- El buen diseño es más importante que la tecnología subyacente.
- El código debe ser fácil de probar.

Además se considera a Spring un *framework lightweight*, es decir, ligero, ya que es una aplicación que no requiere muchos recursos para su ejecución. Además, el *framework* completo puede ser distribuido en un archivo .jar de alrededor de 1 MB, lo cual representa muy poco espacio para la cantidad de servicios que ofrece.

Uno de los motivos de su auge y popularidad es su condición de aplicación *OpenSource*. Esto implica que el uso de Spring no tiene ningún coste ni necesidad de licencia, fomentando que muchas empresas y desarrolladores empiecen a utilizarlo en sus aplicaciones.

11.4.2. Arquitectura de Spring

Spring es un *framework* modular que cuenta con una arquitectura divida en siete capas o módulos, como se muestra en la Figura 61, lo que nos permite tomar y ocupar únicamente las partes que nos interesen para nuestro proyecto e integrarlas con gran libertad.

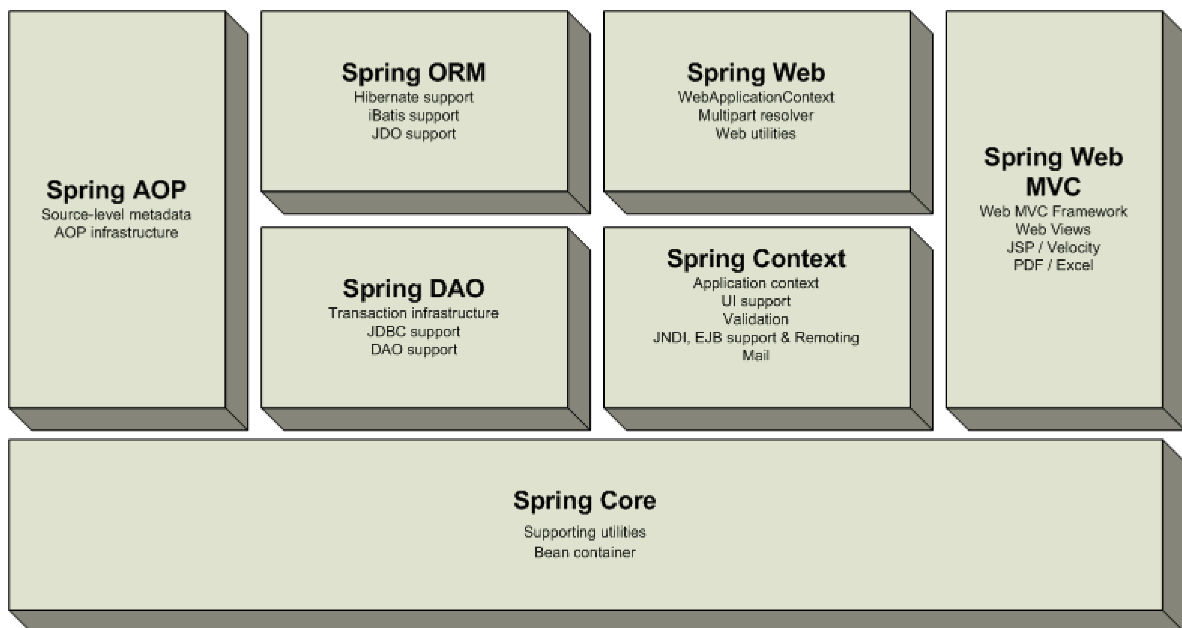


Figura 61 Arquitectura de Spring

Dos de los paquetes más importantes de Spring son `org.springframework.beans` y `org.springframework.context`. El código de estos paquetes proporciona la base de las características principales de la Inversión de Control (*IoC*) de Spring.

BeanFactory proporciona un mecanismo avanzado de configuración que permite la administración de beans (objetos) de cualquier naturaleza, usando potencialmente cualquier tipo de almacenamiento. El contexto de la aplicación (*ApplicationContext*) se construye encima de *BeanFactory*, añadiendo mejoras y otras funcionalidades.

Resumiendo, *BeanFactory* proporciona la configuración del *framework* y la funcionalidad básica, mientras que el *ApplicationContext* añade además otras características mejoradas. Cualquier descripción sobre las capacidades y el comportamiento de *BeanFactory* se puedes aplicar al *ApplicationContext*.

Normalmente cuando se construyen aplicaciones basadas en J2EE, la mejor opción es usar el *ApplicationContext* porque ofrece todas las características de *BeanFactory* y además permite un enfoque más declarativo parar usar parte de la funcionalidad, lo que siempre suele ser recomendable.

11.4.2.1 Spring Core

Esta parte es la que provee la funcionalidad esencial del *framework*. Está compuesta por *BeanFactory* que utiliza el patrón de Inversión de Control (*Inversion of Control*) y configura los objetos a través de Inyección de Dependencia (*Dependency Injection*).

Bean Factory

Es uno de los componentes principales del núcleo de Spring. Es una implementación del patrón *Factory*, pero a diferencia de las demás implementaciones de este patrón, que muchas veces sólo producen un tipo de objeto, `BeanFactory` es de propósito general, ya que puede crear muchos tipos diferentes de *Beans*. Los *Beans* se pueden llamar por su identificador y ellos mismos se encargan de manejar las relaciones entre objetos. Se soportan objetos de dos tipos diferentes:

- *Singleton* – existe únicamente una instancia compartida de un objeto con un nombre particular, que se puede llamar o devolver cada vez que se necesite. Este modo está basado en el patrón de diseño que lleva el mismo nombre.
- *Prototype* – también conocido como *non-singleton*. En este método cada vez que se realiza una llamada se crea un nuevo objeto independiente.

BeanFactory es el contenedor real que instancia, configura y administra los beans. Estos beans normalmente colaboran unos con otros, y los por tanto tienen dependencias entre ellos. Estas dependencias se reflejan en los datos de configuración usados por *BeanFactory*.

La configuración de un *BeanFactory,* en el caso más básico, consiste de definiciones de uno o más beans que deberá administrar. En una factoría definida a través de XML, los beans se declaran de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
 "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

        <bean id="GradingService"
                class="es.eucm.assessment.services.GradingServiceImpl" />

        <bean id="ItemSession"
                class="es.eucm.assessment.session.ItemSessionImpl"
                singleton="false">
                        <property name="gradingService">
                                <ref bean="GradingService"/>
                        </property>
      </bean>

                        . . .

        <bean id="Item"
              class="es.eucm.assessment.pojo.ItemImpl" />

</beans>
```

Las definiciones de los beans se representan a través de objetos *BeanDefinition* que contienen (entre otra información) los siguientes detalles:

- Identificador del bean (*id*) – cada bean tiene uno o más identificadores. Éstos deben ser únicos dentro del *BeanFactory*/*ApplicationContext* en el que esté almacenado. Un bean tendrá casi siempre un único nombre, pero en el caso de tener más de un identificador, todos los demás identificadores se pueden considerar alias.
- Nombre de la clase (*class*) – clase que implementa la definición del bean.
- Elementos de configuración del comportamiento del bean – como por ejemplo *prototype* o *singleton,* métodos de inicialización y destrucción, etc.
- Argumentos del constructor o propiedades del bean – como por ejemplo el límite de un pool de conexiones de una base de datos.
- Otros beans necesarios para este bean (colaboradores) – también se llaman dependecias.

Inversion of Control

Una de las funcionalidades más importantes de Spring es el uso del patrón de Inversión de Control (*IoC, Inversion of Control)* `utilizado por el` *`BeanFactory`*. Esta parte se encarga de separar del código de la aplicación que se está desarrollando, los aspectos de configuración y las especificaciones de dependencia del *framework*. Todo esto se realiza configurando los objetos a través de Inyección de Dependencia o *Dependency Injection,* que se explicará más adelante.

Una forma sencilla de explicar el concepto de *IoC* es el *Principio de Hollywood: No nos llames, nosotros te llamaremos*. Traduciendo este principio a nuestro contexto, en lugar de que el código de la aplicación llame a una clase de una librería, un *framework* que utiliza *IoC* llama al código. Por esta razón se llama *Inversión*, ya que invierte la acción de llamada a alguna librería externa.

Dependency Injection

Con este principio, en lugar de que el código de la aplicación utilice el API del *framework* para resolver dependencias como parámetros de configuración u objetos colaborativos, las clases de la aplicación exponen o muestran sus dependencias a través de métodos o constructores que el *framework* puede llamar con el valor apropiado en tiempo de ejecución, basado en la configuración.

El principio básico es que los beans definen sus dependencias (es decir, los otros objetos con los que trabajan) sólo a través de argumentos para métodos constructores, argumentos para una factoría o propiedades que se inicializan una vez el objeto se ha creado o devuelto a través de una factoría. Por lo tanto, el trabajo del contenedor es el de *inyectar* estas dependencias cuando se crea el bean. Esto es fundamentalmente lo inverso (de aquí el nombre) del bean instanciando o localizando sus dependencias por si mismo usando clases o constructores directamente.

Se ve claramente que el uso de *IoC* proporciona código mucho más limpio y además se logra un alto grado de desacoplamiento entre los beans porque los éstos no buscan sus dependencias directamente.

La inversión de control / Inyección de dependencias tiene dos variantes principales:

- Inyección de dependencias basada en métodos *set* (*setter-based*) – se realiza llamando a los métodos *set* de los beans después de invocar a un constructor sin argumentos o a un método sin argumentos de una factoría estática para instanciar al bean. Spring generalmente recomienda el uso de este tipo de inyección de dependencias porque un constructor con muchos argumentos se puede volver poco manejable, especialmente cuando algunas propiedades son opcionales.
- Inyección de dependencias basada en constructores (*constructor-based*) – se realiza invocando a un constructor con un número de argumentos, cada uno representando a un colaborador o una propiedad. Aunque Spring generalmente recomienda la otra aproximación para la inyección de dependencias, también se soporta completamente la inyección basada en constructores para permitir el uso de beans ya predefinidas que puedan proporcionar constructores con argumentos pero métodos *set*. Además, esta aproximación se puede usar como mecanismo para asegurar que beans sencillos no se construyan en un estado invalido.

La resolución de dependencias de los beans se desarrolla generalmente así:

- Se crea e inicializa el *BeanFactory* con la configuración que describe a todos los beans. Normalmente se usa una variante para el *BeanFactory* o *ApplicationContext* que soporta formato XML para los ficheros de configuración.
- Cada bean tiene dependencias expresadas en forma de propiedades. Estas dependencias serán proporcionadas al bean en el momento de creación.
- Cada propiedad o argumento del constructor es una definición real del valor a establecer o una referencia a otro bean en el *BeanFactory*.
- Es importante darse cuenta que Spring valida la configuración para cada bean del *BeanFactory* cuando éste se crea, incluyendo la validación de las propiedades que sean referencias a otros beans válidos. Sin embargo, las propiedades del bean no se asignan realmente hasta que se crea. Para beans que son *singleton*, la creación se realiza cuando el *BeanFactory* es creado, pero en cualquier otro caso el bean se crea cuando se necesita.
- Generalmente se puede suponer que Spring realizará bien las cosas. Se resolverán los problemas de configuración, referencias a beans no existentes y dependencias circulares. Las propiedades se asignan y se resuelven las dependencias tan tarde como sea posible, que suele ser cuando el bean es creado.

A continuación se muestra un ejemplo:
Primero se muestra una parte del fichero de configuración en XML en el que se definen algunos beans. A continuación se muestra el código del bean, con el método *set* correspondiente para realizar la inyección de dependencias.

```
<bean id="GradingService"
       class="es.eucm.assessment.services.GradingServiceImpl" />

<bean id="ItemSession"
       class="es.eucm.assessment.session.ItemSessionImpl"
       singleton="false">
              <property name="gradingService">
                     <ref bean="GradingService"/>
              </property>
</bean>
```

```
public class ItemSessionImpl implements ItemSession {

       private GradingService gradingService;

       public void setGradingService(GradingService service){
              gradingService = service;
       }

              . . .

}
```

Como se puede ver, el método *set* de ItemSessionImpl se ha declarado para que coincida con la propiedad definida en el fichero de configuración.

11.4.2.2 Spring Context

Mientras que el paquete *beans* proporciona la funcionalidad básica para administrar y manipular beans, normalmente de un modo programático, Spring *context* añade el *ApplicationContext* que mejora la funcionalidad del *BeanFactory*.

En sí, *Spring Context* es un archivo de configuración que provee de información contextual al *framework* general. Además provee servicios *enterprise* como JNDI, EJB, e-mail y validación.

Application Context

La base para el paquete *context* es la interfaz *ApplicationContext* que se encuentra en `org.springframework.context.` Derivando de la interfaz de *BeanFactory,* Spring Context proporciona toda la funcionalidad de ese interfaz y además añade información de la aplicación que se puede utilizar por todos los componentes, proporcionando además lo siguiente:

- Localización y reconocimiento automático de las definiciones de los *Beans*
- Carga de múltiples contextos jerarquizados, permitiendo que cada uno se enfoque en una determinada capa.
- Herencia de contextos
- Jerarquía de mensajes y soporte para internacionalización (i18n)
- Acceso a recursos
- Propagación de eventos, para permitir que los objetos de la aplicación publicar y opcionalmente registrarse para ser notificados de los eventos.

Como el *ApplicationContext* incluye toda la funcionalidad de *BeanFactory*, se recomiendo normalmente el uso del primero, excepto para algunas situaciones en las que el consumo de memoria sea crítico (es decir, en un *Applet*).

11.4.2.3. Spring AOP

La programación orientada a aspectos (*Aspect-oriented programming, AOP*) es una técnica que permite a los programadores modularizar ya sea las preocupaciones *crosscutting*, o el comportamiento de las divisiones de responsabilidad, como:

- Persistencia
- Manejo de transacciones
- Seguridad
- *Logging*
- *Debugging*

AOP es un enfoque diferente y un poco más complicado de acostumbrarse en comparación con la programación orientada a objetos, pero es un complemento no un rival o una causa de conflicto.

Spring AOP es portable entre servidores de aplicación y funciona tanto en servidores Web como en contenedores EJB.

Spring AOP soporta las siguientes funcionalidades:

- Intercepción – se puede insertar comportamiento personalizado antes o después de invocar a un método en cualquier clase o interfaz.
- Introducción – especificando que un *advice* (acción tomada en un punto particular durante la ejecución de un programa) debe causar que un objeto implemente interfaces adicionales.
- *Pointcuts* dinámicos y estáticos – para especificar los puntos en la ejecución del programa donde debe de haber intercepción.

11.4.2.4 Spring ORM

En lugar de proporcionar su propio módulo ORM (*Object-Relational Mapping*), para los usuarios que no se sientan confiados en utilizar simplemente JDBC, Spring propone un módulo que soporta los *frameworks* ORM más populares del mercado, entre ellos:

- Hibernate (2.1 y 3.0) – es una herramienta de mapeo O/R *OpenSource* muy popular que utiliza su propio lenguaje de *query* llamada HQL.
- iBATIS SQL Maps (1.3 y 2.0) – es una solución sencilla pero poderosa para hacer externas las declaraciones de SQL en archivos XML.
- Apache OJB (*ObJectRelationalBridge*, 1.0) – es una plataforma de mapeo O/R con múltiples APIs para clientes.
- Otros como JDO (*Java Data Objects*, 1.0 y 2.0) y Oracle TopLink.

Todo esto se puede utilizar en conjunto con las transacciones estándar del *framework*. Spring e Hibernate es una combinación muy popular.

Algunas de las ventajas que brinda Spring al combinarse con alguna herramienta ORM son:

- Manejo de sesión – Spring hace de una forma más eficiente, sencilla y segura la forma en que se manejan las sesiones de cualquier herramienta ORM que se quiera utilizar.
- Manejo de recursos – se puede manejar la localización y configuración de los SessionFactories de Hibernate o las fuentes de datos de JDBC.
- Manejo de transacciones integrado – se puede utilizar una plantilla (*template*) de Spring para las diferentes transacciones ORM.
- Envolver excepciones – con esta opción se pueden envolver todas las excepciones para evitar las engorrosas declaraciones y los bloques *try–catch* en cada segmento de código.
- Evita limitarse a un sólo producto – si se desea migrar o actualizar a otra versión de un ORM distinto o del mismo, Spring trata de no crear dependencias entre la herramienta ORM, el mismo Spring y el código de la aplicación, para que cuando sea necesario migrar a un nuevo ORM no sea necesario realizar tantos cambios.
- Facilidad de prueba – Spring trata de crear pequeños pedazos que se puedan aislar y probar por separado, ya sean sesiones o un *datasource*.

11.4.2.5 Spring DAO (DAO y JDBC)

El patrón DAO (*Data Access Object*) es uno de los patrones más importantes y usados en aplicaciones J2EE, y la arquitectura de acceso a los datos de Spring provee un buen soporte para este patrón.

Existen dos opciones para llevar a cabo el acceso, conexión y manejo de bases de datos: utilizar alguna herramienta ORM o utilizar la plantilla de JDBC (*Java Database Connectivity*) que brinda Spring. La elección de una de estas dos herramientas es totalmente libre y el desarrollador se debe basar en la complejidad de la aplicación para elegir entre ellas. Si es una aplicación sencilla en la que únicamente una clase realizará la conexión con la base de datos, entonces la mejor opción sería Spring JDBC. En caso contrario, cuando se requiera un mayor soporte y la aplicación sea más robusta se recomienda utilizar una herramienta ORM.

El uso de JDBC muchas veces lleva a repetir el mismo código en distintos lugares como por ejemplo al crear la conexión, buscar información, procesar los resultados y cerrar la conexión. El uso de las dos tecnologías mencionadas anteriormente nos ayuda a mantener simple este código y evitar que sea tan repetitivo, además minimiza los errores al intentar cerrar la conexión con algunas bases de datos.

11.4.2.6 Spring Web

El módulo Web de Spring se encuentra en la parte superior del módulo de contexto, y provee el contexto para las aplicaciones Web. Este módulo proporciona el soporte necesario para la integración con el *framework Struts* de Yakarta.

Este módulo también se encarga de diversas operaciones Web como por ejemplo ejecutar las peticiones multi-parte que puedan ocurrir al realizar cargas de archivos y la relación de los parámetros de las peticiones con los objetos correspondientes (*domain objetcs o business objects*).

11.4.2.7 Spring Web MVC

Spring brinda un patrón MVC (*Model View Controller*) para aplicaciones Web bastante flexible y altamente configurable, pero esta flexibilidad no le quita sencillez, ya que se pueden desarrollar aplicaciones sencillas sin tener que configurar muchas opciones.

Para esto se puede utilizar muchas tecnologías ya que Spring ofrece soporte para JSP, *Struts* y Velocity, entre otros.

El módulo Web MVC de Spring presenta algunas similitudes con otros *framework* que existen en el mercado, pero estas características lo vuelven único:

- Spring hace una clara división entre controladores, modelos de *JavaBeans* y vistas.
- El MVC de Spring esta basado en interfaces y es bastante flexible.
- Provee interceptores (*interceptors*) al igual que controladores.
- Spring no obliga a utilizar JSPs como única tecnología para la vista (*View*).
- Los controladores son configurados de la misma manera que los demás objetos en Spring, a través de IoC.
- Las capas Web (*Web tiers)* son más sencillas de probar que en otros *frameworks*.

Para intentar comprender cada parte de la arquitectura del Web MVC de Spring, se presenta en la figura 62 el ciclo de vida de una petición (*request*):

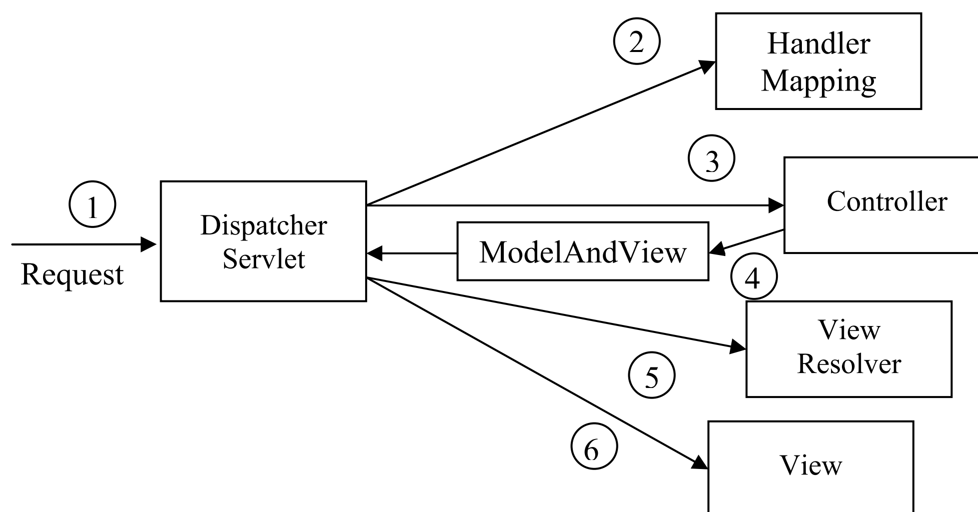


Figura 62 Ciclo de vida de una petición Spring.

1. El navegador manda una petición y lo recibe un `DispatcherServlet`
2. Se debe escoger que controlador (`Controller)` manejará la petición. Para esto el `HandlerMapping` mapea los diferentes patrones de URL hacia los controladores, y se le devuelve al `DispatcherServlet` el `controlador` elegido.

3. El `controlador` elegido toma la petición y ejecuta la tarea.
4. El `controlador` devuelve un *`ModelAndView`* al *`DispatcherServlet`*.
5. Si el *`ModelAndView`* contiene un nombre lógico de un *`View`* se tiene que utilizar un *`ViewResolver`* para buscar ese objeto *`View`* que representará la petición modificada.
6. Finalmente el *`DispatcherServlet`* despacha la petición al *`View`*.

Spring cuenta con una gran cantidad de controladores de los cuales se puede elegir dependiendo de la tarea, entre los más populares se encuentra: *SimpleController* y *AbstractController*.

11.4.3. Administración de Transacciones con Spring

Spring proporciona una abstracción consistente para la administración de transacciones, siendo esta abstracción es uno de los mayores beneficios de Spring. Además también aporta las siguientes ventajas:

- Proporciona un modelo de programación consistente para diferentes APIs como JTA, JDBC o Hibernate.
- Proporciona una gestión de transacciones más sencilla y fácil de usar que la mayoría de las APIs para transacciones.
- Da soporte para la administración declarativa de transacciones.

Los desarrolladores tienen principalmente dos opciones para realizar esta administración: transacciones globales o transacciones locales.

Las transacciones globales se administran a través de un servidor de aplicaciones usando JTA (*Java Transaction API*), lo que permite la posibilidad de gestionar múltiples recursos transaccionales (aunque la mayoría de las aplicaciones usa un único recurso). El uso de JTA para el código es la principal desventaja de esta opción debido a su complejo e incomodo modelo de excepciones.

Las transacciones locales son más fáciles de usar pero también tienen desventajas significativas: tienden a invadir el modelo de programación. Por ejemplo, código que administra transacciones usando JDBC no puede ejecutarse dentro de una transacción JTA.

Spring resuelve estos problemas permitiendo a los desarrolladores el uso de un modelo consistente en cualquier entorno. También proporciona tanto administración programática como declarativa. Esta última es la más recomendaba en la mayoría de los casos gracias a que la administración de las transacciones se puede hacer con poco código y por lo tanto no depende de ningún API.

11.4.3.1. Administración Declarativa de Transacciones

Spring ofrece la administración declarativa de transacciones a través de la Programación Orientada a Aspectos (*AOP, Aspect Oriented Programming*). Este tipo de transacciones es el más usado porque es la opción que menos impacto genera en el código de la aplicación. Las características principales son:

- Spring permite especificar el comportamiento de la transacción para cada método a través de *AOP*.
- Spring trabaja en cualquier entorno. Puede trabajar con JDBC, JDO o Hibernate, simplemente cambiando la configuración.
- Spring ofrece reglas declarativas de *rollback*.
- Spring no soporta propagación de contextos en las transacciones a través de llamadas remotas. Se recomienda usar EJB (*Enterprise JavaBeans*) si se desea usar esta característica.

El concepto de reglas de *rollback* es importante: permite especificar que tipo de excepciones deberían causar un *rollback* automático. Esto se especifica declarativamente en la configuración, no en código Java. Esto tiene la principal ventaja de que los objetos de negocio no dependen de la infraestructura de las transacciones.

La forma habitual de configurar un proxy de transacciones en Spring es a través del uso de `TransactionProxyFactoryBean`. Esta factoría es simplemente una versión especializada de la factoría genérica `ProxyFactoryBean` que crea un proxy para envolver el objeto para el que se van a gestionar las transacciones.

Para usar el bean `TransactionProxyFactoryBean`, primero se necesita especificar el objeto que se va a envolver con el proxy de transacciones a través del atributo `target`. Este objeto es normalmente un bean POJO (*Plain Old Java Object*). También se debe definir una referencia a `HibernateTransactionManager`. Finalmente, se deben especificar los atributos de la transacción, `transactionAttributes`. Estos atributos contienen la definición de la semántica de las transacciones así como los métodos donde se aplican. Consideremos el siguiente ejemplo del proyecto QTI:

```
<bean id="ValueDAOService"
        class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
        <property name="transactionManager" ref="transactionManager" />
        <property name="target" ref="ValueDAOTarget" />
        <property name="transactionAttributes">
                <props>
                        <prop key="*">PROPAGATION_REQUIRED</prop>
                </props>
        </property>
</bean>

<bean id="ValueDAOTarget"
        class="es.eucm.assessment.dao.pojo.outcomes.ValueDAOImpl">
        <property name="sessionFactory" ref="mySessionFactory" />
</bean>

<bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="mySessionFactory"/>
</bean>
```

```
<bean id="mySessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource" ref="myDataSource"/>
        <property name="mappingResources"> [...] </property>
        <property name="hibernateProperties"> [...] </property>
</bean>
```

Los atributos de la transacción se definen a través de un par consistente en el nombre del método y el valor. El valor es una cadena de texto con el siguiente formato:

```
PROPAGATION_NAME,ISOLATION_NAME,readOnly,timeout_NNNN,+Exception1,-Exception2
```

PROPAGATION_NAME: normalmente todo el código se ejecuta dentro del ámbito de una misma transacción. Sin embargo hay varias opciones para especificar el comportamiento si un método transaccional se ejecuta cuando otra transacción ya existía.
ISOLATION_NAME: grado de aislamiento de esta transacción respecto a otras transacciones.
timeout_NNNN: específica cuanto tiempo se debe esperar a la transacción antes de hacer *time out*, provocando automáticamente *rollback*.
+Exception1: especifica que esta excepción fuerza un *commit* cuando se lanza.
−Exception2: especifica que esta excepción fuerza un rollback cuando se lanza.
A continuación se muestra un diagrama que representa el modo en el que Spring ensambla todos los beans de una parte del proyecto QTI según la configuración de la página anterior.

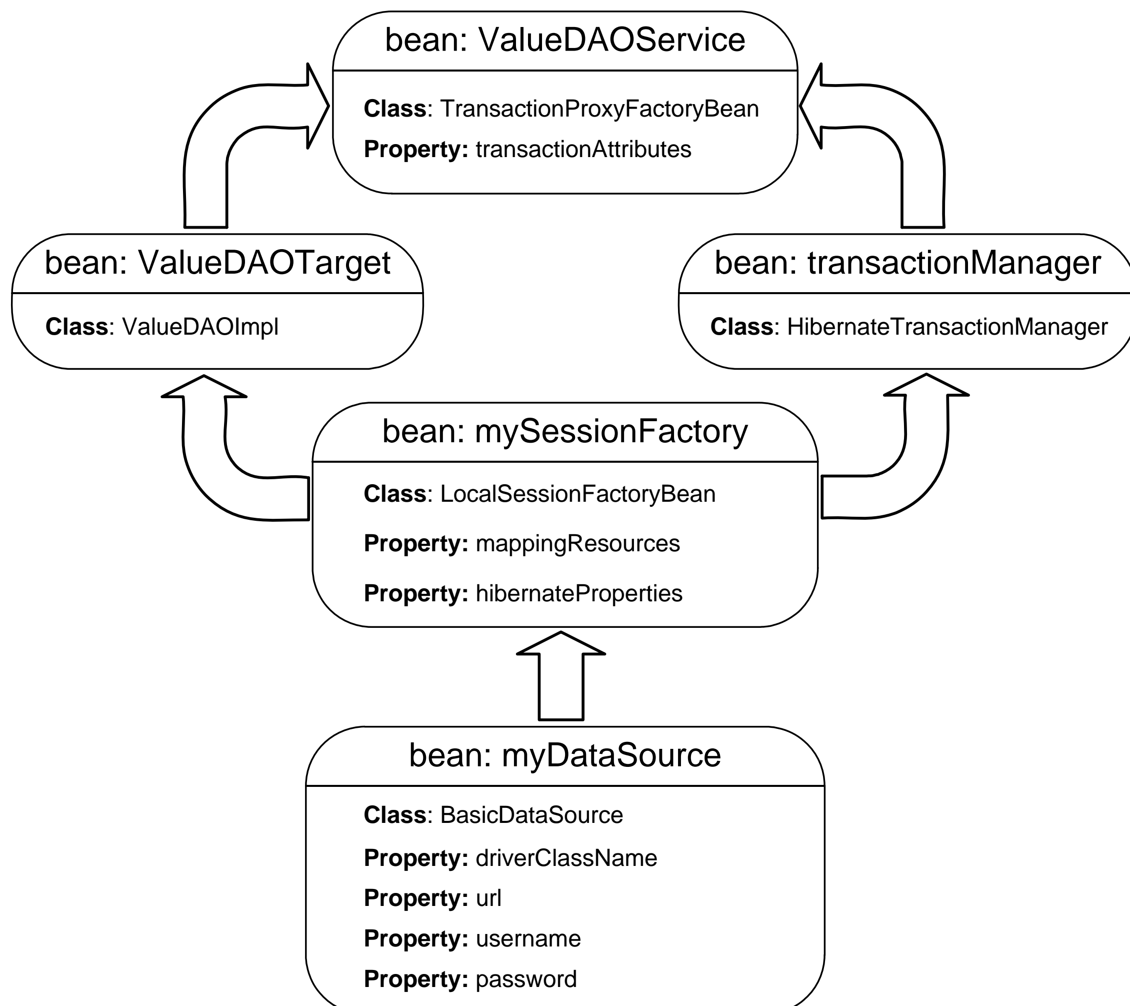


Figura 63 Generación de los beans basado en la configuración de Spring

11.4.3.2. Administración Programática de Transacciones

Spring ofrece dos mecanismos para la administración programática de transacciones:

- Usando `TransactionTemplate`: se adopta la misma aproximación que con otras plantillas de Spring.
- Usando `PlatformTransactionManager: similar al API de JTA.`

11.4.3.3. Administración Declarativa vs. Programática

La elección del tipo de gestión de transacciones dependerá de la arquitectura de la aplicación. La administración de transacciones programática es una buena opción si se tiene un número pequeño de operaciones transaccionales. Por el contrario, si la aplicación tiene muchas operaciones con transacciones, la administración declarativa es la adecuada porque mantiene la gestión de las transacciones fuera de la lógica de negocios y es fácilmente configurable en Spring.

11.5. Interconexión entre las distintas tecnologías

Maven

Maven piensa en términos de proyectos. Un proyecto es cualquier directorio que contenga un archivo *project.xml*. Este archivo se conoce con el nombre de POM (*Project Object Model*), y contiene toda la información y estructura acerca del proyecto; nombre del proyecto, tipo, versión, autor, dependencias, etc. Un proyecto puede consistir de varios subproyectos, sin embargo estos subproyectos son igualmente tratados como proyectos.

Project.xml (Project Object Model → POM)

Fichero XML que describe declarativamente un proyecto, describe la estructura de directorios del proyecto. Es el meta-datos del proyecto que incluye información de control del proyecto y dependencias (proyectos que dependen de otros proyectos, esto es, un fichero, un jar u otras cosas que dependen del proyecto).

Nuestro *project.xml* (extracto) es el siguiente:

```
<project>
  <groupId>es.eucm</groupId>
  <artifactId>eQTI_API_impl</artifactId>
  <currentVersion>1.0-SNAPSHOT</currentVersion>
  <name>API Implementation - IMS QTI Library Project</name>
  <organization>
    <name>e-UCM Research Group</name>
    <url>http://www.e-ucm.es/</url>
  </organization>
  <inceptionYear>2005</inceptionYear>
  <dependencies>
    <dependency>
      <groupId>es.eucm</groupId>
      <artifactId>common</artifactId>
```

```
      <version>1.0-SNAPSHOT</version>
    </dependency>
        . . .
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
    </dependency>
</dependencies>
  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
<unitTestSourceDirectory>src/test/java</unitTestSourceDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
      </resource>
      <resource>
        <directory>target/generated-source/java</directory>
      </resource>
    </resources>
    <!--
    <unitTest>
      <resources>
        <resource>
          <directory>src/test/resources</directory>
        </resource>
        <resource>
          <directory>target/generated-source/java</directory>
        </resource>
      </resources>
      <includes>
        <include>**/*Test.java</include>
      </includes>
    </unitTest>-->
  </build>
</project>
```

GroupID: Un grupo ID es un identificador universalmente único para un proyecto. Mientras esto es a menudo justo el nombre del proyecto esto es útil para usar un nombre de paquete totalmente cualificado para distinguirlo de otro proyecto con un nombre similar.
ArtifactID: Cualquiera de las cosas producidas o usadas por un proyecto. Ejemplos de artefactos producidos por Maven para un proyecto incluyen: jars, fuentes y distribuciones binarios, wars. Cada artefacto es únicamente definido por un *groupID* y un artefacto ID cualquiera es único en un grupo.

Project.properties

Fichero que define propiedades específicas para el proyecto y puede ser usado para poner los valores para enchufar y propiedades Maven que son apropiadas para cada proyecto. Este archivo debería ser chequeado en tu repositorio fuente y distribuido.

En nuestro caso este fichero es utilizado por que Maven para generar los archivos **.hbm.xml* a partir de los *tags* incluidos en los *beans* de nuestra aplicación y así poder realizar la persistencia con Hibernate:

```
maven.compile.source=1.4
maven.compile.target=1.4
maven.compile.debug=true
# XDoclet 2 first instance
maven.xdoclet.0=hibernate
# XDoclet 2 first instances filesets
maven.xdoclet.0.fileset.0.dir=${pom.build.sourceDirectory}
maven.xdoclet.0.fileset.0.include=**/*.java
# XDoclet 2 first instance hibernate component
```

```
maven.xdoclet.0.hibernate=org.xdoclet.plugin.hibernate.HibernateMappingPlugin
maven.xdoclet.0.hibernate.destdir=${basedir}/target/generated-source/java
maven.xdoclet.0.hibernate.version=3.0
# XDoclet
maven.xdoclet.springdoclet.destDir=${basedir}/target/generated-source/java
maven.xdoclet.springdoclet.verbose=true
maven.xdoclet.springdoclet.force=true
maven.xdoclet.springdoclet.fileset.0=true
maven.xdoclet.springdoclet.fileset.0.include=**/*.java
maven.xdoclet.springdoclet.springxml.0=true
maven.xdoclet.springdoclet.springxml.0.destinationFile=application-context.xml
#maven repositories
maven.repo.remote=http://www.ibiblio.org/maven/,http://cvs.sakaiproject.org/maven/,http:
//dist.codehaus.org/
maven.test.skip=true
```

Un repositorio es una carpeta o estructura de almacenamiento, de artefactos de proyecto, que puede ser local o de red compartido. Estos artefactos son organizados bajo la siguiente estructura:

MAVEN_REPO/group id/artifact type/project_version.extension

Ejemplo: Un jar de Maven:

/repository/maven/jars/maven-1.0.2.jar

Hay diferentes repositorios que usa Maven. Los repositorios remotos son una lista de repositorios de descarga. Esto puede incluir un repositorio de Internet, un servidor y una compañía repositoria privada.
Un repositorio central es el único artefacto generado (por desarrolladores de una compañía de instancias).
El repositorio local es el único que tú tendrás en tu ordenador. Los artefactos son descargados solo una vez (menos si son SNAPSHOT) desde un repositorio remoto a tu repositorio local.

Maven.xml

En este fichero se definen los *goals* específicos del proyecto. Los *goals* son funciones ejecutables que actúan sobre un proyecto, escritos en *scripts* Jelly(xml ejecutable-Jelly en un lenguaje de *script* basado en xml, combinación de Ant y etiquetas JSTL para propósito más general). Pueden ser específicos del proyecto o reutilizables entre proyectos. Actúan sobre un proyecto utilizando los POM. En términos OOP, piensa en tu proyecto y sus meta-datos como un objeto, y los *goals* son los métodos que actúan sobre el proyecto, esto es, lo que es ejecutado para efectuar una acción en el proyecto.

Además Maven esta organizada en *plugins* (reutilización de *goals*) que proporcionan *goals* y usan el meta-datos encontrado en el POM para ejecutar sus tareas. Ejemplos de *plugins* son: jar, eclipse, war. Los *plugins* son escritos en Jelly y pueden ser añadidos, eliminados y editados en tiempo de ejecución. Así en el comando; jar:jar, el primer jar es el *plugin* y el segundo es el *goal*.

En nuestro proyecto tenemos el siguiente *maven.xml* en el que se invoca al *goal springdoclet* del *plugin xdoclet* que junto con el archivo *project.properties* genera los **.hbm.xml* que necesitamos para la persistencia con Hibernate.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:ant="jelly:ant"
         xmlns:j="jelly:core"
         default="eQTI:build">
  <preGoal name="java:compile">
    <attainGoal name="xdoclet2"/>
    <attainGoal name="xdoclet:springdoclet"/>
  </preGoal>
</project>
```

Ant

Ant es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (*build*). Es similar a *make* pero sin las engorrosas dependencias del sistema operativo. Esta herramienta, hecha en Java, tiene la ventaja de no depender de las órdenes de *shell* de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multi-plataforma.

Build.xml

Como hemos dicho, Ant se basa en ficheros XML. Normalmente configuramos el trabajo a hacer con nuestra aplicación en un fichero llamado *build.xml*, así que vamos a ver algunas de las etiquetas que podemos meter ahí.

project
: Este es el elemento raíz del fichero XML, y como tal, así que solo puede haber uno en todo el fichero, el que se corresponde a nuestra aplicación Java.

target
: Un *target* u objetivo es un conjunto de tareas (ver el siguiente elemento, *task*) que queremos aplicar a nuestra aplicación en algún momento. Se puede hacer que unos objetivos dependan de otros, de forma que eso lo trate Ant automáticamente.

task
: Un *task* o tarea es un código ejecutable que aplicaremos a nuestra aplicación, y que puede contener distintas propiedades (como por ejemplo el *classpath*). Ant incluye ya muchas básicas, como compilación y eliminación de ficheros temporales, pero podemos extender este mecanismo si nos hace falta.

property
: Una propiedad o *property* es simplemente algún parámetro (en forma de par nombre-valor) que necesitamos para procesar nuestra aplicación, como el nombre del compilador, etc. Ant incluye ya las más básicas, como son *BaseDir* para el directorio base de nuestro proyecto, *ant.file* para el *path* absoluto del fichero *build.xml*, y *ant.java.version* para la versión de la JVM.

Utilizamos Ant para generar el archivo *schema-export.sql* a partir de los **.hbm.xml* generados por Maven, y así a través de este archivo poder comprobar la corrección de las estructuras necesarias para almacenar los datos oportunos en la base de datos.

En el fichero *build.xml* se define el repositorio donde tenemos los jar, la herramienta que vamos a utilizar y el *hibernate.properties* que define las características de configuración de Hibernate, para generar el *schema-export.sql*:

```
<project basedir="." default="schemaexport">
  <property name="maven.local.repo.dir" value="C:/Documents and
Settings/Programador/.maven/repository" />
  <target name="schemaexport" >
    <taskdef name="schemaexport"
             classname="org.hibernate.tool.hbm2ddl.SchemaExportTask">
      <classpath>
        <fileset dir="${maven.local.repo.dir}">
          <include name="**/*.jar"/>
        </fileset>
        <path location="target/classes" />
      </classpath>
    </taskdef>
    <copy file="hibernate.properties"
      tofile="target/classes/hibernate.properties"
      overwrite="true"
      verbose="true"
    />
    <schemaexport properties="target/classes/hibernate.properties"
                  quiet="no"
                  text="yes"
                  drop="no"
                  delimiter=";"
                  output="target/schema-export.sql">
      <fileset dir="target/classes">
        <include name="**/*.hbm.xml" />
      </fileset>
    </schemaexport>
  </target>
</project>
```

Hibernate

Hay varias maneras de conseguir la persistencia con Hibernate. Pero básicamente lo que hay que hacer es:

1. Crear tu tabla del SQL para guardar tus objetos persistentes.
2. Crear un JavaBean que represente ese objeto en código.
3. Crear un archivo de mapeo de manera que Hibernate sepa qué características del *bean* se mapean a que campos del SQL. Los archivos que definen el mapeo (**.hbm.xml*) describen cómo se relacionan clases y tablas y propiedades y columnas.
4. Crear un archivo de propiedades de manera que Hibernate conozca la configuración JDBC para acceder a la base de datos.
5. Comenzar a usar el Hibernate API.

Nosotros utilizamos un *plugin* de Maven para generar el fichero de mapeo automáticamente y a partir de ahí autogenerar el SQL de los *beans*, es decir, nosotros creamos los beans del punto 2 y a partir de ellos utilizando diferentes herramientas generamos automáticamente los puntos 1 y 3. Los puntos 4 y 5 lo explicamos a continuación:

Hibernate.properties (El archivo de propiedades de Hibernate)

Este es el archivo de propiedades de Hibernate con información sobre la conexión JDBC, que se encarga de determinar los aspectos relacionados con el gestor de bases de datos y las conexiones con él, esto es, se dice qué gestor de bases de datos usaremos y a qué base de datos nos conectaremos y cómo lo haremos, así tiene los ajustes para las secuencias de la conexión, las contraseñas, etc.
En este archivo, se pueden especificar muchísimas cosas. Nosotros nos hemos limitado a proporcionar los datos necesarios para que Hibernate se pueda conectar a nuestra base de datos HSQLDB. Este es nuestro código:

```
## HypersonicSQL
#hibernate.query.substitutions true 1, false 0, yes 'Y', no 'N'
hibernate.dialect=org.hibernate.dialect.HSQLDialect
hibernate.connection.driver_class=org.hsqldb.jdbcDriver
hibernate.connection.url=jdbc:hsqldb:file:testdb
hibernate.connection.username=sa
hibernate.connection.password=
hibernate.connection.pool_size=100
#hibernate.proxool.pool_alias=pool1
hibernate.show_sql=true
#hibernate.jdbc.batch_size=0
#hibernate.jdbc.use_streams_for_binary=true
#hibernate.use_outer_join=true
#hibernate.max_fetch_depth=1
hibernate.cache.use_query_cache=true
hibernate.cache.provider_class=org.hibernate.cache.HashtableCacheProvider
```

La configuración es sencilla. Especificamos el dialecto SQL y los datos necesarios para poder establecer una conexión con la base de datos vía JDBC (*Driver*, URL, usuario y contraseña). Hibernate se encarga de traducir el HQL al dialecto HSQL y de conectarse a la base de datos.

En cualquier aplicación que use Hibernate aparecen cuatro objetos básicos:

1. *Configuration*: es el objeto que contiene la información necesaria para conectarse a la base de datos. Es el encargado de leerse el archivo *Hibernate.properties*. También es el encargado de procesar la información correspondiente a los aparejamientos, es decir, es el encargado de leerse y verificar los archivos de emparejamiento *nombreDeClasse.hbm.xml*.
   ```
   Configuration conf = new Configuration();
   conf.addClass(nombreDeClasse.class);
   ```
2. *SessionFactory*: es una fábrica de *Sessions*. Un objeto *Configuration* es capaz de crear una *SessionFactory* ya que tiene tota la información necesaria.
   ```
   SessionFactory sessionFactory = conf.buildSessionFactory();
   ```
3. *Session*: La principal interfaz entre la aplicación Java e Hibernate. Es la que mantiene las conversaciones entre la aplicación y la base de datos. Permite añadir, modificar y borrar objetos en la base de datos.
   ```
   Session session = sessionFactory.openSession();
   ```
4. *Transaction*: Como su nombre indica, se encarga de la transaccionalidad. Permite definir unidades de trabajo.
   ```
   Transaction tx = session.beginTransaction();
   [...]
   session.save(…);
   tx.commit();
   [...]
   tx.rollback();
   ```

Nosotros inicialmente para hacer los tres primeros puntos utilizamos la clase *HibernateUtil.java* pero esto ahora se realiza a través de Spring.

Spring

Al utilizar Spring ya no es necesaria la clase HibernateUtil ya que Spring se encarga de realizar todas esas operaciones.

Applicacion context

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<!--Application context -->
<beans>
<bean id="GradingService" class="es.eucm.assessment.services.GradingServiceImpl" />

<bean id="ItemSession"
         class="es.eucm.assessment.session.ItemSessionImpl"
         singleton="false">
              <property name="gradingService">
                     <ref bean="GradingService"/>
              </property>
</bean>

<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource"
       destroy-method="close">
              <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
              <property name="url" value="jdbc:hsqldb:file:testdb"/>
              <property name="username" value="sa"/>
              <property name="password" value=""/>
</bean>

<bean id="mySessionFactory"
  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="myDataSource"/>
  <property name="mappingResources">
     <list>
       <value>es/eucm/assessment/pojo/outcomes/MapEntryImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/outcomes/MappingImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/outcomes/CorrectResponseImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/outcomes/ValueImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/outcomes/VariableImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/outcomes/VariableDeclarationImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/AssessmentImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/BasicModelElementImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/ControlFlagsImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/FeedbackImpl.hbm.xml</value>
       <value>es/eucm/assessment/pojo/HeadingImpl.hbm.xml</value>
       <value>es/eucm/assessment/session/ItemSessionImpl.hbm.xml</value>
     </list>
   </property>
   <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</prop>
      <prop
       key="hibernate.current_session_context_class">
              es.eucm.persistence.ExtendedThreadLocalSessionContext</prop>
       <prop key="hibernate.show_sql">true</prop>
       <prop key="hibernate.hbm2ddl.auto">create</prop>
     </props>
    </property>
</bean>

<bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
       <property name="sessionFactory"><ref bean="mySessionFactory"/></property>
</bean>
```

```
<bean id="ValueDAOService"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"><ref bean="transactionManager"/></property>
        <property name="target"><ref bean="ValueDAOTarget"/></property>
        <property name="transactionAttributes">
                <props> <prop key="*">PROPAGATION_REQUIRED</prop></props>
        </property>
</bean>

<bean id="ValueDAOTarget" class="es.eucm.assessment.dao.pojo.outcomes.ValueDAOImpl">
        <property name="sessionFactory" ref="mySessionFactory" />
</bean>

<bean id="VariableDeclarationDAO"
class="es.eucm.assessment.dao.pojo.outcomes.VariableDeclarationDAOImpl">
        <property name="sessionFactory" ref="mySessionFactory" />
</bean>

<bean id="ItemSessionDAOTarget" class="es.eucm.assessment.dao.pojo.ItemSessionDAOImpl">
        <property name="sessionFactory" ref="mySessionFactory" />
</bean>

<bean id="ItemSessionDAOService"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"><ref bean="transactionManager"/></property>
        <property name="target"><ref bean="ItemSessionDAOTarget"/></property>
        <property name="transactionAttributes">
                <props>
                        <prop key="*">PROPAGATION_REQUIRED</prop>
                </props>
        </property>
</bean>

<bean id="ItemDAOTarget" class="es.eucm.assessment.dao.pojo.ItemDAOImpl">
        <property name="sessionFactory" ref="mySessionFactory" />
</bean>

<bean id="ItemDAOService"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"><ref bean="transactionManager"/></property>
        <property name="target"><ref bean="ItemDAOTarget"/></property>
        <property name="transactionAttributes">
                <props><prop key="*">PROPAGATION_REQUIRED</prop></props>
        </property>
</bean>

<bean id="Value" class="es.eucm.assessment.pojo.outcomes.ValueImpl" singleton="false" />

<bean id="VariableDeclaration" class="es.eucm.assessment.pojo.outcomes.VariableDeclarationImpl"
singleton="false"/>

<bean id="Variable" class="es.eucm.assessment.pojo.outcomes.VariableImpl"
singleton="false" />

<bean id="Assessment" class="es.eucm.assessment.pojo.AssessmentImpl" singleton="false"/>

<bean id="BasicModelElement" class="es.eucm.assessment.pojo.BasicModelElementImpl"
singleton="false"/>

<bean id="ControlFlags" class="es.eucm.assessment.pojo.ControlFlagsImpl"
singleton="false"/>

<bean id="Feedback" class="es.eucm.assessment.pojo.FeedbackImpl" singleton="false"/>

<bean id="Heading" class="es.eucm.assessment.pojo.HeadingImpl" singleton="false"/>

<bean id="Item" class="es.eucm.assessment.pojo.ItemImpl" singleton="false"/>

<bean id="DBUtils" class="es.eucm.persistence.DBUtils" singleton="true"/>

</beans>
```

Eclipse

Hibernate.cfg.xml (El archivo de configuración de Hibernate)

Es igual al fichero *Hibernate.properties* sólo que en este caso lo utiliza Eclipse para saber como se configura el *jdbcDriver* para Hsqlbd y para saber cuales son las clases mapeadas:

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- Don't forget to copy your JDBC driver to the lib/ directory! -->
        <!-- Settings for a local HSQL (testing) database.-->
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.url">jdbc:hsqldb:file:testdb</property>
        <property name="connection.username">sa</property>
        <property name="connection.password" />
        <!-- Settings for a local PostgreSQL database.
        <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property name="query.substitutions">yes 'Y', no 'N'</property>
        <property
name="connection.driver_class">org.hibernate.ce.auction.persistence.pgsql.NativeAdapter<
/property>
        <property name="connection.url">jdbc:postgresql://localhost/test</property>
        <property name="connection.username">test</property>
        <property name="connection.password"></property>
        -->
        <!-- Settings for a MySQL database.
        <property name="dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost/test</property>
        <property name="connection.username">test</property>
        <property name="connection.password">test</property>
        -->
        <!-- Settings for an Oracle9/10g database.
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:ORAC9</property>
        <property name="connection.username">test</property>
        <property name="connection.password">test</property>
        -->
        <!-- Use the C3P0 connection pool.
        <property name="c3p0.min_size">3</property>
        <property name="c3p0.max_size">5</property>
        <property name="c3p0.timeout">1800</property>
        -->
        <!-- Use the Hibernate built-in pool for tests. -->
        <property name="connection.pool_size">1</property>

        <!-- Use EHCache but not the query cache. -->
        <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
        <property name="cache.use_query_cache">false</property>
        <property name="cache.use_minimal_puts">false</property>
        <property name="max_fetch_depth">3</property>
        <!-- Print SQL to stdout. -->
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <!-- Drop and then re-create schema on SessionFactory build, for testing. -->
        <property name="hbm2ddl.auto">create</property>
        <!-- Bind the getCurrentSession() method to the thread (don't use for EJBs) -->
        <property name="current_session_context_class">thread</property>
        <!-- Batch inserts are currently broken, no idea why... -->
        <property name="jdbc.batch_size">0</property>
<mapping resource="es/eucm/assessment/pojo/outcomes/ValueImpl.hbm.xml"/>
```

```
<mapping resource="es/eucm/assessment/pojo/outcomes/VariableDeclarationImpl.hbm.xml"/>
<mapping resource="es/eucm/assessment/pojo/outcomes/VariableImpl.hbm.xml"/>
<mapping resource="es/eucm/assessment/pojo/AssessmentImpl.hbm.xml"/>
<mapping resource="es/eucm/assessment/pojo/BasicModelElementImpl.hbm.xml"/>
<mapping resource="es/eucm/assessment/pojo/ControlFlagsImpl.hbm.xml"/>
<mapping resource="es/eucm/assessment/pojo/FeedbackImpl.hbm.xml"/>
<mapping resource="es/eucm/assessment/pojo/HeadingImpl.hbm.xml"/>
<mapping resource="es/eucm/assessment/session/ItemSessionImpl.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

11.6. J2EE

11.6.1. Introducción a J2EE

J2EE es un estándar que ofrece un modelo distribuido multicapa, componentes reutilizables, un modelo de seguridad unificado, control de transacciones flexibles y soporte para servicios Web a través de protocolos y estándares basados en XML.
La lógica de la aplicación se divide en componentes según su funcionalidad, y los componentes de la aplicación que constituyen la aplicación J2EE se instalan en diferentes máquinas dependiendo de la capa a la que pertenezcan.

- Componentes de la capa del cliente (*client-tier*) que se ejecutan en la máquina del cliente.
- Componentes de la capa Web (*Web-tier*) que se ejecutan en el servidor J2EE.
- Componentes de la capa de Negocios (*Bussiness-tier*) que se ejecutan en el servidor J2EE.
- Software de la capa del sistema de información (*Enterprise information system-tier*) que se ejecuta en el servidor EIS.

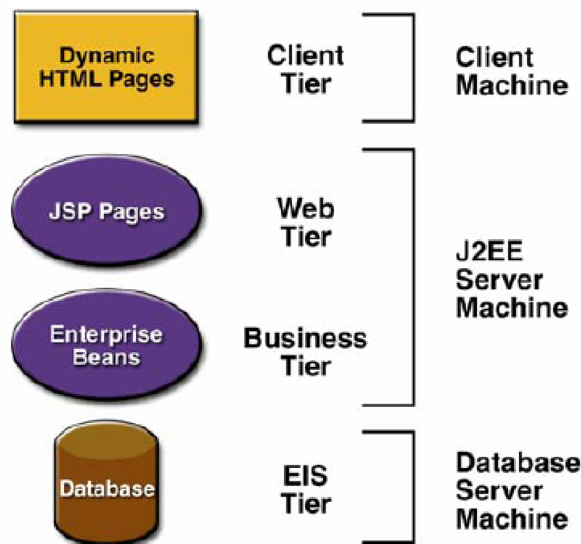


Figura 64 Aplicación J2EE

Las aplicaciones J2EE se suelen considerar de tres capas porque se distribuyen en tres lugares: máquina del cliente, máquina del servidor J2EE y la base de datos (*back end*).

11.6.2. Componentes J2EE

Un componente J2EE es una unidad de software funcional auto-contenido que se une a una aplicación J2EE con sus clases y ficheros, y que se comunica con otros componentes. La especificación del estándar J2EE define los siguientes componentes:

- Aplicaciones cliente y Applets son componentes que se ejecutan en el lado del cliente.
- Servlets y JavaServer Pages (JSP) son componentes Web que se ejecutan en el servidor.
- Enterprise JavaBeans (EJB) son componentes de negocio que se ejecutan en el servidor.

11.7. Aplicaciones Web con tecnologías Java.

Java ofrece las siguientes tecnologías para el desarrollo de aplicaciones Web:

Java Servlet: es una clase Java que procesa peticiones y construye respuestas dinámicamente. Los servlets son muy apropiados para servicios Web y funciones de control.

Java Server Pages (JSP): son documentos basados en texto que se ejecutan como servlets pero que ofrecen un enfoque más natural para la creación de contenido estático. Las JSP se usan para la generación de texto de marcado como HTML o XML.

JavaServer Pages Standard Tag Library (JSTL): es una librería de etiquetas que agrupa las funcionalidades más comunes para la creación de JSP.

JavaServer Faces (JSF): es un *framework* (patrón de desarrollo) basado en el modelo MVC. Permite que las aplicaciones Web gestionen la complejidad de la interfaz de usuario en el servidor, de modo que el desarrollador se centra en el código de la aplicación.

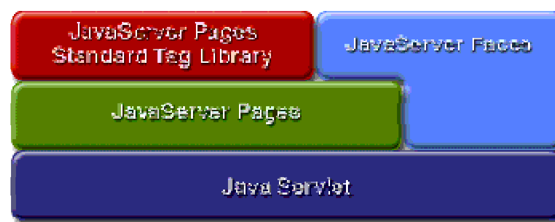


Figura 65 Tecnologías Web

Los servlets son la base de todas las aplicaciones Web. Cada tecnología añade un nivel de abstracción que hace que el desarrollo de aplicaciones Web sea más rápido y robusto.

11.7.1. Java Servlets

Definición: es una clase Java que se usa para extender las capacidades de los servidores que albergan aplicaciones a través de peticiones-respuestas (*request* - *response*).

Usos principales:

- Procesar y almacenar datos enviados desde un formulario HTTP.
- Proveer contenido dinámico (ej. accediendo a una BD)
- Administrar sesiones para las peticiones HTTP.

Ciclo de vida: cuando una petición se mapea a un servlet, el contenedor en el que se encuentra dicho servlet realiza los siguientes pasos:

- Si no existe una instancia del servlet
    - Se carga la clase del servlet
    - Se crea una instancia de la clase
    - Se inicializa la instancia llamando al método *init*.
- Se invoca al método *service*, pasando los objetos de petición y respuesta.
- Si el contenedor necesita borrar el servlet, se llama al método *destroy*.

Ejemplo de servlet.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet{

 public void doGet(HttpServletRequest request,HttpServletResponse response)
                  throws ServletException, IOException {

    PrintWriter out = response.getWriter();

    out.println("Hello World");
  }
}
```

11.7.2. JavaServer Pages (JSP)

Definición: es un documento de texto que contiene dos tipos de texto: datos estáticos (HTML, XML, etc.) y elementos JSP que construyen contenido dinámico.

Ciclo de vida: el servicio de una página JSP se procesa con un servlet. Cuando una petición se mapea a una JSP, el contenedor Web primero comprueba si el servlet de la página es más antiguo. Si esto ocurre, el contenedor traduce la página JSP en una clase servlet y la compila. Este proceso se genera automáticamente.

Ejemplo de página JSP.

```
<%@page contentType="text/html"%>
<html>
        <head>
<title>
Hello JSP
</title>
</head>
         <body>
                <h1>Hello World</h1>
                 The time is <%= new java.util.Date() %>
         </body>
 </html>
```

11.7.3. JavaServer Pages Standard Tag Library (JSTL)

Definición: es una librería de etiquetas que reúne la funcionalidad más común para las páginas JSP.

Librerías:
- **Core**: etiquetas básicas para iteraciones, instrucciones condicionales y entrada/salida.
- **XML**: procesamiento de documentos XML.
- **Internacionalización**: formato I18N.
- **SQL**: etiquetas para acceso a base de datos.

Ejemplo de página JSP con JSTL.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
	<body>
		<c:out value="Hello world!"/>
	</body>
</html>
```

11.7.4. JavaServer Faces (JSF)

Definición: es un framework de componentes de interfaz de usuario (*user interface*, UI) del lado del servidor (*server-side*) para aplicaciones Web basadas en Java.

Los principales componentes de JavaServer Faces son:
- Un API para representar componentes UI y manejar sus estados, eventos, conversión de datos, navegación de páginas, validación del lado del servidor e internacionalización, proporcionando además extensibilidad para todas estas características.
- Dos librerías de etiquetas propias para definir componentes UI dentro de páginas JSP y para conectar componentes a objetos del lado del servidor.

Beneficios
Uno de los mayores beneficios de la tecnología JavaServer Faces es la posibilidad de separar claramente el modelo y la presentación. Las aplicaciones Web construidas con tecnología JSP logran esta separación en parte.
La tecnología JSF incluye una librería de etiquetas propia para representar componentes en páginas JSP, pero esto no impide el uso de otras tecnologías para la presentación de la capa gracias a que el API de JSF está basado directamente en el API de Java Servlet.
Por último, JavaServer Faces proporciona una arquitectura para la administración del estado, procesamiento de datos de los componentes, validación de la entrada del usuario y manejo de eventos.

Aplicaciones Web con JavaServer Faces

Una aplicación JSF es similar a cualquier otra aplicación Web basada en Java. Una aplicación típica contiene los siguientes componentes:

- Componentes JavaBean que contienen la funcionalidad y datos de la aplicación.
- *Listeners* para eventos (*event listeners*)
- Páginas (ej. páginas JSP)
- Clases auxiliares del lado del servidor (ej. *beans* de acceso a base de datos)

Además, las aplicaciones con tecnología JSF también tienen:

- Librería de etiquetas para la representación de componentes UI.
- Librería de etiquetas para representar manejadores de eventos, validadores, y otras acciones.
- Componentes UI representados como objetos con estado en el servidor.
- *Beans* de soporte (*backing beans*), que definen propiedades y funciones para los componentes UI.
- Validadores, conversores, *listeners* de eventos, y manejadores de eventos.
- Fichero de configuración de recursos de la aplicación.

La librería de etiquetas de componentes evita la necesidad de escribir código directamente en HTML u otro lenguaje de marcado. La librería de etiquetas principal (*core tag library*) facilita el registro de eventos, validadores y otras acciones sobre los componentes.

Ejemplo de Aplicación usando JavaServer Faces

El desarrollo de una aplicación JSF normalmente requieres los siguientes pasos:

1. Crear las páginas usando la librería de etiquetas principal (*core*) y de componentes UI. Esta tarea consiste en el diseño de la página usando componentes UI, asociando estos componentes a beans y añadiendo otras etiquetas principales (*core tags*)

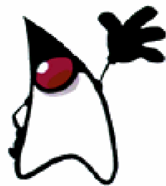


Figura 66. Ejemplo de apliación

Ejemplo de página: `greeting.jsp`

```
<HTML>
    <HEAD> <title>Hello</title> </HEAD>
    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
    <body bgcolor="white">
    <f:view>
        <h:form id="helloForm" >
            <h2>Hi. My name is Duke. I'm thinking of a number from
            <h:outputText value="#{UserNumberBean.minimum}"/> to
            <h:outputText value="#{UserNumberBean.maximum}"/>.
            Can you guess it?</h2>
            <h:graphicImage id="waveImg" url="/wave.med.gif" />
            <h:inputText id="userNo" value="#{UserNumberBean.userNumber}">
                <f:validateLongRange
                    minimum="#{UserNumberBean.minimum}"
                    maximum="#{UserNumberBean.maximum}" />
            </h:inputText>
            <h:commandButton id="submit" action="success" value="Submit"/> <p>
            <h:message style="color: red;
                font-family: 'New Century Schoolbook', serif;
                font-style: oblique;
                text-decoration: overline"
                id="errors1"
                for="userNo"/>
        </h:form>
    </f:view>
</HTML
```

La etiqueta `form` representa un formulario de entrada que permite al usuario introducir datos y enviarlos al servidor, normalmente pulsando un botón.

La etiqueta `outputText` muestra el valor de dicha etiqueta. Los atributos `value` de estas etiquetas obtienen las propiedades `minimum` y `maximum` de `UserNumberBean`, que se usan para referenciar datos almacenados en otros objetos, como por ejemplos *beans*.

La etiqueta `inputText` es un campo de texto. El atributo `value` asocia el valor del componente `userNo` a la propiedad del *bean* `UserNumberBean.userNumber`, que almacena los datos introducidos en el campo de texto.

La etiqueta `commandButton` representa un botón usado para enviar los datos introducidos en el campo de texto. El atributo `action` especifica una salida que ayuda al mecanismo de navegación a decidir cual es la próxima página que tiene que abrir.

La etiqueta `message` muestra un mensaje de error si los datos introducidos en el campo no cumplen con las reglas especificadas en la implementación de `LongRangeValidator`. Anidando esta validación dentro de una etiqueta de un componente se registra dicho validador en el componente. De este modo se comprueba si los datos están dentro de un determinado rango, definido por las propiedades `minimum` y `maximum` de `UserNumberBean`, usando las expresiones de asociación de valor (*value-binding expressions*) `#{UserNumberBean.minimum}` y `#{UserNumberBean.maximum}`.

2. Definir la navegación de páginas en el fichero de configuración de recursos.

La navegación por la aplicación se define en el fichero de configuración de recursos de la aplicación usando un sistema basado en reglas.

Ejemplo:

```
<navigation-rule>
    <from-view-id>/greeting.jsp</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/response.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Cada regla de navegación (`navigation-rule`) define como se llega de una página (especificada en `from-view-id`) a las otras páginas de la aplicación. Puede haber varios elementos `navigation-case`, cada uno de los cuales define la siguiente página a abrir (definida por `to-view-id`) según el valor de la salida (`from-output`).

3. Desarrollar los *beans* de soporte.

Una aplicación JSF lleva unido un *bean* de soporte con cada página en la aplicación. Un *bean* de soporte define las propiedades y métodos que se asocian con los componentes UI usados en la página. Cada propiedad del *bean* está asociada a una instancia de un componente o a su valor. Un *bean* de soporte también puede definir un conjunto de métodos que realicen funciones para el componente, como validación de datos o manejo de eventos.

El valor de un componente se asocia a una propiedad de un *bean* a través de la etiqueta `value` que referencia dicha propiedad. De la misma forma, una instancia del componente se asocia a una propiedad del *bean* referenciando en el atributo `binding` de la etiqueta del componente a la propiedad.

Ejemplo de *bean* de soporte:

```
Integer userNumber = null;
...
public void setUserNumber(Integer user_number) {
   userNumber = user_number;
}
public Integer getUserNumber() {
   return userNumber;
}
public String getResponse() {
   if(userNumber != null && userNumber.compareTo(randomInt) == 0) {
        return "Yay! You got it!";
   } else {
        return "Sorry, "+userNumber+" is incorrect.";
   }
}
```

Este *bean* de soporte es como cualquier otro *bean*: tiene un conjunto de métodos de acceso y un campo de datos privado. JSF convierte automáticamente los datos al tipo especificado en la propiedad *bean*.

4. Añadir las declaraciones de los *beans* administrados en el fichero de configuración de recursos de la aplicación.

Después de definir los *beans* de soporte que se usarán en la aplicación, se tienen que configurar en el fichero de configuración de recursos de la aplicación de modo que la implementación de JSF pueda automáticamente crear nuevas instancias de estos *beans* cuando sean necesarios.

Ejemplo de declaración de *bean*:

```
<managed-bean>
    <managed-bean-name>UserNumberBean</managed-bean-name>
    <managed-bean-class>
        guessNumber.UserNumberBean
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>minimum</property-name>
        <property-class>long</property-class>
        <value>0</value>
    </managed-property>
    <managed-property>
        <property-name>maximum</property-name>
        <property-class>long</property-class>
        <value>10</value>
    </managed-property>
</managed-bean>
```

La etiqueta outputText en la página JSP (greeting.jsp) asocia el valor de este componente a la propiedad minimum de UserNumberBean, usando la expresión:

```
<h:outputText value="#{UserNumberBean.minimum}"/>
```

La implementación de JavaServer Faces procesa este fichero al inicio de la ejecución de la aplicación. Cuando `UserNumberBean` se referencia por primera vez en la página, JSF lo inicializa y lo almacena en el ámbito de la sesión si no existe otra instancia. De esta forma, el *bean* está disponible para todas las páginas de la aplicación.

11.8. Qué es XML.

XML es una tecnología en realidad muy sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores.

XML, con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, más avanzada, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. Así pues, el XML juega un papel importantísimo en este mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable y fácil. Además, XML permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

XML no es un lenguaje, sino varios lenguajes, no es una sintaxis, sino varias y no es una manera totalmente nueva de trabajar, sino una manera más refinada que permitirá que todas las anteriores se puedan comunicar entre si sin problemas, ya que los datos cobran sentido.

11.8.1. Historia del XML.

El XML proviene de un lenguaje que inventó IBM allá por los años 70. El lenguaje de IBM se llama GML (General Markup Language) y surgió por la necesidad que tenían en la empresa de almacenar grandes cantidades de información de temas diversos.

Imaginar por un momento la cantidad de documentación que generaría IBM sobre todas las áreas en las que trabajaba e investigaba, y la cantidad de información que habrá generado hasta hoy. Así pues, necesitaban una manera de guardar la información y los expertos de IBM se inventaron GML, un lenguaje con el que poder clasificarlo todo y escribir cualquier documento para que se pueda luego procesar adecuadamente.

Este lenguaje gustó mucho a la gente de ISO, una entidad que se encarga de normalizar cuantas cosas podáis imaginar para los procesos del mundo actual, de modo que allá por el 86 trabajaron para normalizar el lenguaje, creando el SGML, que no era más que el GML pero estándar (Standar en inglés).

SGML es un lenguaje muy trabajado, capaz de adaptarse a un gran abanico de problemas y a partir de él se han creado los siguientes sistemas para almacenar información.

Por el año 89, para el ámbito de la red Internet, un usuario que había conocido el lenguaje de etiquetas (Markup) y los hiperenlaces creo un nuevo lenguaje llamado HTML, que fue utilizado para un nuevo servicio de Internet, la Web. Este lenguaje fue adoptado rápidamente por la comunidad y varias organizaciones comerciales crearon sus propios visores de HTML y riñeron entre ellos para hacer el visor más avanzado, inventándose etiquetas como su propia voluntad les decía. Desde el 96 hasta hoy una entidad llamada W3C ha tratado de poner orden en el HTML y establecer sus reglas y etiquetas para que sea un estándar. Sin embargo el HTML creció de una manera descontrolada y no cumplió todos los problemas que planteaba la sociedad global de Internet.

El mismo W3C en el 98 empezó y continúa, en el desarrollo de XML (Extended Markup Language). En este lenguaje se ha pensado mucho más y muchas personas con grandes conocimientos en la materia están trabajando todavía en su gestación. Pretendían solucionar las carencias del HTML en lo que se respecta al tratamiento de la información. Problemas del HTML como:

- El contenido se mezcla con los estilos que se le quieren aplicar.
- No permite compartir información con todos los dispositivos, como pueden ser ordenadores o teléfonos móviles.
- La presentación en pantalla depende del visor que se utilice.

Imagínese, una persona que conoce el HTML y lo difícil que puede llegar a ser entender su código, que tuviese que procesarlo para extraer datos que necesite en otras aplicaciones. Sería muy difícil saber dónde está realmente la información que busca, siempre mezclada entre etiquetas <FONT>, <TABLE>, <TD>, etc. Esto es una mala gestión de la información y el XML la soluciona.

11.8.2. Sintaxis del XML.

Dicen que el XML es un 10% del SGML y de verdad lo es, porque en realidad las normas que tiene son muy simples. Se escribe en un documento de texto ASCII, igual que el HTML y en la cabecera del documento se tiene que poner el texto

```
<?xml version="1.0"?>
```

En el resto del documento se deben escribir etiquetas como las de HTML, las etiquetas que nosotros queramos, por eso el lenguaje se llama XML, lenguaje de etiquetas extendido. Las etiquetas se escriben anidadas, unas dentro de otras.

```
<ETIQ1>...<ETIQ2>...</ETIQ2>...</ETIQ1>
```

Cualquier etiqueta puede tener atributos. Le podemos poner los atributos que queramos.

```
<ETIQ atributo1="valor1" atributo2="valor2"...>
```

Los comentarios de XML se escriben igual que los de HTML.

```
<!-- Comentario -->
```

Y esto es todo lo que es el lenguaje XML en sí, aunque tenemos que tener en cuenta que el XML tiene muchos otros lenguajes y tecnologías trabajando alrededor de él. Sin embargo, no cabe duda que la sintaxis XML es realmente reducida y sencilla.
Para definir qué etiquetas y atributos debemos utilizar al escribir en XML tenemos que fijarnos en la manera de guardar la información de una forma estructurada y ordenada. Por ejemplo, si deseamos guardar la información relacionada con una película en un documento XML podríamos utilizar un esquema con las siguientes etiquetas.

```
<?xml version="1.0"?> <PELICULA nombre="El Padrino" año=1985> <PERSONAL>
<DIRECTOR nombre="Georgie Lucar"/> <INTERPRETE nombre="Marlon Brando" interpreta-
a="Don Corleone"/> <INTERPRETE nombre="Al Pacino" interpreta-a="Michael
Corleone"/> </PERSONAL> <ARGUMENTO descripción="Pelicula de mafias sicilianas en
Estados Unidos"/> </PELICULA>
```

11.8.3. Contenidos: DTD o XML Schema

Un documento XML puede contener muchos tipos de información. Es decir, pueden haber muchos lenguajes escritos en XML para cualquier colectivo de usuarios.

Como vemos, se pueden crear infinitos lenguajes a partir del XML. Para especificar cada uno de los usos de XML, o lo que es lo mismo, para especificar cada uno de los sublenguajes que podemos crear a partir de XML, se utilizan unos lenguajes propios.

Son unos lenguajes que sirven para definir otros lenguajes, es decir, son metalenguajes. Los definen especificando qué etiquetas podemos o debemos encontrarnos en los documentos HTML, en qué orden, dentro de qué otras, además de especificar los atributos que pueden o deben tener cada una de las etiquetas.

Hay dos metalenguajes con los que definir los lenguajes que podemos obtener a partir de XML, el DTD y el XML Schema.

El DTD, Definition Type Document, tiene una sintaxis especial, distinta de la de XML, que es sencilla, aunque un poco rara si nunca hemos visto un documento similar.

Para evitar el DTD, que tiene una sintaxis muy especial, se intentó encontrar una manera de escribir en XML la definición de otro lenguaje XML. Se definió entonces el lenguaje XML Schema y funciona bien, aunque puede llegar a ser un poco más complicado que especificarlo en DTD. Simplemente nos ahorramos de aprender un nuevo lenguaje con su sintaxis particular.

11.8.4. Diseño: CSS o XSL.

Para cada documento XML que se desee presentar en pantalla formateado de la manera que deseemos se tiene que escribir una hoja de estilos o similar.

También tenemos dos posibles lenguajes con los que formatear los textos de un documento XML para poder verlo por pantalla. La primera posibilidad es el CSS y la segunda, el XSL, bastante más avanzada.

11.8.5. Programación: SAX o DOM.

Si queremos realizar acciones con nuestros datos escritos en XML tenemos también mucho camino ya implementado. El W3C ha especificado dos mecanismos para acceder a documentos XML y trabajar con ellos. Se tratan simplemente de unas normas que indican a los desarrolladores la manera de acceder a los documentos. Estas normas incluyen una jerarquía de objetos que tienen unos métodos y atributos con los que tendremos que trabajar y que nos simplificarán las tareas relativas al recorrido y acceso a las partes del documento.

Estos dos mecanismos se denominan **SAX y DOM**. SAX se utiliza para hacer un recorrido secuencial de los elementos del documento XLM y DOM implica la creación de un árbol en memoria que contiene el documento XML, y con él en memoria podemos hacer cualquier tipo de recorrido y acciones con los elementos que queramos.

11.8.5.1. El API SAX

SAX define un API para un analizador de archivos *XML* basado en eventos. Estar "basado en eventos" significa que el analizador lee un documento *XML* desde el principio hasta el final, y cada vez que reconoce una sintaxis de construcción, se lo notifica a la aplicación que lo está ejecutando. SAX notifica a la aplicación llamando a los métodos del interface *ContentHandler*. Por ejemplo, cuando el analizador encuentra un símbolo ("<"), llama al método *startElement*; cuando encuentra caracteres de datos, llama al método *characters*; y cuando encuentra un símbolo ("</"), llama al método *endElement*, etc.

Cuando comenzamos a pensar en nuestro proyecto, se nos planteo la duda de que API utilizar para la lectura de *XML's*, teníamos dos opciones: SAX o DOM. DOM es un conjunto de interfaces para construir una representación de objeto, en forma de árbol, de un documento XML analizado, al ser más complicado que SAX, y debido a que los documentos de configuración que íbamos a crear eran sencillos y no muy extensos, nos decidimos por este último.

11.8.5.2. El API JDOM

JDOM es un API para leer, crear y manipular documentos XML de una manera sencilla y muy intuitiva para cualquier programador en Java, en contra de otras APIs tales como DOM y SAX, las cuales se idearon sin pensar en ningún lenguaje en concreto, de ahí que resulte un poco incomoda su utilización. Precisamente esta comodidad que ofrece JDOM frente a SAX o DOM, es la que nos ha llevado a utilizarlo en nuestro proyecto.

El *xml* se forma con objetos *Element*, que representan a los elementos de un *xml* (*cada tag* que se abre y se cierra es un elemento). Para añadir un elemento dentro de otro elemento se utiliza el método *addContent (Element element)*. Para añadir un atributo a un elemento se utiliza el método *setAttribute (Attribute atributte)*, de donde podemos deducir que la clase *Attribute* representa a un atributo de un elemento del *xml*. Por último tenemos el método *setText (String text)*, que está en la clase *Element* y sirve para añadir un texto al elemento. Estos son los métodos y clases básicas, con ellos vamos creando una especie de árbol de elementos que representa el documento, y que se convierte fácilmente en un Archivo *xml.*

12. Librerías utilizadas

A continuación mostramos una tabla con las librerías utilizadas para la implementación de la aplicación, con una breve descripción de la misma y la utilización dentro de nuestro proyecto.

Nombre	*Versión*	*Descripción*	*Módulo de Uso*
antlr	2.7.5	ANTLR: ANother Tool for Language Recognition (Otra herramienta para el Reconocimiento del Idioma). Proporciona un framework para construir reconocedores, compiladores, y traductores de las descripciones gramaticales que contienen Java, C + +, o C#.	Necesaria para las librerías groovy-all e hibernate
bsf	2.3.0	BSF: Bean Scripting Framework es un conjunto de clases java que permite escribir JSPs en otros lenguajes distintos de java proporcionando acceso a la librería de clases java.	Necesaria para corregir los exámenes
cglib-nodep	2.1_3	CGLIB es una poderosa librería de generación de código de alto rendimiento y calidad. Es usada para extender clases Java e implementa interfaces en tiempo de ejecución.	Necesaria para hibernate
commons-beanutils	0	Librería de Apache que facilita el uso de beans de manera dinámica (sin compilado).	Todos
commons-collections	0	Librería de Apache que extiende o aumenta la librería de Colecciones de Java.	Todos
commons-dbcp	1.2	Librería de Apache que aporta servicios de conexión a la base de datos.	Módulo de persistencia
commons-digester	0	Librería de Apache que se utiliza para mapear XML a objetos Java	Módulo de la interfaz gráfica
commons-discovery	0.2	Librería de Apache que proporciona facilidades para instanciar clases en general, y para la gestión del ciclo de vida da clases singleton (factoría)	Necesario para el patrón factoría
commons-logging	0	Librería de Apache para la generación de logs	Todos
commons-pool	1.2	Librería de Apache que proporciona una reunión de objetos.	Todos
dom4j	1.6.1	Librería que ofrece soporte para API's de procesamiento de XML, en nuestro caso para SAX.	Módulo de la interfaz gráfica
ehcache	1.1	Ehcache es generalmente para java cache distribuidas para caching de propósito general, J2EE y contenedores ligeros.	Modulo de la persistencia
groovy-all	1.0-jsr-04	Librería que contiene todas las clases de groovy utilizadas para realizar la corrección de exámenes.	Necesaria para corregir los exámenes
hibernate	3.1.3	Librería de Hibernate	Módulo de la persistencia
hsqldb	1.8.0.1	Librería de HipersonicSQLDB	Módulo de la persistencia
jdom	0	Librería que proporciona una solución completa, basada en java para tener acceso, manipulación, y salida XML para código java.	Módulo de la interfaz gráfica

Nombre	***Versión***	***Descripción***	***Módulo de Uso***
joda-time	1.2.1	Librería que proporciona un reemplazo (suplente) de calidad para la fecha Java y las clases time.	Todos
jsf-api	0	Librería que contiene las clases del API javax.faces.*	Módulo de la interfaz gráfica
jsf-impl	0	Librería que contiene la implementación de las clases JavaServer Faces.	Módulo de la interfaz gráfica
jstl	0	Proyecto de Apache que contiene un repositorio de librerías de etiqueta JSP de encargo y proyectos asociados.	Módulo de la interfaz gráfica
jta	1.0.1B	Librería para la Java Transaction API	Módulo de la persistencia
log4j	1.2.9	Proyecto de apache para la generación de logs	Todos
spring	1.2.7	Librería de Spring.	Módulo de la persistencia
standard	0	Librería de Apache de Tag Libraries	Módulo de la interfaz gráfica

Tabla 13.1 Librerías utilizadas

13. Gestión de configuración

A continuación se detallan los diferentes puntos que describen la planificación que se ha llevado a lo largo del proyecto.

13.1. Modelo de proceso

Hemos seguido un modelo de proceso unificado de desarrollo, también conocido como RUP (*Rational Unified Process*). Caracteristicas:

- Modelo basado en componentes software interconexionados a través de interfaces bien definidas.
- Muy ligado a UML.
- Dirigido por casos de uso.
- Centrado en la arquitecturas
- Iterativo e incremental.

Está formado por cinco flujos de trabajo que se iteran:

- Requisitos
- Análisis
- Diseño
- Implementación
- Prueba

13.2. Personal

13.2.1. Participantes

El grupo se compone de cinco miembros, tres alumnos que han desarrollado el proyecto:

- Lourdes Costero Ranz
- Susana Oliva Pérez
- Miguel Ángel Sánchez Fernández

Más dos profesores-directores, que han guiado a los alumnos:

- Baltasar Fernández Manjón
- Iván Martínez Ortiz

13.2.2. Jefe de equipo

Al tratarse de un grupo de tres personas, no ha habido división jerárquica entre los miembros del grupo, aunque si que ha habido un jefe de grupo encargado de ponerse en contacto con los profesores-directores.

13.2.3. Equipo de desarrollo

Descentralizado democrático
-no tiene un jefe permanente
-se nombre un jefe en función de cada tarea
-las decisiones, problemas y enfoques se llevan a consenso del grupo
-la comunicación entre los miembros del equipo es horizontal

13.2.4. Seguimiento y reuniones

El seguimiento y las reuniones entre los tres alumnos y los directores profesores se ha llevado a cabo los miércoles a las 15.30, en el despacho 414 de la cuarta planta de la facultad de informática correspondiente al profesor Baltasar Fernández Manjón o en el despacho 218 de becarios de la segunda planta de la facultad de informática.

Cuando ha sido necesario también se han realizado reuniones virtuales a través del Messenger o mediante Skype.

13.2.5. Comunicación entre el grupo

La comunicación entre los tres alumnos del grupo y los profesores-directores se ha llevado a cabo mediante correos electrónicos, mediante Messenger y mediante Skype.

13.2.6. Gestión de archivos

Mediante SVN Repository

13.3. Hardware necesario

Para el desarrollo de la aplicación se requiere como configuración mínima un Pentium III a 800 Mhz.

13.4. Software necesario

El software requerido para el desarrollo del proyecto es:

- Eclipse 3.1.2
- Tomcat 5.5.17
- JDK 1.5.0_06
- SVN
- Plugin tomcatPluginV31
- Plugin subclipse_1.0.1
- Ant 1.6.5
- Maven 1.0.2
- Librerías mencionadas anteriormente
- Mozilla Firefox
- Together
- Microsoft Office

14. Bibliografía

[1] IMS Question and Test Interoperability Information Model. Version 2.0 Final Specification http://www.imsproject.org/question/qti_v2p0/imsqti_infov2p0.html

[2] IMS Question & Test Interoperability: ASI Best Practice & Implementation Guide - http://www.imsproject.org/question/qtiv1p2/imsqti_asi_bestv1p2.html

[3] IMS Question & Test Interoperability: ASI XML Binding Specification - http://www.imsproject.org/question/qtiv1p2/imsqti_asi_bindv1p2.html

[4] IMS Question and Test Interoperability Overview - http://www.imsproject.org/question/qti_v2p0/imsqti_oviewv2p0.html

[5] IMS Question and Test Interoperability XML Binding - http://www.imsglobal.org/question/qti_v2p0/imsqti_bindv2p0.html#binding_prompt

[6] IMS Question & Test Interoperability Specification http://www.imsproject.org/question/

[7] XDoclet Attribute-Oriented Programming - Tag Reference http://xdoclet.sourceforge.net/xdoclet/tags/hibernate-tags.html

[8] Hibernate and XDoclet - How generate Hibernate mapping files with XDoclet - http://www.downside.ch/hibernate/hibernatecheatsheet-1.4.pdf

[9] Bauer Gaving King C. *Hibernate in Action* - Manning Publications Co. 2005

[10] Hibernate website - http://www.hibernate.org/

[11] Hibernate Reference Documentation version3.1 - http://www.hibernate.org/hib_docs/v3/reference/en/html/

[12] Hibernate - Relational Persistence for Idiomatic Java - http://www.hibernate.org/hib_docs/reference/en/html/

[13] Hibernate Basics - http://www.developer.com/open/article.php/10930_3559931_1

[14] Persistencia de Objetos Java: El Camino hacia Hibernate - http://www.programacion.com/tutorial/hibernate/

[15] Spring Framework - http://www.springframework.org

[16] Introduction to the Spring Framework- http://www.theserverside.com/articles/article.tss?l=SpringFramework

[17] The Spring series - http://www-128.ibm.com/developerworks/views/web/libraryview.jsp?search_by=The+Spring+Series

[18] Object-relation mapping without the container - http://www-128.ibm.com/developerworks/library/j-hibern/?ca=dnt-515

[19] Spring Framework - http://sourceforge.net/projects/springframework

[20] Spring Framework - http://en.wikipedia.org/wiki/Spring_framework

[21] The spring - http://javaboutique.internet.com/tutorials/spring_frame/

[22] Bruce Eckel. *Piensa en Java. Segunda edición.* Pearson Education, S. A. Madrid. 2002

[23] The J2EE 1.4 Tutorial http://java.sun.com/j2ee/1.4/docs/tutorial/doc/

[24] JavaServer Faces HTML Tags - http://www.exadel.com/tutorial/jsf/jsftags-guide.html

[25] Html code tutorial - http://www.htmlcodetutorial.com/forms/

[26] Java Server Faces. Constant Field Values - http://java.sun.com/javaee/javaserverfaces/1.1_01/docs/api/constant-values.html

[27] XSL Transformations (XSLT) - Http://www.w3.org/TR/xslt#local-variables

[28] JSF communication - http://balusc.xs4all.nl/srv/dev-j2p-com.html

[29] The Java EE Tutorial - http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

[30] Core JavaServer Faces by David Geary and Cay Horstmann, Sun Microsystems Press 2004. - http://www.horstmann.com/corejsf/

[31] The Apache MyFaces Project - http://myfaces.apache.org/

[32] JSF Core Tag Library - http://myfaces.apache.org/impl/tlddoc/f/tld-frame.html
Sun Developer Network. Java Technology Forums - http://forum.java.sun.com/index.jspa

[33] The J2EE 1.4 Tutorial - http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html (Tema 17)

[34] JSF for nonbelievers: JSF conversion and validation - http://www-128.ibm.com/developerworks/java/library/j-jsf3/index.html

[35] Yang Shen, Derek , (1999) *Integración de JSF, Spring e Hibernate para crear una Aplicación Web del Mundo Real* - http://www.programacion.com/tutorial/jap_jsfwork/3/

[36] Apache Ant Project - http://ant.apache.org/

[37] Apache Ant - http://es.wikipedia.org/wiki/Ant

[38] Introducción a Ant - http://www.javahispano.org/articles.article.action?id=31

[39] Apache Maven Project - http://maven.apache.org/

[40] Maven - http://es.wikipedia.org/wiki/Maven

[41] Apache Maven Simplifica el Proceso de Construcción - Incluso más que Ant - http://www.programacion.com/java/tutorial/jap_maven/

[42] Ant & Maven http://metaware-inc.wiki.mailxmail.com/AntMaven

[43] HSQL database engine - http://www.hsqldb.org/

[44] HSQLDB User Guide - http://hsqldb.sourceforge.net/web/hsqlDocsFrame.html

[45] Eclipse and HSQLDB - http://www-128.ibm.com/developerworks/opensource/library/os-echsql/

[46] G. Booch, J. Rumbaugh, I. Jacobson. *El lenguaje unificado de modelado*. Addison Wesley iberoamericana. Madrid. 1999

[47] A. Gutiérrez, R. Martínez. *XML a través de ejemplos*. Ra-Ma. 2001.