No se puede mostrar la imagen. Puede que equipo no tenga suficiente memoria para abrir la imagen. Puede que esta imagen esté dañada. Intente volver a insertarla. Si sigue apareciendo no copia, puede que tenga que borrar la imagen y insertarla de nuevo.

UNIVERSIDAD REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2009/2010

Proyecto de Fin de Carrera

Desarrollo de un lenguaje XML para la participación electrónica

Autor: Oscar Pascual Núñez

Tutores: David Ríos Insua

Cesar Alfaro Gimeno

AGRADECIMIENTOS

Quiero agradecer de una manera muy especial a David Ríos y a César Alfaro, por darme la posibilidad de realizar el presente proyecto bajo su tutela; aportándome en todo momento su ayuda.

Por otro lado, no me puedo olvidar del apoyo incondicional que he encontrado en mis padres, en mis hermanos y sobre todo en Angelina, mi novia, y mi hijo Alfonso, por que aun teniendo menos de un año me ha dejado trabajar en el proyecto y así poder terminarlo. Tampoco me puedo olvidar de mis compañeros de clase, que aunque nos veamos poco siempre han estado ahí cuando les he necesitado, porque más que compañeros son grandes amigos.

RESUMEN

Este proyecto fin de carrera es parte de la arquitectura necesaria para realizar una aplicación sobre participación electrónica.

Durante el desarrollo de esta memoria trataremos de describir en que consiste la participación electrónica y las necesidades en la sociedad de la implantación de este mecanismo de decisión, y el camino elegido para poder llevar a cabo este proyecto de participación electrónica. También describiremos la arquitectura de XML y, por qué, éste ha sido el lenguaje elegido para la interacción entre los distintos módulos en los que está dividida la aplicación NAML (Negotiation Analysis Markup Language).

En este Proyecto Fin de Carrera (PFC) el trabajo realizado se ha basado en la obtención de la información que se encuentra en la base de datos de cada módulo y en el tratamiento de esta información para montar un XML que pueda entender el resto de módulos de NAML, y el tratamiento del XML que llegue de otros módulos y la inserción en base de datos. El XML obtenido será enviado a otros módulos de la aplicación para que sea tratado.

El objetivo principal de este proyecto fin de carrera es el intercambio de información entre distintos módulos de la aplicación NAML utilizando un desarrollo en Java para tal fin.

Para ello necesitamos cumplir varios objetivos intermedios:

1. Obtención de la información de cada módulo de la base de datos MySQL.

2. Almacenar esta información en un XML.

3. Validar el XML obtenido contra un esquema XSD.

4. Tratar el XML validado por el esquema.

5. Almacenar la información que contiene el XML en la base de datos MySQL del módulo receptor.

Para poder llegar a tales objetivos se ha decidido desarrollar las clases necesarias mediante el uso del lenguaje de programación Java, en el cual hemos echo uso de bibliotecas open source que nos han ayudado en nuestra tarea. También se han utilizado varias aplicaciones open source.

ÍNDICE

Índice de contenido

AGRADECIMIENTOS i
RESUMEN ii
ÍNDICE iii
Índice de contenido iii
Índice de ilustraciones iv
Índice de ficheros v
INTRODUCCIÓN 1
1. Participación electrónica 1
1.1. Definición 1
1.2.Ventajas de la participación electrónica frente a la participación no electrónica. 2
2. XML <Extensible Markup Language> 4
2.1. Ventajas 4
2.2. Estructura de un XML 5
2.3. Documentos bien formados. 5
2.4. XML Schemas (XSD) 6
DESCRIPCIÓN INFORMÁTICA 7
3. Arquitectura 7
3.1. Módulo de seguridad 9
3.2. Interfaz 9
3.3. Módulo controlador 9
3.4. Módulo de estructuración del problema 9
3.5. Módulo de debate 10
3.6. Módulo de comunicación de preferencias 11
3.7. Módulo de negociación 12
3.8. Módulo de votación 13
4. Software de apoyo 13
4.1. Aplicaciones utilizadas para el desarrollo 14
4.2. Bibliotecas java de soporte 17
5. Desarrollo realizado 19
5.1. Base de datos 19
5.2. Desarrollo Java 25
CONCLUSIONES Y MEJORAS 51
BIBLIOGRAFIA 52

Índice de ilustraciones

Ilustración 1: Diagrama arquitectura NAML 8
Ilustración 2: Eclipse 3.2 15
Ilustración 3: DBDesigner 4 16
Ilustración 4: Tablas base de datos Estructuración 20
Ilustración 5: Tablas base de datos Preferencias 21
Ilustración 6: Tablas base de datos Debate 22
Ilustración 7: Tablas base de datos Negociación 23
Ilustración 8: Tablas base de datos Votación 24
Ilustración 9: Tabla base de datos Controlador 25
Ilustración 10: Clase PSObject.java 26
Ilustración 11: Diagrama de clases del Módulo de Estructuración del problema 30
Ilustración 12: Diagrama de clases del Módulo de Debate 33
Ilustración 13: Diagrama de clases del Módulo de comunicación de preferencias 39
Ilustración 14: Diagrama de clases del Módulo de Negociación 44
Ilustración 15: Diagrama de clases del Módulo de Votación 48
Ilustración 16: Diagrama de clase del Módulo Controlador 50

Índice de ficheros

Fichero 1: log4j.properties 17
Esquema XSD 2: problemStructuring.xsd 29
Esquema XSD 3: dialog.xsd 33
Esquema XSD 4: preference.xsd 39
Esquema XSD 5: negotiation.xsd 43
Esquema XSD 6: Voting.xsd 48
Esquema XSD 7: controller.xsd 50

INTRODUCCIÓN

1. Participación electrónica

1.1. Definición

En la actualidad cada vez está más de moda 'la nube', nombre con el que se empieza a conocer a Internet y las posibilidades que este entorno aporta a la vida diaria. Desde el punto de vista de la política y de la negociación, todavía no se ha conseguido sacar todo el potencial que Internet proporciona a estas actividades.

Uno de los pilares que se tendrían que resolver para poder llevar a cabo la democracia electrónica y que el ciudadano tenga una mayor participación en la política es la ***participación electrónica***. En la participación electrónica, hay que identificar el papel del ciudadano y las reglas para poder llevarla a cabo en condiciones.

De hecho, tenemos varios ejemplos de participación electrónica llevada de forma errónea. algunos de ellos se encuentran en la web change.gov (**"Citizen´s Briefing Book", "ARRA for comment"**), pues no tuvieron en cuenta las reglas o el papel del ciudadano. En el caso de **"Citizen´s Briefing Book"** no se siguió la iniciativa de participación ciudadana, por lo que los resultados pudieron ser falseados por acciones coordinadas de grupos favorables a la medida en la que participen. Por tanto, deberían **evitar el rapto por las élites de la participación electrónica:** *cualquier experiencia a desarrollar no consiste tan sólo en abrir una puerta para la opinión, sino en conseguir que en la misma tomen parte con el mismo peso, todos los grupos sociales afectados*.

Para poder realizar un seguimiento apropiado de las iniciativas de participación ciudadana, no es suficiente con realizar una pregunta y pasar a ver los resultados, sino que hay que realizar una labor de moderación, para lo cual los promotores de la iniciativa deben contar con personal especializado y **establecer unas reglas claras de cual va a ser la línea de actuación durante el desarrollo de las iniciativas desplegadas. Además,** hay que definir que es lo que obtiene el ciudadano a cambio de su participación en estas iniciativas. **La gestión de las expectativas es un aspecto crucial de la Participación Electrónica,** debiéndose, cuando menos, facilitar algún tipo de conclusiones del promotor de la iniciativa. Hay que dejar claro qué se va a hacer con la respuesta de la ciudadanía. En el ejemplo **"ARRA for comment"** se observa negativamente este punto, ya que lo único que tenía el ciudadano para participar era un formulario de opinión, **sin tener conocimiento de la opinión de otros ciudadanos ni dar la posibilidad de un diálogo sobre el tema de la participación**.

En resumen, **no toda la ciudadanía ha de participar en toda iniciativa, pero sí ha de participar aquella ciudadanía que es necesaria.** Además, si se quiere establecer un modelo sostenible de participación electrónica, éste ha de basarse en la **implicación cotidiana de los promotores**, en realizar una adecuada **gestión de las expectativas ciudadanas** que den lugar al compromiso cívico con las mismas y en la **existencia de un diálogo real entre ciudadanos y gobierno**.

Font y Blanco resumen tales características y en qué consiste un proceso de participación ciudadana con la siguiente afirmación: "la gente sólo participará si el proceso participativo es ampliamente visible, si los objetivos de la participación son claros, si tienen certeza de que la participación no implicará una pérdida inútil de tiempo, si prevén que podrán expresar sus opiniones libremente y si perciben que, efectivamente, las autoridades políticas tendrán en cuenta su opinión" (Font y Blanco 2003: p. 36).

1.2.Ventajas de la participación electrónica frente a la participación no electrónica.

La participación electrónica, al tener como medio de interacción ente los distintos participantes de la misma un medio telemático (Tecnologías de la Información y la Comunicación (TIC)), facilita la participación del ciudadano, el intercambio de información y la toma de decisiones de los distintos participantes.

1.2.1 Distribuir y compartir informaciones

Los promotores de la iniciativa tienen la posibilidad de compartir con los ciudadanos, de manera eficiente, gran cantidad de información y de documentación útil para el proceso. Los propios ciudadanos pueden, a su vez, compartir información y documentación que crean necesaria para poder apoyar sus ideas en el diálogo previo a la posible toma de decisiones.

1.2.2 Discusiones de calidad y mejora de la comunicación

De forma general, las aportaciones realizadas por los ciudadanos suelen ser más relevantes y constructivas en la participación electrónica que en la presencial, ya que, debido al escaso tiempo que se tiene en las presenciales, los participantes no pueden preparar o documentar sus intervenciones adecuadamente, mientras que en la participación electrónica, al disponer de más tiempo para poder llevar a cabo los diálogos entre ciudadanos, el tiempo para poder preparar y documentar sus intervenciones es mayor, por lo que son más argumentadas y reflexivas.

Es más, si los procesos de discusión están adecuadamente planificados y con un seguimiento adecuado por parte de los mediadores, es posible obtener debates caracterizados por una atmósfera

constructiva y creativa en la que los participantes comprometidos aportan contribuciones ajustadas y precisas (Lührs et al. 2004: p. 9).

1.2.3 Documentación, transparencia y neutralidad

En la participación electrónica, al encontrarse todas las contribuciones disponibles en formato electrónico, es más fácil la labor de análisis del proceso de discusión y la elaboración de informes y la redacción de conclusiones. Todas las intervenciones de las distintas partes de esta participación quedan registradas y accesibles para el público, por lo que se incrementa la transparencia de información. En el caso de que la mediación de la iniciativa la desarrolle alguna institución independiente y con probada autonomía respecto a las instituciones gubernamentales, es posible garantizar una mayor neutralidad del proceso.

1.2.4 Virtualidad e inclusión: extensión de la participación

La participación electrónica permite la implicación de sectores de la sociedad que en el caso de la presencial no tendrían cabida, como es el caso de los jóvenes y adolescente, de personas con deficiencias, ciudadanos faltos de tiempo, etc. Además gracias a Internet es posible realizar aportaciones a la iniciativa a cualquier hora y desde cualquier lugar, permitiendo extender la participación a sectores de la sociedad que, de otra forma, no podrían hacerlo.

1.3. Participación electrónica en la actualidad.

Hace unos años, el coste para llevar a cabo este tipo de procesos era excesivo debido al alto valor de los recursos y de la logística necesarias para realizarlo, por lo que se desestimaba el poder llevar acabo este tipo de proyectos. En la actualidad debido al abaratamiento de los recursos y al creciente índice de hogares con acceso a Internet, este movimiento está cobrando más fuerza, promoviendo el desarrollo e implantación de sistemas y aplicaciones para poder llevar a cabo este paso hacia la e-democracia.

Entre los beneficios potenciales de la naciente democracia electrónica se incluirían:

- Acercar las decisiones a los ciudadanos.
- Decisiones públicas, tomadas públicamente.
- Eliminación de la apatía.
- Eliminación de la alienación.
- Maximización del potencial del ciudadano.
- Legitimación.

- Transparencia.

2. XML <Extensible Markup Language>

El XML es un metalenguaje de etiquetas desarrollado por el World Wide Web Consortium (W3C). Este metalenguaje permite la definición de gramáticas específicas para otros lenguajes (XHTML,SVG,MathML,...). XML también se puede utilizar para estandarizar el intercambio de información estructurada entre plataformas. Este papel es muy importante actualmente, pues posibilita el intercambio entre plataformas de manera segura, fiable y fácil.

2.1. Ventajas

Los documentos XML aportan ventajas que no aportan otros documentos a la hora del intercambio de datos:

- Es un metalenguaje extensible, por lo que se le pueden añadir etiquetas en cualquier momento sin excesivas complicaciones.
- El analizador es estándar, por lo que no se pierde tiempo desarrollando uno para cada XML, sino que se utiliza alguno de los ya disponibles (xalan, xerces,...).
- Posee una estructura fácil de entender y procesar, facilitando la compatibilidad entre aplicaciones.

2.2. Estructura de un XML

Los XML permiten representar de forma abstracta y reutilizable la información estructurada,por ejemplo con componentes bien definidos, tales que cada componente puede estar compuesto, a su vez, de mas componentes.

La estructura de los XML está definida por una etiqueta inicial, que será la etiqueta padre del documento (etiqueta más externa), la cual contendrá información ya sea en forma de más etiquetas o con valores. Una etiqueta consiste en una marca que señala un trozo de información con un sentido claro y definido. Las etiquetas tienen la forma **`<nombre>`**, donde *nombre* es el nombre del elemento que se está señalando.

2.3. Documentos bien formados.

Los documentos XML bien formados son aquellos que cumplen todas las definiciones básicas de formato y pueden ser analizados por cualquier analizador sintáctico que cumpla con la norma. Para que un documento esté bien formado, este tiene que seguir una estructura jerárquica con respecto a las etiquetas que delimitan sus elementos. Las etiquetas deben estar correctamente

anidadas y los elementos con contenido correctamente cerrados. Solo puede haber un nodo o elemento raíz. Los atributos de los elementos o etiquetas deben ir encerrados entre comillas dobles o simples.

El XML es Case-Sensitive (Sensible a mayúsculas y minúsculas),por ejemplo, los analizadores de estos documentos no darían por valido un elemento que abriera <forum> y cerrara </Forum>, ya que para el analizador son etiquetas de dos nodos distintos.

Se muestra, a continuación, un ejemplo para entender la estructura de un XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<forum id="4113bafd27f6e91f0127f6fef78f0002">
	<name>PROYECTO PILOTO DE PRESUPUESTOS PARTICIPATIVOS</name>
	<description>Descripcion del proyecto piloto</description>
	<categories num_categories="2">
		<category id="4113bafd27f7a9180127f7aa90f10001">
			<name>Attributes</name>
			<description>Attribute's description</description>
			<topics num_topics="1">
				<topic id="4113bafd27f7a9180127f7aa92680006">
					<title>Problem's attributes</title>
					<description>Description A</description>
					<messages num_messages="1">
						<message id="80007">
							<title>titulo 2</title>
							<body>cuerpo del mensaje</body>
							<date>2010-04-13</date>
							<participant id="cpg"/>
						</message>
					</messages>
					<participant id="cpg"/>
				</topic>
			</topics>
		</category>
	</categories>
	<participants num_participants="3">
		<participant id="admin">
			<name>admin</name>
			<surname>admin</surname>
			<DNI>01234567G</DNI>
		</participant>
		<participant id="cesar">
			<name>Cesar</name>
			<surname>Cesar</surname>
			<DNI>12345678W</DNI>
		</participant>
		<participant id="david">
			<name>David</name>
			<surname>David</surname>
			<DNI>12457896C</DNI>
		</participant>
	</participants>
</forum>
```

Como se puede apreciar este XML esta bien formado, pues tiene un único elemento raíz (forum), el cual tiene cuatro elementos internos (name, description, categories y participants), a su

vez estos elementos pueden tener hijos o datos, en el caso de name y description su contenido son datos, mientras que en los casos de categories y participants lo que incluyen son mas nodos o elementos.

2.4. XML Schemas (XSD)

XSD es un lenguaje de esquema utilizado para describir la estructura y restricciones de los contenidos de un documento XML de forma precisa, facilitando una percepción del tipo de documento con un nivel de abstracción elevado. En pocas palabras XSD define qué elementos puede contener un documento XML, cómo están organizados y que atributos y de qué tipo pueden ser dichos elementos.

Los distintos esquemas XSD que se han desarrollado durante este proyecto fin de carrera estan incluidos en la descripción informática en el punto de desarrollo realizado. Cada módulo de la aplicación tiene su propio esquema XSD.

DESCRIPCIÓN INFORMÁTICA

El proyecto fin de carrera que nos ocupa forma parte de un proyecto del Departamento de Estadística e Investigación Operativa (DEIO) que está realizando una aplicación de apoyo a la participación electrónica.

A continuación pasamos a describir la arquitectura de esta aplicación.

3. Arquitectura

El objetivo de esta arquitectura es aportar una plataforma tecnológica con herramientas innovadoras y servicios que faciliten la participación electrónica y fomentar el uso de las tecnologías de la información y comunicación, posibilitando el aumento de satisfacción ciudadana ademas de transparencia en el proceso de toma de decisiones. Esta arquitectura es genérica, ya que permite eliminar o repetir etapas para adaptar el sistema a cada proceso participativo específico. La metodología implementada descompone el proceso de elaboración participativa en varias fases con fines específicos. La preparación del proceso, la discusión y la consolidación, la obtención de preferencias de los participantes, la negociación, la votación y posteriormente el asentimiento.

Describimos a continuación la arquitectura dividida en módulos, véase la ilustración 3.1, que apoya cada una de estas fases. En este contexto se distinguen tres perfiles de usuario básico.

- El propietario del problema, es la entidad que tiene por objetivo resolver un problema de proceso participativo, estructura el problema y lo publica para que el ciudadano pueda acceder al mismo. Por lo general, sería un ayuntamiento representado por su alcalde.

- El participante, que hace aportaciones a la toma de decisiones del proceso participativo. Estas entradas son sus opiniones y preferencias sobre los proyectos que conviene incluir. Este perfil debería ser utilizado por los ciudadanos, representados por una muestra de ellos o por sus representantes electos.

- El administrador, que se encarga del desarrollo técnico del proceso: apoyando al propietario del problema en su estructuración, concediendo derechos de acceso a los participantes, definiendo las ventanas de tiempo de cada fase y velando por la seguridad de todo el proceso participativo.

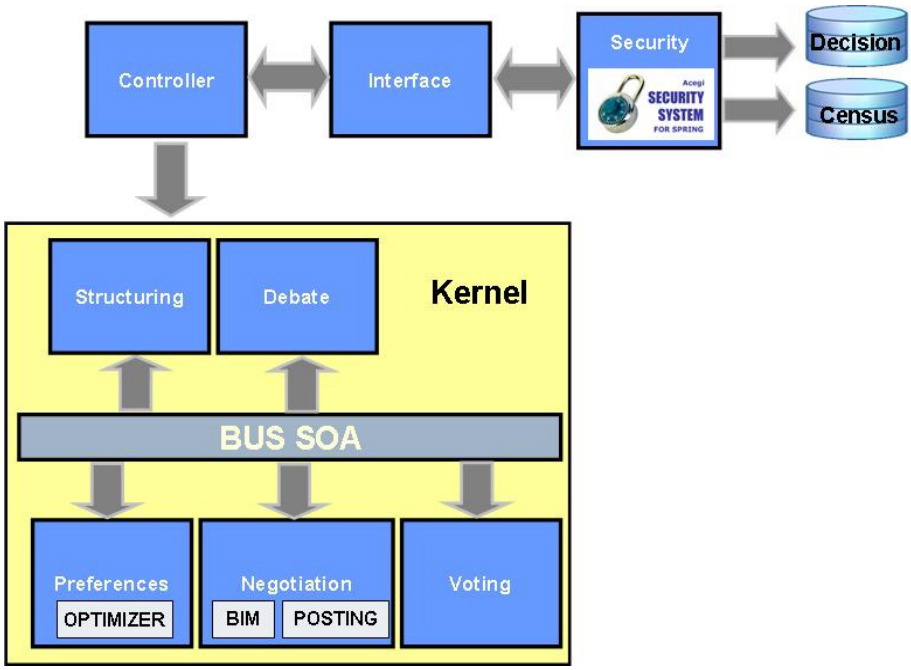

Ilustración 1: Diagrama arquitectura NAML

La arquitectura guía al administrador, en nombre del propietario del problema, y a los participantes en el proceso de elaboración del proceso participativo, siguiendo un protocolo, basado en las fases que describiremos después. En cualquier momento, la acción que cada agente puede llevar acabo dependerá de su acción anterior en el sistema y de las acciones del resto de agentes. A modo de ejemplo, ninguno de los agentes puede comunicar sus preferencias al sistema hasta que el administrador finalice la fase de preparación del proceso. El estado del proceso en cada momento viene definido por la base de acciones llevadas a cabo por los agentes hasta ese momento.

La arquitectura se ha implementado con Java/J2EE, Open Source, con tecnologías de desarrollo como Spring, JSF, Hibernate, Acegi, Maven, Subversion, etc. Desde el punto de vista de tecnologías Web, la arquitectura ha sido configurada como una Arquitectura Orientada a Servicios (SOA). Así, es posible adoptar la filosofía de Software As a Service (SaaS), que elimina la necesidad de instalar y ejecutar aplicaciones en el propio ordenador del cliente, ya que la aplicación se puede alojar como un servicio prestado a los clientes a través de Internet. SaaS, en un principio, permite reducir el gasto por adelantado por la compra de software, a través de un coste menor, sobre el precio bajo demanda. Por otra parte, la arquitectura tiene la intención de dar un paso más en la evolución de la Web 2.0, ya que con ella los ciudadanos no sólo podrán comunicarse entre ellos, o expresar sus puntos de vista, a través de redes sociales, blog, … sino que también tendrán la oportunidad de participar y decidir, para aumentar la transparencia en la toma de decisiones y, eventualmente, aumentar su satisfacción.

Un sistema de bases de datos relacional permite almacenar y gestionar la información sobre el proceso de elaboración participativa. Otra base de datos contiene la información de todos los

participantes, sus roles, sus permisos, etc... Además, el sistema tiene una interfaz de usuario web.

A continuación se describen los módulos principales incluidos en la arquitectura.

3.1. Módulo de seguridad

El módulo de seguridad es el responsable de verificar qué usuarios están registrados en la aplicación. Para iniciar sesión en el sistema, los usuarios deben autenticarse usando una contraseña. El sistema, utilizando tecnología Acegi, verifica que el usuario está en la base de datos y tiene permisos para participar. Acegi Security es un proyecto de Open Source que permite la autenticación y la autorización global de las aplicaciones empresariales basadas en un framework Spring. A continuación, el sistema asigna el rol que desempeña el participante y lo que puede hacer en el sistema dependiendo de su rol.

3.2. Interfaz

Para facilitar el uso y mitigar la brecha digital, el interfaz es gráfico y simple. Se reúnen todas las condiciones de accesibilidad y cumplen los estándares para hacer el contenido Web accesible a las personas con discapacidad.

3.3. Módulo controlador

La arquitectura incluye un módulo que se encarga de controlar en qué fase se encuentra cada problema y cuando tiene que pasar de un módulo a otro, según haya configurado el propietario del mismo.

3.4. Módulo de estructuración del problema

La arquitectura incluye un módulo que permite estructurar el problema del proceso participativo. Sólo el administrador, en nombre del propietario del problema, puede acceder a este módulo. Suponemos que una lista inicial de proyectos y criterios propuestos son generados por el propietario del problema. Los ciudadanos pueden utilizar la estructura inicial para explorar temas de interés sobre el problema. Este módulo permite:

- Definir y estructurar los criterios o atributos a evaluar en las propuestas de problemas por los participantes. El nombre y la escala de cada criterio que se defina.

- Definir los propósitos del problema, incluyendo sus nombres y una descripción de cada propósito, la especificación de sus costes asociados y la evaluación con respecto a un criterio, es decir, sus niveles en todos los atributos.

- Especificar el proceso disponible, para determinar tanto las restricciones, como las

dependencias entre problemas o la incompatibilidad de los problemas. Por ejemplo, un parking de hospital no puede ser construido hasta que el hospital no sea construido.

Los participantes no tendrán acceso a la información del problema hasta que éste sea publicado por el administrador, permitiendo a los ciudadanos autorizados participar en sus discusiones y formación. Para hacer esto, es necesario que el administrador rellene un formulario especificando las características generales del proceso de elaboración participativa, incluyendo:

- Intervalos de tiempo para el debate y la negociación.
- La fecha límite para la fase de comunicación de preferencias.
- Porcentaje de participantes que tienen que estar de acuerdo con la oferta en la fase de negociación a fin de que la implementación sea una decisión de grupo.
- Intervalo de tiempo en el cual los participantes pueden votar si finalmente se alcanza la fase de votación, así como las reglas de votación para hacerlo. Para seleccionar las reglas de votación el propietario del problema debe introducir varios parámetros tales como el número de proyectos que un participante puede votar, o el número de puntos que se pueden dar a cada proyecto dependiendo de las reglas de votación elegidas.

3.5. Módulo de debate

Cuando el propietario del problema lo ha publicado, los participantes autorizados reciben un correo electrónico con el usuario y la contraseña, que les invita a participar en la formación del proceso. Una vez que el problema ha sido publicado, los ciudadanos pueden participar en la consolidación de la lista final de propósitos a través de un foro supervisado por el equipo técnico. Así, los ciudadanos y los interesados pueden ampliar el conjunto de alternativas proponiendo nuevos proyectos. Los ciudadanos también pueden proponer nuevos atributos que reflejen sus objetivos que no han sido considerados en la fase de preparación del proceso. Este módulo permite a los ciudadanos y a los interesados expresar sus opiniones y discutir a través del envío de mensajes sobre la adición, cambio o eliminación de criterios o proyectos. El módulo permite al administrador en nombre del propietario de problema, posiblemente ayudado por un equipo de analistas de decisión:

- Moderar el foro y definir el nivel de acceso al foro de los participantes.
- Decidir la inclusión de nuevas propuestas de proyectos o criterios y, en caso necesario, eliminar o cambiar alguna de ellas.

3.6. Módulo de comunicación de preferencias

Los ciudadanos son invitados a evaluar de forma privada y confidencial la lista final de propuestas. Los participantes reciben un correo electrónico comunicando que se está a la espera de que comuniquen sus preferencias antes de una fecha límite. El sistema incluye un módulo que permite a los participantes construir sus modelos de preferencias dentro de un plazo determinado. En esa ventana de tiempo, el propietario del problema no puede modificar ninguna información del problema, pero tendrá acceso al porcentaje de participantes que han comunicado sus preferencias.

Este es un módulo muy técnico, aunque está construido de manera amigable. Se supone que cualquier usuario (ciudadano) puede construir su propia función de valor. Sin perder demasiada generalidad, asumimos que las preferencias del usuario pueden ser inicialmente modeladas a través de una función de valor añadido ponderado. Por lo tanto, deben cuantificar sus preferencias respecto a los niveles de cada atributo y dar peso a las funciones de valor añadido, véase (Keeney and Raiffa, 1976) o (French and Rios Insua, 2000) para más detalles. En el módulo de estructuración, el propietario del problema ha especificado las propiedades básicas de los atributos: número de atributos, sus escalas y rango. Se asume que todos los participantes comparten estos atributos para medir los resultados del proyecto. Algunos participantes pueden ignorar algunos de los atributos, por ejemplo, dándoles peso cero.

En este módulo, cada usuario evalúa su función de valor privadamente y comunica ésta al sistema. El sistema guiará a los participantes en sus asignaciones de obtención de preferencias, permitiendo:

- *La evaluación de cada valor del componente de función.* Si el atributo se mide con una escala numérica objetiva, los participantes pueden usar un método directo para evaluar la función de valor respecto los niveles de este atributo. Los participantes deben comunicar primero si el objetivo es minimizar o maximizar ese atributo, asignando sus peores y mejores valores. Más tarde, los participantes deben determinar tres puntos de la función de valor y la función del valor final de este atributo se calcula mediante interpolación lineal.

- *Evaluación de los pesos de la función de valor aditivo*. Se utiliza un método directo para obtener los pesos. Los participantes deben presentar el peso de cada atributo, dando mayor peso a aquellos que consideran más importantes.

- *Evalúa la coherencia*. Una vez que un participante evalúa una función de valor, los valores de los participantes de los proyectos se muestran en una escala de 0 a 100, que se enumeran en orden decreciente de valor. Para validar las preferencias del usuario, los

usuarios pueden cambiar los valores de sus proyectos de preferencia antes de que finalice la fase de preferencias de comunicación de preferencias.

• Guarda la función de valor para propósitos posteriores. Una vez que las preferencias de un participante se evalúan, el sistema puede calcular su evaluación privada de cada cada proceso posible, así como su proceso preferido. Para este propósito, el sistema incluye un submódulo que permite a los usuarios evaluar los procesos según su función de valor y obtener su conjunto de opciones óptimo resolviendo el problema de optimización correspondiente. Para hacer esto, el sistema calcula todos los posibles conjuntos de opciones y devuelve el conjunto con mayor valor de utilidad para ese participante. Una vez finalizada la fase de comunicación de preferencias, el sistema verifica si todos los participantes que han comunicado sus preferencias prefieren el mismo proceso. Si éste no es el caso, el sistema continuará con la siguiente fase para intentar llegar a un acuerdo.

3.7. Módulo de negociación

Este módulo es compatible con una negociación multipartidista denominada como POSTING, véase (Ríos y Ríos Insua, 2008), en el que a los participantes se les permite hacer ofertas en las que proponen proyectos para gastar el proceso y discutir a través de un foro. Los participantes pueden votar a favor o en contra de cada propuesta ofertada. La oferta con mayor porcentaje de aceptación entre los participantes se llevará a cabo si este porcentaje es lo suficientemente alto antes de que finalice el plazo de negociación.

Además, el módulo podría utilizarse si el sistema detecta que una solución está dominada. En este caso, los participantes tienen la opción de renegociar e iniciar las negociaciones de la solución del proceso dominante como punto de desacuerdo, utilizando el método de incrementos equilibrados (BIM) (Ríos y Ríos Insua, 2008).

3.8. Módulo de votación

Como es posible que los participantes pueden no llegar a un acuerdo sobre un proceso, nuestro sistema incluye un módulo de votación, lo que permite el diseño de una sesión de votación, con varias normas de votación disponible y de su ejecución como el conteo de Borda, voto por aprobación, regla de la mayoría o el voto acumulativo.

Los participantes reciben un correo electrónico invitándoles a votar en una ventana de tiempo. Los participantes podrán votar vía web en ese tiempo. Aunque este módulo implementa varias reglas de votación, se recomienda utilizar el voto por aprobación para calcular el proceso ganador. En este caso, los participantes ofrecerán un voto a tantos proyectos como desee. Para facilitar esta

tarea, el sistema presenta los proyectos ordenados de acuerdo a los valores que los participantes comunicaron durante la fase de comunicación de preferencia, si ésta ya ha sido ejecutada. El módulo permite modificar su voto, siempre que no haya terminado la sesión de votación. Una vez que finalice la sesión de votación, los proyectos son ordenados por número de votos. Entonces, el proceso ganador se obtiene como el subconjunto de proyectos que cumplan las restricciones del problema. Para ello, los proyectos, siguiendo el orden de votación anterior, se añaden a la solución de voto siempre que todas las restricciones del problema puedan ser satisfechas. Por último, el módulo envía un correo electrónico a todos los participantes con los resultados.

4. Software de apoyo

En este proyecto fin de carrera se ha trabajado el desarrollo de la aplicación de participación electrónica NAML. Esta aplicación se puede dividir en varias partes definidas y diferenciables (cajas negras) a la hora de trabajar para su desarrollo completo. Estas partes son:

1. Interfaz Web.
2. Comunicación entre los distintos módulos.
3. Creación de Base de datos.
4. Almacenamiento y obtención de la información de los distintos módulos que se envían en un XML.
5. Desarrollo del Controlador.
6. Seguridad.
7. Generación de documentos.

En este proyecto fin de carrera se han desarrollado dos partes, la de creación de base de datos (3) y la de almacenamiento y obtención de la información de los distintos módulos (4).

4.1. Aplicaciones utilizadas para el desarrollo

El software utilizado para la realización de este proyecto incluye MySQL5.2, Eclipse3.2 y DBDesigner 4. Esta sección presenta una breve descripción de dicho software.

4.1.1. MySQL 5.2

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB —desde enero de 2008 una subsidiaria de Sun

Microsystems y, ésta, a su vez, de Oracle Corporation desde abril de 2009— desarrolla MySQL como software libre en un esquema de licenciamiento dual. Por un lado, se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C. Al contrario de proyectos como Apache, donde el software es desarrollado por una comunidad pública y el copyright del código está en poder del autor individual, MySQL es propietario y está patrocinado por una empresa privada, que posee el copyright de la mayor parte del código. Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias privativas, la compañía ofrece soporte y servicios. Para sus operaciones contratan trabajadores alrededor del mundo que colaboran vía Internet. MySQL AB fue fundado por Axmark, Larsson y Widenius.

MySQL es muy utilizado en aplicaciones web, como Drupal o phpBB, en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Su popularidad como aplicación web está muy ligada a PHP que, a menudo, aparece en combinación con MySQL. MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos. En cambio, el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

4.1.2. Eclipse 3.2

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma ha sido usada típicamente para programar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el propio Eclipse).

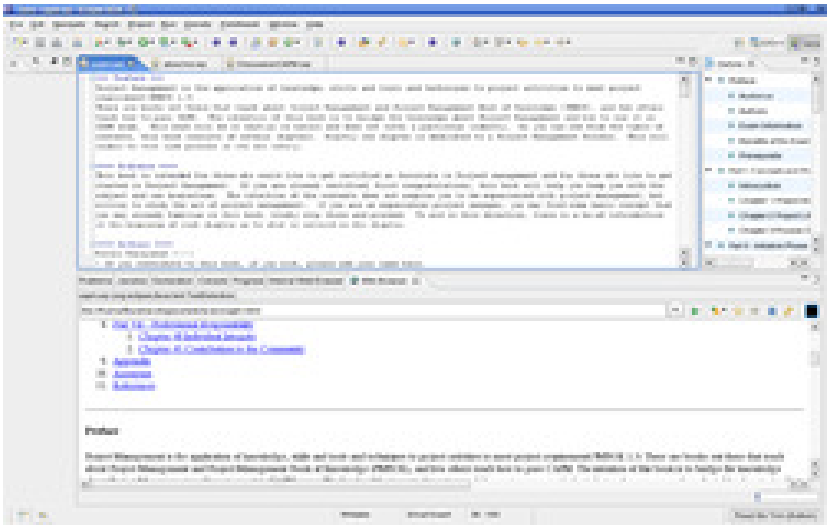

Ilustración 2: Eclipse 3.2

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, que cubre casi todas las áreas de Model Driven Engineering. Eclipse fue desarrollado originalmente por IBM como sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse dispone de un Editor de texto con resaltado de sintaxis. La compilación es en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (*wizards*) para creación de proyectos, clases, tests, etc., y refactorización. Asimismo, a través de "plugins" libremente disponibles es posible añadir control de versiones con Subversion e integración con Hibernate, además de muchos otros plugins como el de comprobación de posibles errores (FindBug), generación de UML, etc.

4.1.3. DBDesigner 4

Es una aplicación para el diseño visual de base de datos, que integra el diseño, el modelado, la creación y el mantenimiento de una base de datos en una única aplicación. Combina características profesionales y una interfaz de usuario clara y sencilla para ofrecer la forma más eficiente para gestionar bases de datos.

Es un proyecto de código abierto disponible para Microsoft Windows 2k/XP y Linux KDE/Gnome. Esta bajo licencia GPL. Además esta aplicación está desarrollada y optimizada para

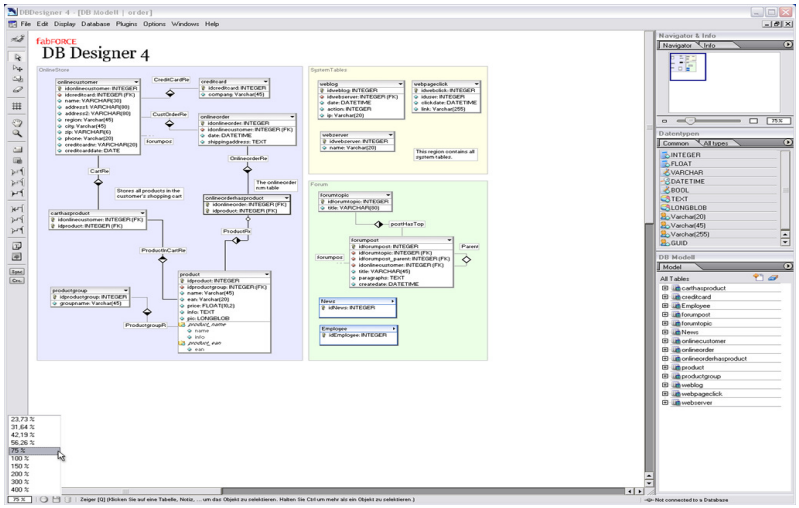

Ilustración 3: DBDesigner 4

trabajar con la base de datos MySQL, pará apoyar a los usuarios de esta base de datos en el diseño.

4.2. Bibliotecas java de soporte

Para realizar el proyecto fin de carrera se han utilizado bibliotecas java de código abierto de terceras personas o entidades. A continuación describimos estas bibliotecas:

4.2.1. log4j

Es una biblioteca open source desarrollada en Java por la *Apache Software Foundation* que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o "logs" (logging) en tiempo de ejecución mediante ficheros de configuración. En nuestro caso el fichero de configuración del Log4J contiene las siguientes lineas:

```
log4j.rootCategory=ALL
log4j.logger.NAML=Default
log4j.appender.Default=org.apache.log4j.RollingFileAppender
log4j.appender.Default.Threshold=ERROR
log4j.appender.Default.ImmediateFlush=true
log4j.appender.Default.file=./logs/naml.log
log4j.appender.Default.layout=org.apache.log4j.PatternLayout
log4j.appender.Default.layout.ConversionPattern=%d{HH:mm:ss.SSS} (%6r) %-5p [%-
7t] - %l - %m%n
log4j.appender.Default.append=true
log4j.appender.Default.MaxFileSize=10MB
log4j.appender.Default.MaxBackupIndex=5
```

Fichero 1: log4j.properties

La URL de descarga es:

http://logging.apache.org/log4j/1.2/download.html

4.2.2. mysql-connector-java

Biblioteca desarrollada por MySQL, que se encarga de dar conectividad a las aplicaciones Java con la base de datos MySQL a través de un conector JDBC, cuyo nombre es MySQL Connector / J. Es un conector JDBC de tipo 4. Esto quiere decir que es el cliente java el que controla la conexión mediante el protocolo MySQL, y no se basa en las bibliotecas cliente de MySQL. Su URL de descarga es :

http://dev.mysql.com/downloads/connector/j/

4.2.3. castor-core y castor-xml

Son bibliotecas open source que se encargan de transformar objetos java a documentos XML y viceversa. Son las bibliotecas que usaremos para, una vez obtenida la información de base de datos y creados los objetos necesarios, obtener un XML válido, que comprobaremos con los esquemas XSD. También haremos el paso inverso: recibir un documento XML y, aplicándole Castor, convertirlo en objetos java. Para poder realizar estas operaciones y que sólo incluya en el XML los atributos de cada clase que sean necesarios y, así poder cumplir las restricciones de las plantillas XSD, se tienen que crear ficheros XML que contengan la estructura de los objetos. Sus URL de descarga son:

Castor XML: http://dist.codehaus.org/castor/1.3.1/castor-1.3.1-xml.jar

Castor Core: http://dist.codehaus.org/castor/1.3.1/castor-1.3.1-core.jar

4.2.4. commons-dbcp

Es un componente desarrollado por Apache Projects que se encarga de interactuar con una base de datos relacional. Se encarga de mejorar las conexiones y el rendimiento a la hora de realizar un conjunto de conexiones a bases de datos. Por ello se utiliza junto a commos-pool en el desarrollo de este proyecto fin de carrera para crear el conjunto de conexiones a MySQL. Su URL de descarga es:

http://commons.apache.org/dbcp/download_dbcp.cgi

4.2.5. commons-pool

Es una biblioteca encargada de crear un conjunto de conexión a base de datos con tres aspectos principales:

1. Interfaz de conjuntos de conexiones genérico que los clientes y desarrolladores pueden

utilizar para proporcionar implementaciones.

2. Un conjunto de herramientas para la creación de conjuntos de conexiones..

3. Varias implementaciones de conjuntos de conexiones de propósito general.

Su URL de descarga es:

http://commons.apache.org/pool/download_pool.cgi

4.2.6. commons-logging

Es una biblioteca utilizada por las dos anteriores (commons-pool y commons-dbcp) para escribir información de log. Hace de interfaz con diferentes implementaciones de log (log4j,...). Su URL de descarga es:

http://commons.apache.org/logging/download_logging.cgi

5. Desarrollo realizado

En esta sección describiremos los distintos desarrollos que se han realizado durante el proyecto fin de carrera que nos ocupa. Esto es, una descripción de las bases de datos creadas y de las clases java que se han desarrollado para cada modulo.

5.1. Base de datos

Para el desarrollo de este proyecto fin de carrera, se han generado las tablas que necesitará cada módulo en base de datos, enlazando las tablas mediante las claves ajenas. Cada módulo tiene su propia base de datos, como describimos a continuación.

5.1.1. Módulo de estructuración del problema

Este módulo necesita almacenar la información del problema y cómo lo estructura el propietario del mismo. Para poder almacenar la información relevante a este módulo se ha creado la siguiente base de datos:

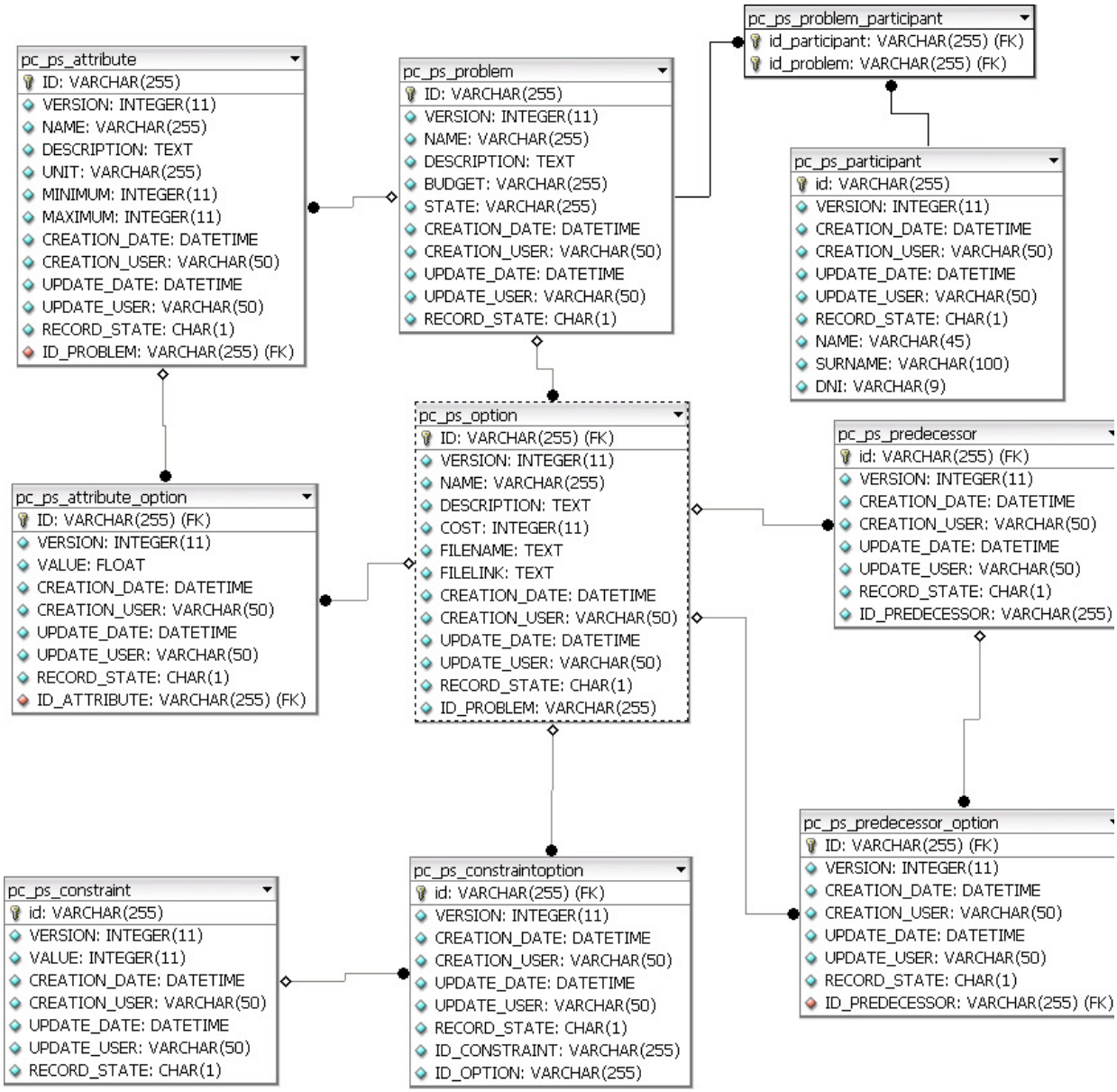

Ilustración 4: Tablas base de datos Estructuración

Como se aprecia el propietario del problema define el problema, las alternativas (option) así como las restricciones de número que tiene cada una (constraint) y las restricciones de dependencia (predecessor). También se definen los participantes en la participación (participant).

5.1.2. Módulo de comunicación de preferencias

En este módulo se almacena la información referente a las preferencias que cada participante tiene sobre las consecuencias del problema.

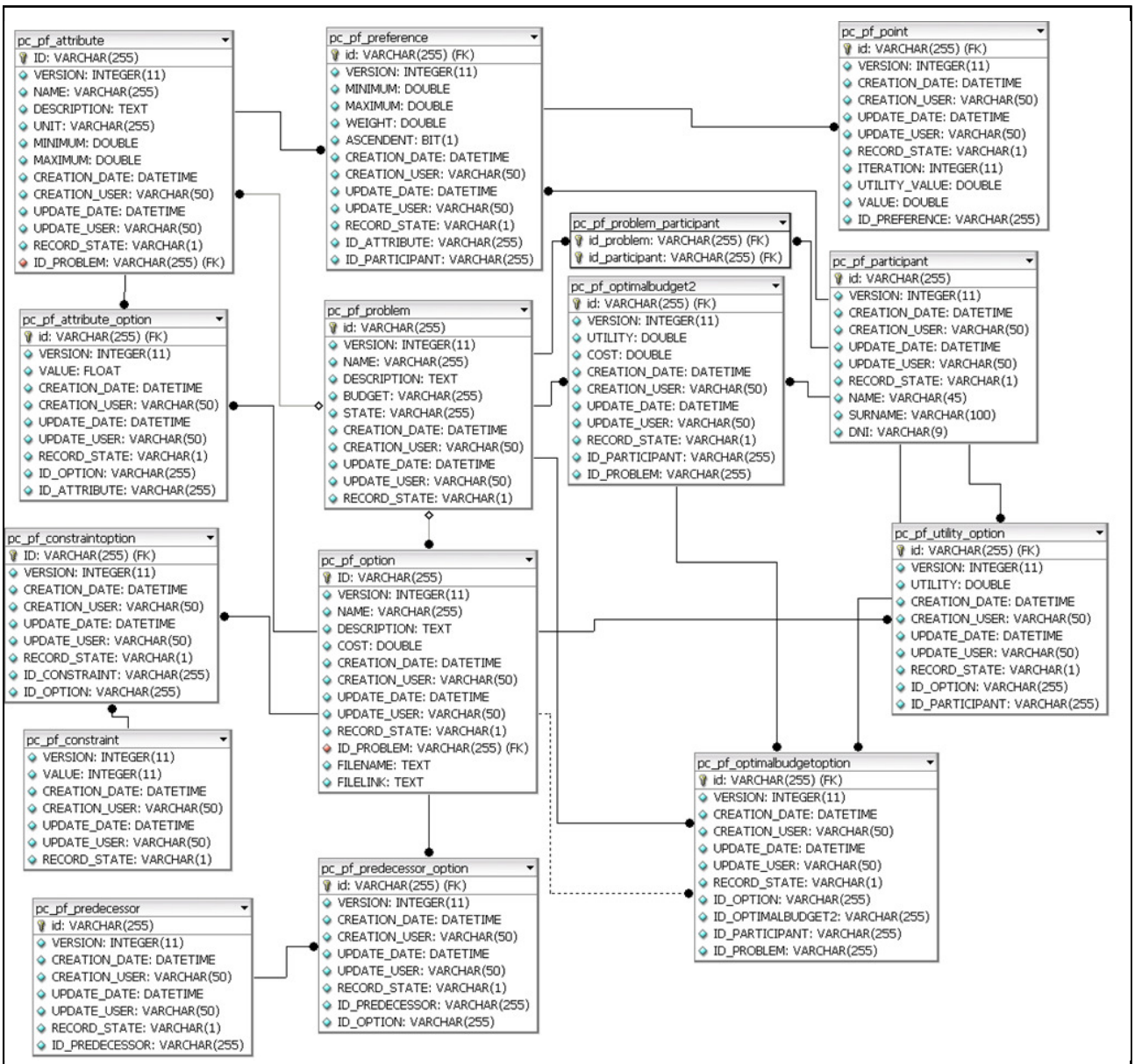

Ilustración 5: Tablas base de datos Preferencias

Estas tablas almacenan las preferencias de los participantes respecto a cada alternativa, asignándoles puntos a cada una de ellas, que se tendrán en cuenta en las fases de debate y negociación.

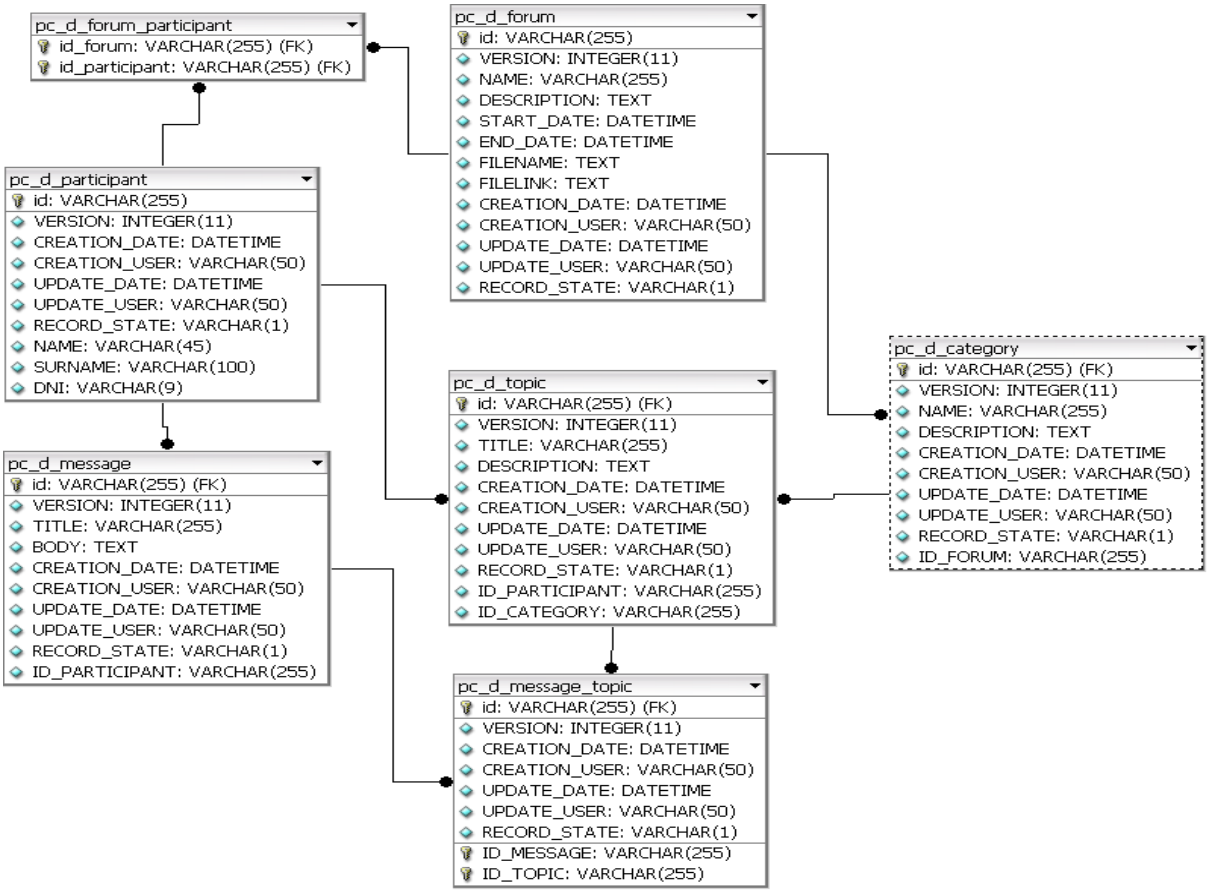

Ilustración 6: Tablas base de datos Debate

5.1.3. Módulo de debate

Es un módulo encargado de almacenar los intercambios de información entre participantes. Es el foro de la aplicación.

En este módulo cada participante puede abrir temas o escribir mensajes en temas abiertos por otros participantes, para intentar llegar a un acuerdo, e intentar poner fin al problema en un tiempo corto, sin tener que pasar por otros módulos, como el de votación.

Los diferentes temas se organizan por categorías, por lo que a la hora de localizarlos se facilita la tarea.

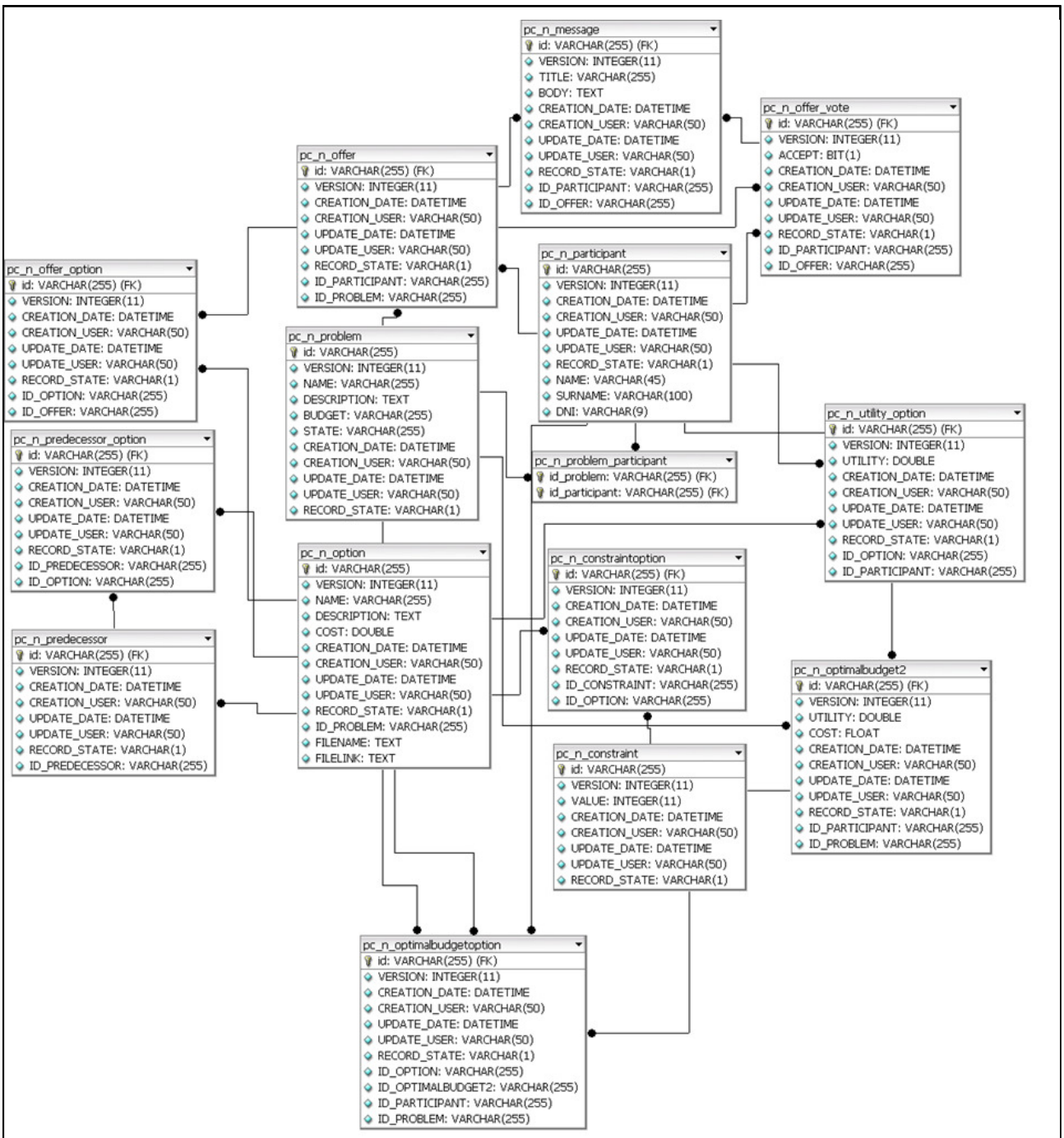

Ilustración 7: Tablas base de datos Negociación

5.1.4. Módulo de negociación

En este módulo cada participante puede ofrecer soluciones al problema, lo que se almacenará en las tablas del módulo.

Los participantes pueden hacer “ofertas” a la solución del problema, asignando unas propiedades para la votación en caso de ser necesaria. Cada solución aportada debe tener sus restricciones y sus costes.

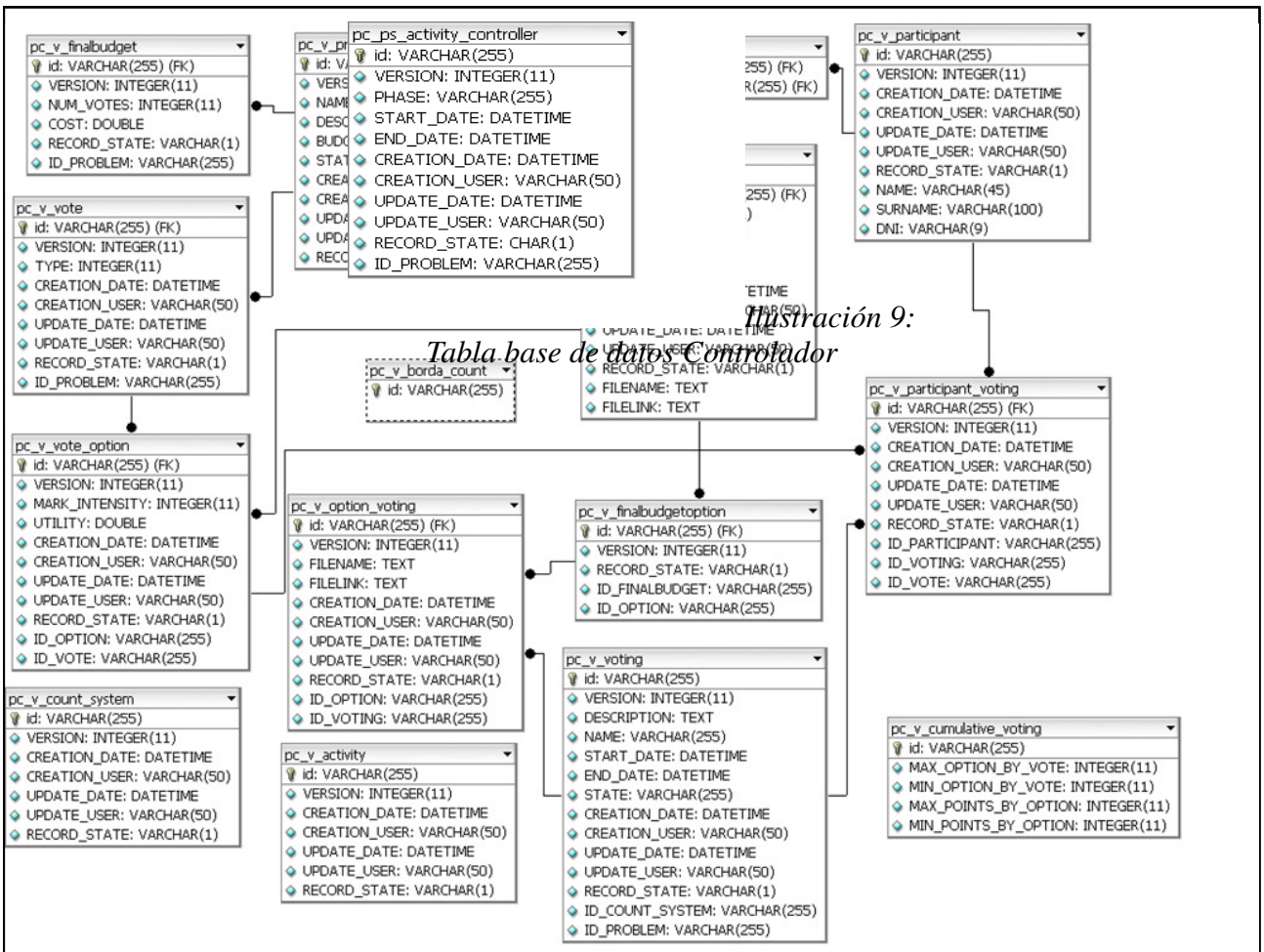

Ilustración 9: Tabla base de datos Controlador

Ilustración 8: Tablas base de datos Votación

5.1.5. Módulo de votación

Es el módulo encargado de la gestión de las votaciones, almacenando las votaciones de cada usuario.

Cada participante debe votar a todas las opciones del problema, cada opción tendrá un peso a la hora de decidir cual es la solución mas valorada por los participantes.

5.1.6. Módulo controlador

El módulo controlador es el que se encarga de comprobar cuando inicia o finaliza una fase y cual es la fase siguiente. Por ello, en lo que respecta a base de datos es un módulo muy sencillo con una única tabla en base de datos.

En esta tabla se representa cada fase (phase) así como las fechas en las que empieza y acaba cada una de ellas. Y es lo que el controlador tiene en cuenta para iniciar o finalizar las fases correspondientes.

5.2. Desarrollo Java

En este proyecto fin de carrera hemos desarrollado diversas clases Java que, una vez

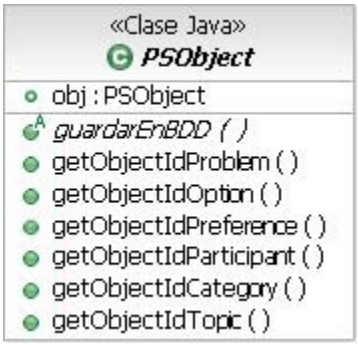

Ilustración 10: Clase PSObject.java

obtenidos los objetos, se han transformado en ficheros XML mediante Castor. Cada módulo de la aplicación ha necesitado un desarrollo especifico, debido a las características de las tablas en base de datos de cada uno de ellos. También se han desarrollado clases que sirven de unión entre otras para facilitar la obtención del XML final de cada módulo. Aun así, se ha desarrollado una clase común a casi todas ellas, montando un sistema jerárquico, con una clase padre y muchas hijo, tantas como tablas tenemos en la base de datos.

Los diagramas de clases de los distintos módulos no incluyen todas las clases que operan en cada uno de ellos, ya que el diagrama se haría inmanejable, por lo que se omiten las clases que lo que hacen es obtener y almacenar datos de la base de datos.

5.2.1. Desarrollo común

Se ha desarrollado una clase padre, que es una clase abstracta, en la que se han implementado algunas funciones, para poder obtener valores de nodos superiores en el XML y así poder almacenar correctamente la relación entre las tablas en la base de datos. A su vez, se han incluido funciones abstractas para asegurar el buen funcionamiento de la aplicación, tanto a la hora de obtener los datos para montar el XML como a la hora de almacenar datos en la base de datos. El diagrama de clase de este elemento es:

También se han desarrollado algunas clases de apoyo, como son:

- Una clase que realiza la conexión a base de datos mediante un conjunto de conexiones.
- Una clase que se encarga de ejecutar todas las consultas a base de datos.
- Una clase de logs.
- Una clase que contiene los resultados de las consultas realizadas a base de datos.
- Una clase para la interpretación de un fichero de configuración.

- Incluso, una clase con una función estática que formatea la hora para que MySQL sea capaz de interpretarla correctamente.

5.2.2. Módulo de estructuración del problema

Este módulo está compuesto de una clase principal o raíz llamada ProblemStructuring.java, que es a la que se llamará para montar el XML (getStructuringXML()) o para tratar el XML y guardar en base de datos (setStruncturingXML()). La función setStructuringXML() es la que se encarga, además, de tratar el XML, comprobando que el XML indicado tiene el formato especificado según el esquema XSD siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

	<!-- Participante del problema -->

	<xs:complexType name="participant">
		<xs:sequence>
			<xs:element name="name" type="xs:string" />
			<xs:element name="surname" type="xs:string" />
			<xs:element name="DNI" type="xs:string" />
		</xs:sequence>
		<xs:attribute name="id" id="ID_participant" use="required" />
	</xs:complexType>

	<!-- Nodo que contiene todos los participantes del problema →

	<xs:complexType name="participants">
		<xs:sequence>
			<xs:element name="participant" type="participant" minOccurs="0"
			maxOccurs="unbounded" />
		</xs:sequence>
		<xs:attribute name="num_participants" type="xs:nonNegativeInteger" />
	</xs:complexType>

	<!--Nodo que contiene las características (atributos, restricciones y opciones) del problema -->

	<xs:complexType name="characteristics">
		<xs:sequence>
			<xs:element ref="options" minOccurs="0" maxOccurs="1" />
			<xs:element ref="restrictions" minOccurs="0" maxOccurs="1" />
			<xs:element ref="attributes" minOccurs="0" maxOccurs="1" />
		</xs:sequence>
	</xs:complexType>

	<!-- Opción del problema -->

	<xs:complexType  name="option">
		<xs:sequence>
			<xs:element name="name" type="xs:string" minOccurs="0" />
			<xs:element name="description" type="xs:string" minOccurs="0" />
			<xs:element name="cost" type="xs:integer" minOccurs="0" />
			<xs:element ref="attribute-option" maxOccurs="unbounded"/>
		</xs:sequence>
		<xs:attribute name="id" id="ID_option" use="required" />
	</xs:complexType>
```

```
<!-- Nodo que contiene la lista de opciones del problema -->

<xs:element name="options">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="option" type="option" minOccurs="0"
			maxOccurs="unbounded" />
		</xs:sequence>
		<xs:attribute name="num_options" type="xs:nonNegativeInteger" />
	</xs:complexType>
</xs:element>

<!-- Nodo que enlaza una opcion con un atributo del problema -->

<xs:element name="attribute-option">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="value" type="xs:float"/>
			<xs:element name="attribute" type="string_attribute"/>
		</xs:sequence>
		<xs:attribute name="id" id="ID_attributeOption" use="required" />
	</xs:complexType>
</xs:element>

<!-- Restricción de tipo numérico del problema -->

<xs:complexType name="numberRestriction">
	<xs:sequence>
		<xs:element ref="constraintOption" minOccurs="0" maxOccurs="unbounded" />
		<xs:element name="numMax" type="xs:nonNegativeInteger" />
	</xs:sequence>
	<xs:attribute name="id" id="ID_numberRestriction" use="required" />
</xs:complexType>

<!-- Lista de opciones que incluye una restricción numérica del problema -->

<xs:element name="constraintOption">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="option" type="string_option" minOccurs="0"
			maxOccurs="unbounded" />
		</xs:sequence>
		<xs:attribute name="id" id="ID_constraintOption" use="required" />
	</xs:complexType>
</xs:element>

<!-- opcion dependiente de una restricción de dependencia del problema →

<xs:complexType name="optionDepends">
	<xs:sequence>
		<xs:element name="option" type="string_option"  maxOccurs="1"/>
	</xs:sequence>
	<xs:attribute name="id" id="ID_optionDepends" use="required" />
</xs:complexType>

<!-- Restricción de dependencia del problema →

<xs:complexType name="dependsRestriction">
	<xs:sequence>
		<xs:element  name="option_depends" type="optionDepends"
		id="option_depends" maxOccurs="1"/>
		<xs:element name="option" type="string_option" id="option" maxOccurs="1"/>
```

```
		</xs:sequence>
		<xs:attribute name="id" id="ID_dependsRestriction" use="required" />
	</xs:complexType>

	<!-- Nodo que contiene una lista de restricciones, ya sean de dependencia o de numero, del
	problema →

	<xs:element name="restrictions">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="numberRestriction" type="numberRestriction"
				minOccurs="0" maxOccurs="unbounded" />
				<xs:element name="dependsRestriction" type="dependsRestriction"
				minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>

	<!-- Atributo del problema -->

	<xs:element name="attribute">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="description" type="xs:string" />
				<xs:element name="unit" type="xs:string" />
				<xs:element name="min_value" type="xs:long" />
				<xs:element name="max_value" type="xs:long" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_attribute" use="required" />
		</xs:complexType>
	</xs:element>

	<!-- Nodo que contiene una lista de atributos del problema →

	<xs:element name="attributes">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="attribute" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
			<xs:attribute name="num_attributes" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>

	<!-- Nodo principal, el nodo problem, que describe el contenido del problema a resolver -->

	<xs:element name="problem">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="description" type="xs:string" />
				<xs:element name="characteristics" type="characteristics" />
				<xs:element name="participants" type="participants" minOccurs="0"
				maxOccurs="1" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_problem" use="required" />
		</xs:complexType>
	</xs:element>

	<!-- tipos de datos que creamos para poner solo el id del elemento y no todo -->

	<xs:complexType name="string_attribute">
```

```
                <xs:attribute name="id" use="required" />
        </xs:complexType>

        <xs:complexType name="string_option">
                <xs:attribute name="id" use="required" />
        </xs:complexType>
</xs:schema>
```

Esquema XSD 2: problemStructuring.xsd

El diagrama de clases que utiliza este módulo, en el cual no incluimos las clases que hacen de interfaz entre la base de datos y el módulo, es el siguiente:

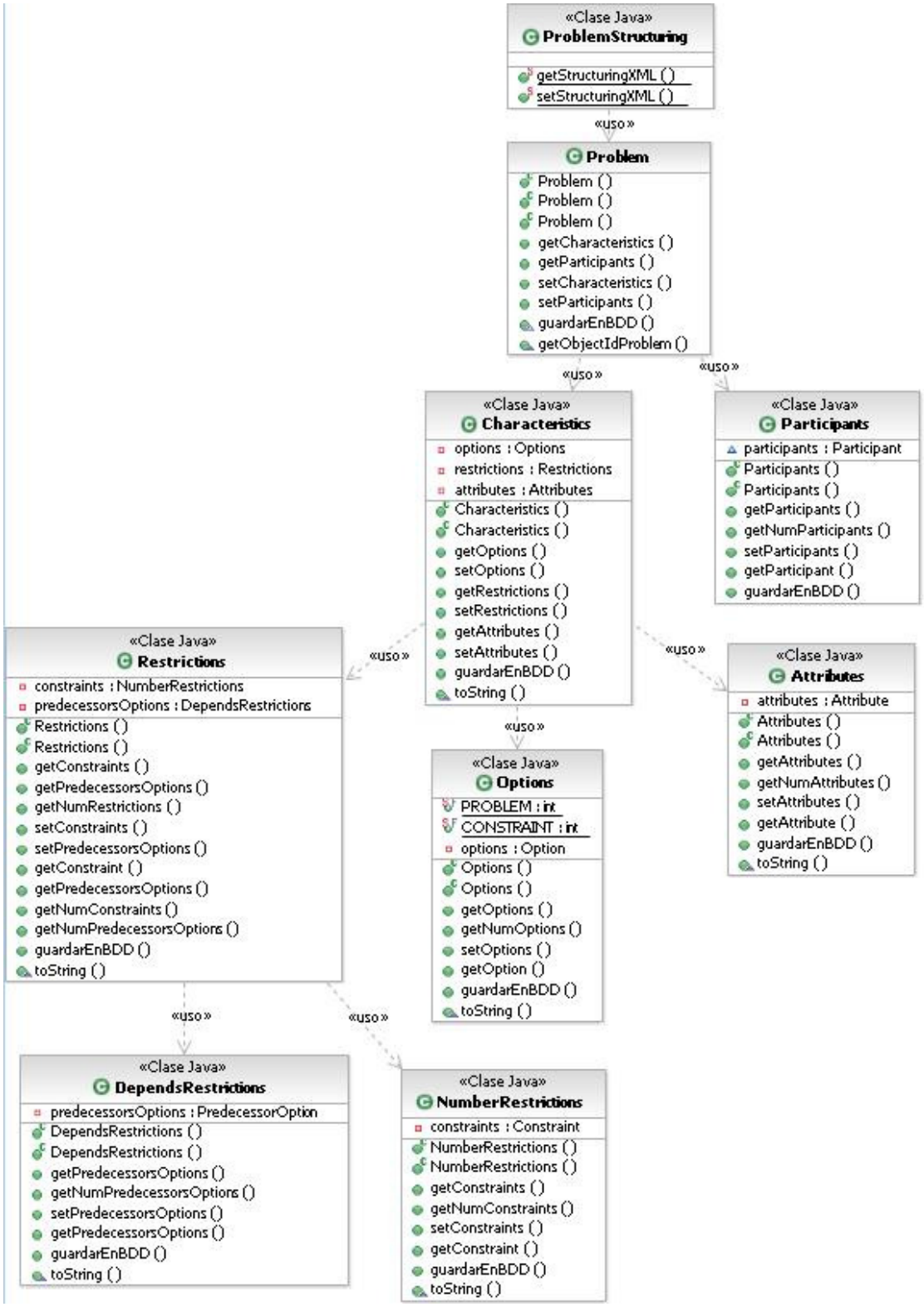

Ilustración 11: Diagrama de clases del Módulo de Estructuración del problema

De este diagrama, y del esquema XSD, se puede obtener la estructura que tendrá el XML. Un problema tendrá participantes y características. Las características están divididas en atributos, restricciones y opciones. A su vez, las restricciones están divididas en restricciones de número y de dependencia.

5.2.3. Módulo de debate

El esquema XSD de este módulo es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<!-- Participantes del problema -->
	<xs:element name="participant">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="surname" type="xs:string" />
				<xs:element name="DNI" type="xs:string" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_participant" use="required" />
		</xs:complexType>
	</xs:element>

	<xs:element name="participants">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="participant" minOccurs="0" maxOccurs="unbounded"
				/>
			</xs:sequence>
			<xs:attribute name="num_participants" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>

	<!-- Algoritmo de debate -->
	<xs:element name="message">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="title" type="xs:string" />
				<xs:element name="body" type="xs:string" />
				<xs:element name="date" type="xs:dateTime" />
				<xs:element name="participant" type="string_participant"/>
			</xs:sequence>
			<xs:attribute name="id" id="ID_message" use="required" />
		</xs:complexType>
	</xs:element>

	<xs:element name="messages">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="message" minOccurs="1" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="num_messages" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>

	<xs:element name="topic">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="title" type="xs:string" />
				<xs:element name="description" type="xs:string" />
				<xs:element ref="messages" />
				<xs:element name="participant" type="string_participant"/>
			</xs:sequence>
			<xs:attribute name="id" id="ID_topic" use="required" />
		</xs:complexType>
	</xs:element>
```

```
	<xs:element name="topics">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="topic" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
			<xs:attribute name="num_topics" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>

	<xs:element name="category">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="description" type="xs:string" />
				<xs:element ref="topics" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_category" use="required" />
		</xs:complexType>
	</xs:element>

	<xs:element name="categories">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="category" minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="num_categories" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>

	<xs:element name="forum">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="description" type="xs:string" />
				<xs:element ref="categories" />
				<xs:element ref="participants"/>
			</xs:sequence>
			<xs:attribute name="id" id="ID_forum" use="required" />
		</xs:complexType>
	</xs:element>
	<!-- tipos de datos que creamos para poner solo el id del elemento y no todo -->
	<xs:complexType name="string_participant">
		<xs:attribute name="id" use="required" />
	</xs:complexType>
</xs:schema>
```

Esquema XSD 3: dialog.xsd

El diagrama de clases de este módulo es el siguiente:

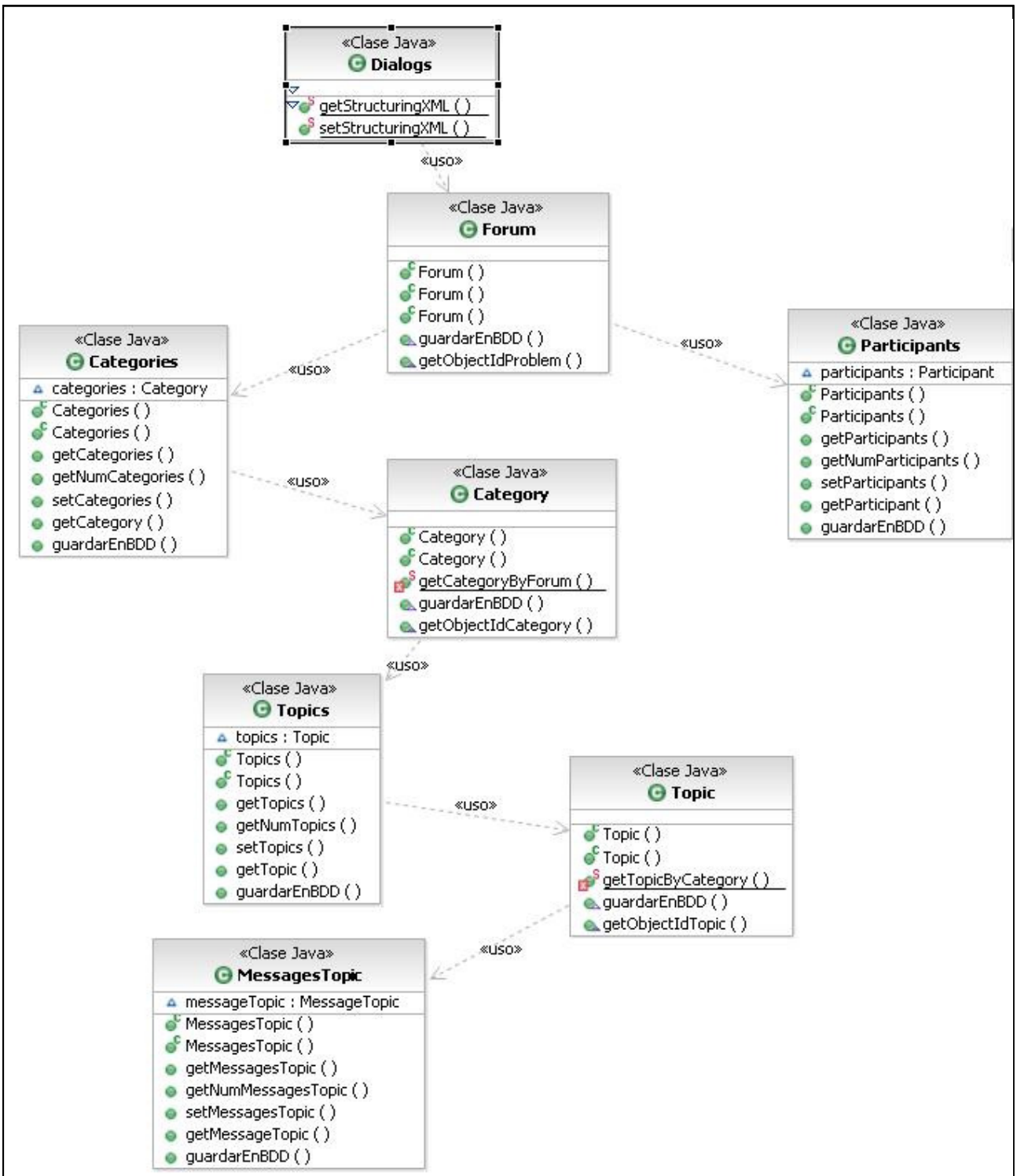

Ilustración 12: Diagrama de clases del Módulo de Debate

Al igual que el resto de módulos tiene una clase principal, con los métodos getStructuringXML(), encargado de crear el XML a partir de la información que hay en la base de datos, y setStructuringXML() encargado de comprobar que el XML sea válido según el esquema y de almacenar los datos en la base de datos. Al tratarse de un módulo de debate, se trata como un foro, en el cual cada participante puede escribir sobre un problema, iniciando temas o respondiendo a temas propios o de otros usuarios. Estos temas se organizan en categorías.

5.2.4. Módulo de comunicación de preferencias

El módulo de preferencias está compuesto por varias clases, que se encargan de obtener datos de la base de datos, construir un XML, de tratarlo y almacenarlo en base de datos. Como en el resto de módulos, el XML debe cumplir ciertas condiciones, que vienen definidas en su esquema XSD:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<!-- Participantes del problema -->
	<xs:element name="participant">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="surname" type="xs:string" />
				<xs:element name="DNI" type="xs:string" />
				<xs:element name="profile" type="profile" minOccurs="0" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_participant" use="required" />
		</xs:complexType>
	</xs:element>

	<xs:element name="participants">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="participant" minOccurs="0" maxOccurs="unbounded"
				/>
			</xs:sequence>
			<xs:attribute name="num_participants" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>

	<!-- Características (atributos, restricciones y opciones) del problema -->
	<xs:element name="characteristics">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="options" minOccurs="0" maxOccurs="1" />
				<xs:element ref="restrictions" minOccurs="0" maxOccurs="1" />
				<xs:element ref="attributes" minOccurs="0" maxOccurs="1" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>

	<!-- Opciones del problema -->
	<xs:element name="option">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" minOccurs="0" />
				<xs:element name="description" type="xs:string" minOccurs="0" />
				<xs:element name="cost" type="xs:integer" minOccurs="0" />
				<xs:element ref="attribute-option" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="id" id="ID_option" use="required" />
		</xs:complexType>
	</xs:element>
	<xs:element name="options">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="option" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
			<xs:attribute name="num_options" type="xs:nonNegativeInteger" />
```

```
		</xs:complexType>
	</xs:element>
	<xs:element name="attribute-option">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="value" type="xs:float"/>
				<xs:element name="attribute" type="string_attribute"/>
			</xs:sequence>
			<xs:attribute name="id" id="ID_attributeOption" use="required" />
		</xs:complexType>
	</xs:element>

	<!-- Restricciones del problema -->
	<xs:complexType name="numberRestriction">
		<xs:sequence>
			<xs:element ref="constraintOption" minOccurs="0" maxOccurs="unbounded" />
			<xs:element name="numMax" type="xs:nonNegativeInteger" />
		</xs:sequence>
		<xs:attribute name="id" id="ID_numberRestriction" use="required" />
	</xs:complexType>

	<xs:element name="constraintOption">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="option" type="string_option" minOccurs="0"
				maxOccurs="unbounded" />
			</xs:sequence>
		<xs:attribute name="id" id="ID_constraintOption" use="required" />
		</xs:complexType>
	</xs:element>

	<!-- La opción 'option' depende de la opción 'option_depends' -->
	<xs:complexType name="dependsRestriction">
		<xs:sequence>
			<xs:element  name="option_depends" type="optionDepends"
			id="option_depends" maxOccurs="1"/>
			<xs:element name="option" type="string_option" id="option" maxOccurs="1"/>
		</xs:sequence>
		<xs:attribute name="id" id="ID_dependsRestriction" use="required" />
	</xs:complexType>

	<xs:complexType name="optionDepends">
		<xs:sequence>
			<xs:element name="option" type="string_option"  maxOccurs="1"/>
		</xs:sequence>
		<xs:attribute name="id" id="ID_optionDepends" use="required" />
	</xs:complexType>

	<xs:element name="restrictions">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="numberRestriction" type="numberRestriction"
				minOccurs="0" maxOccurs="unbounded" />
				<xs:element name="dependsRestriction" type="dependsRestriction"
				minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>

	<!-- Atributos del problema -->
	<xs:element name="attribute">
		<xs:complexType>
```

```
                <xs:sequence>
                        <xs:element name="name" type="xs:string" />
                        <xs:element name="description" type="xs:string" />
                        <xs:element name="unit" type="xs:string" />
                        <xs:element name="min_value" type="xs:long" />
                        <xs:element name="max_value" type="xs:long" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_attribute" use="required" />
        </xs:complexType>
</xs:element>

<xs:element name="attributes">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="attribute" minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="num_attributes" type="xs:nonNegativeInteger" />
        </xs:complexType>
</xs:element>

<xs:element name="preference">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="minimum" type="xs:double" />
                        <xs:element name="maximum" type="xs:double" />
                        <xs:element name="weigth" type="range" />
                        <xs:element name="ascendent" type="xs:boolean"/>
                        <xs:element ref="preference_points" minOccurs="0" maxOccurs="1" />
                        <xs:element name="attribute" type="string_attribute" />
                </xs:sequence>
                <xs:attribute name="id" type="xs:string"/>
        </xs:complexType>
</xs:element>

<!-- Perfil de cada participante con la función de utilidad que tiene cada una de las alternativas
        y con las preferencias sobre cada criterio -->

<xs:complexType name="profile">
        <xs:sequence>
                <xs:element ref="utilities_function" minOccurs="0" maxOccurs="1" />
                <xs:element ref="preferences" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
</xs:complexType>

<xs:element name="preferences">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="preference" minOccurs="0" maxOccurs="unbounded"
                        />
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:simpleType name="iteration_values">
        <xs:restriction base="xs:integer">
                <xs:minInclusive value="1" />
                <xs:maxInclusive value="3" />
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="range">
        <xs:restriction base="xs:double">
```

```
                <xs:minInclusive value="0.0" />
                <xs:maxInclusive value="100.0" />
            </xs:restriction>
        </xs:simpleType>

        <xs:element name="preference_points">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="preference_point" minOccurs="0"
                    maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>

        <xs:element name="preference_point">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="iteration" type="iteration_values" />
                    <xs:element name="utility" type="range" />
                    <xs:element name="value" type="xs:double" />
                </xs:sequence>
                <xs:attribute name="id" type="xs:string"/>
            </xs:complexType>
        </xs:element>

        <xs:element name="utilities_function">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="utility_function" type="utility_function"
                    minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>

        <xs:complexType name="utility_function">
            <xs:sequence>
                <xs:element ref="optimalBudgetOption" minOccurs="0" maxOccurs="1" />
                <xs:element name="option" type="string_option" />
            </xs:sequence>
            <xs:attribute name="id" id="ID_utilityOption" use="required" />
            <xs:attribute name="utility_fv" type="xs:double" use="required" />
        </xs:complexType>


        <xs:element name="optimalBudgetOption">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="optimalBudget2" minOccurs="1" maxOccurs="1" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_optimalbudgetoption" use="required" />
            </xs:complexType>
        </xs:element>

        <xs:element name="optimalBudget2">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="cost" type="xs:double" minOccurs="1"
                    maxOccurs="1" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_optimalbudget2" use="required" />
                <xs:attribute name="utility" type="xs:double" use="required" />
            </xs:complexType>
```

```
        </xs:element>

        <!-- Problema a resolver -->
        <xs:element name="problem">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="name" type="xs:string" />
                                <xs:element name="description" type="xs:string" />
                                <xs:element ref="characteristics" />
                                <xs:element ref="participants" minOccurs="0"
                                        maxOccurs="1" />
                        </xs:sequence>
                        <xs:attribute name="id" id="ID_problem" use="required" />
                </xs:complexType>
        </xs:element>

<!-- tipos de datos que creamos para poner solo el id del elemento y no todo -->
        <xs:complexType name="string_attribute">
                <xs:attribute name="id" use="required" />
        </xs:complexType>
        <xs:complexType name="string_option">
                <xs:attribute name="id" use="required" />
        </xs:complexType>

</xs:schema>
```

Esquema XSD 4: preference.xsd

El diagrama de clases de este módulo sigue un esquema muy parecido a los módulos de estructuración del problema y al de negociación ya que, en los tres casos, los datos a tratar son muy parecidos. En este módulo se trata la información relativa a las preferencias que tiene cada participante sobre el problema, las distintas opciones y sus restricciones.

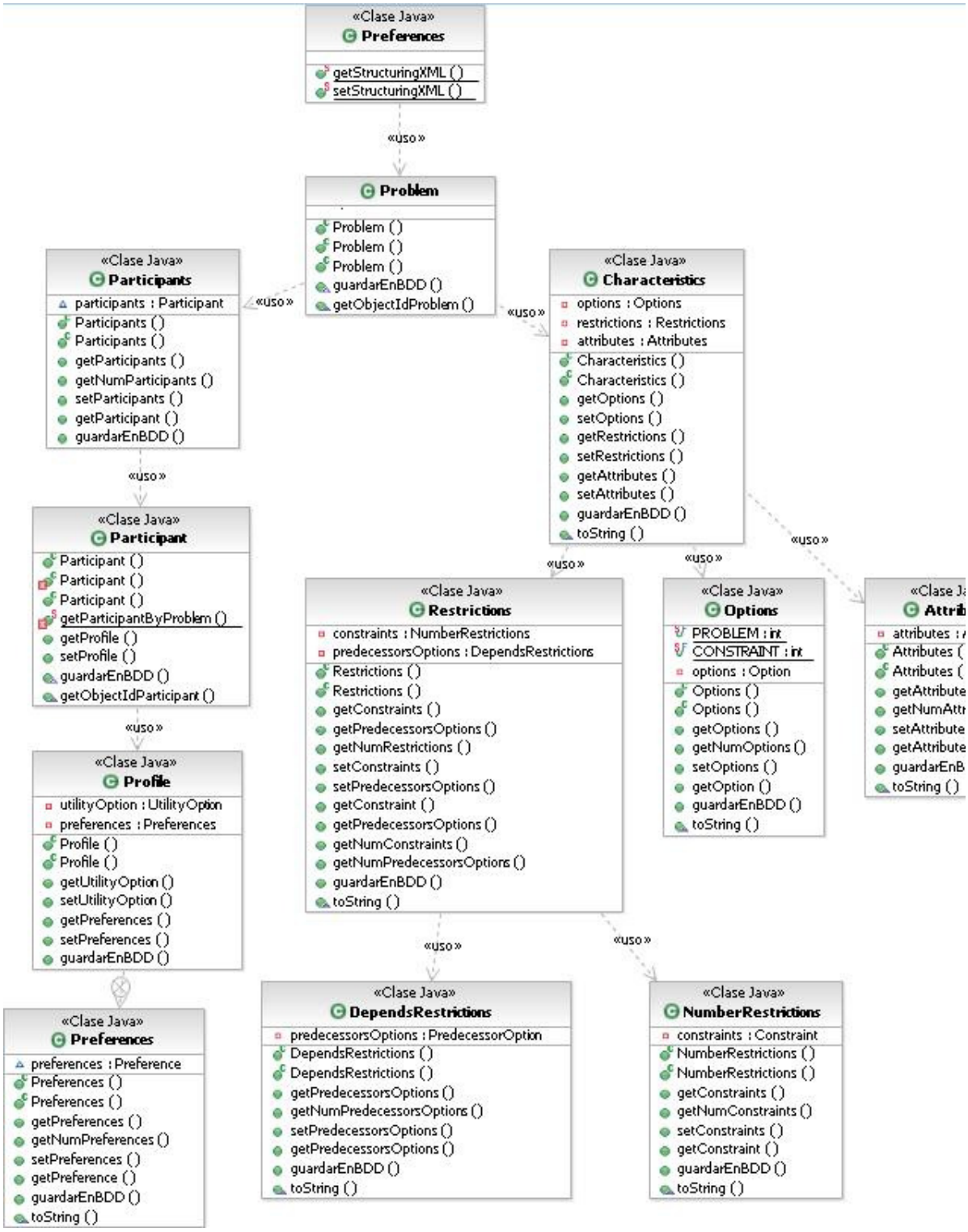

Ilustración 13: Diagrama de clases del Módulo de comunicación de preferencias

5.2.5. Módulo de negociación

Al igual que el resto de módulos, este tiene una parte de verificación de XML de entrada, para comprobar que tiene todos los campos obligatorios. El esquema XSD es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
```

```
<!-- Participantes del problema -->
<xs:element name="participant">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="name" type="xs:string" maxOccurs="1" />
			<xs:element name="surname" type="xs:string" maxOccurs="1" />
			<xs:element name="DNI" type="xs:string" maxOccurs="1" />
			<xs:element ref="utility_option" minOccurs="0"
			maxOccurs="unbounded" />
			<xs:element ref="offers" minOccurs="0" maxOccurs="1"/>
		</xs:sequence>
		<xs:attribute name="id" id="ID_participant" use="required" />
	</xs:complexType>
</xs:element>

<xs:element name="offer_vote">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="accept" type="xs:boolean" />
			<xs:element name="participant" type="string_participant" />
		</xs:sequence>
		<xs:attribute name="id" use="required" />
	</xs:complexType>
</xs:element>

<xs:element name="offer_votes">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="offer_vote" minOccurs="0" maxOccurs="unbounded"
			/>
		</xs:sequence>
		<xs:attribute name="num_offer_votes" type="xs:nonNegativeInteger" />
	</xs:complexType>
</xs:element>

<xs:element name="offer_option">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="option" type="string_option" />
		</xs:sequence>
		<xs:attribute name="id" use="required" />
	</xs:complexType>
</xs:element>

<xs:element name="message">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="title" type="xs:string" />
			<xs:element name="body" type="xs:string" minOccurs="0"
			maxOccurs="1"/>
			<xs:element name="participant" type="string_participant"/>
		</xs:sequence>
		<xs:attribute name="id" use="required" />
	</xs:complexType>
</xs:element>

<xs:element name="messages">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="message" minOccurs="0" maxOccurs="unbounded"
			/>
		</xs:sequence>
```

```
                <xs:attribute name="num_messages" type="xs:nonNegativeInteger" />
            </xs:complexType>
        </xs:element>

        <xs:element name="offer">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="offer_votes" minOccurs="0" maxOccurs="1" />
                    <xs:element ref="offer_option" minOccurs="0"
                    maxOccurs="unbounded" />
                    <xs:element ref="messages" minOccurs="0" maxOccurs="1" />
                </xs:sequence>
                <xs:attribute name="id" use="required" />
            </xs:complexType>
        </xs:element>

        <xs:element name="offers">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="offer" minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="num_offers" type="xs:nonNegativeInteger" />
            </xs:complexType>
        </xs:element>

        <xs:element name="optimal_budget2">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="cost" type="xs:float" />
                    <xs:element name="utility" type="xs:float" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_optimal_budget2" use="required" />
            </xs:complexType>
        </xs:element>

        <xs:element name="optimal_budget">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="optimal_budget2" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_optimal_budget" use="required" />
            </xs:complexType>
        </xs:element>

        <xs:element name="utility_option">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="utility" type="xs:float" />
                    <xs:element ref="optimal_budget" minOccurs="0" maxOccurs="1" />
                    <xs:element name="option" type="string_option" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_utility_option" use="required" />
            </xs:complexType>
        </xs:element>

        <xs:element name="participants">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="participant" minOccurs="0"
                    maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="num_participants" type="xs:nonNegativeInteger" />
```

```
		</xs:complexType>
	</xs:element>
	<!-- Características (atributos, restricciones y opciones) del problema -->
	<xs:element name="characteristics">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="options" minOccurs="0" maxOccurs="1" />
				<xs:element ref="restrictions" minOccurs="0" maxOccurs="1" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- Opciones del problema -->
	<xs:element name="option">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" minOccurs="0" />
				<xs:element name="description" type="xs:string" minOccurs="0" />
				<xs:element name="cost" type="xs:integer" minOccurs="0" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_option" use="required" />
		</xs:complexType>
	</xs:element>
	<xs:element name="options">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="option" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
			<xs:attribute name="num_options" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>
	<!-- Restricciones del problema -->
		<xs:complexType name="numberRestriction">
		<xs:sequence>
			<xs:element ref="constraintOption" minOccurs="0" maxOccurs="unbounded" />
			<xs:element name="numMax" type="xs:nonNegativeInteger" />
		</xs:sequence>
		<xs:attribute name="id" id="ID_numberRestriction" use="required" />
	</xs:complexType>
	<xs:element name="constraintOption">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="option" type="string_option" minOccurs="0"
				maxOccurs="unbounded" />
			</xs:sequence>
		<xs:attribute name="id" id="ID_constraintOption" use="required" />
		</xs:complexType>
	</xs:element>
	<!-- La opción 'option' depende de la opción 'option_depends' -->
	<xs:complexType name="dependsRestriction">
		<xs:sequence>
			<xs:element  name="option_depends" type="optionDepends"
			id="option_depends" maxOccurs="1"/>
			<xs:element name="option" type="string_option" id="option" maxOccurs="1"/>
		</xs:sequence>
		<xs:attribute name="id" id="ID_dependsRestriction" use="required" />
	</xs:complexType>
	<xs:complexType name="optionDepends">
		<xs:sequence>
			<xs:element name="option" type="string_option"  maxOccurs="1"/>
		</xs:sequence>
		<xs:attribute name="id" id="ID_optionDepends" use="required" />
	</xs:complexType>
```

```
	<xs:element name="restrictions">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="numberRestriction" type="numberRestriction"
				minOccurs="0" maxOccurs="unbounded" />
				<xs:element name="dependsRestriction" type="dependsRestriction"
				minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- Problema a resolver -->
	<xs:element name="problem">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="description" type="xs:string" />
				<xs:element ref="characteristics" />
				<xs:element ref="participants" minOccurs="0" maxOccurs="1" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_problem" use="required" />
		</xs:complexType>
	</xs:element>
	<!-- tipos de datos que creamos para poner solo el id del elemento y no todo -->
	<xs:complexType name="string_attribute">
		<xs:attribute name="id" use="required" />
	</xs:complexType>
	<xs:complexType name="string_option">
		<xs:attribute name="id" use="required" />
	</xs:complexType>
	<xs:complexType name="string_participant">
		<xs:attribute name="id" use="required" />
	</xs:complexType>
</xs:schema>
```

Esquema XSD 5: negotiation.xsd

Su diagrama de clases es el siguiente:

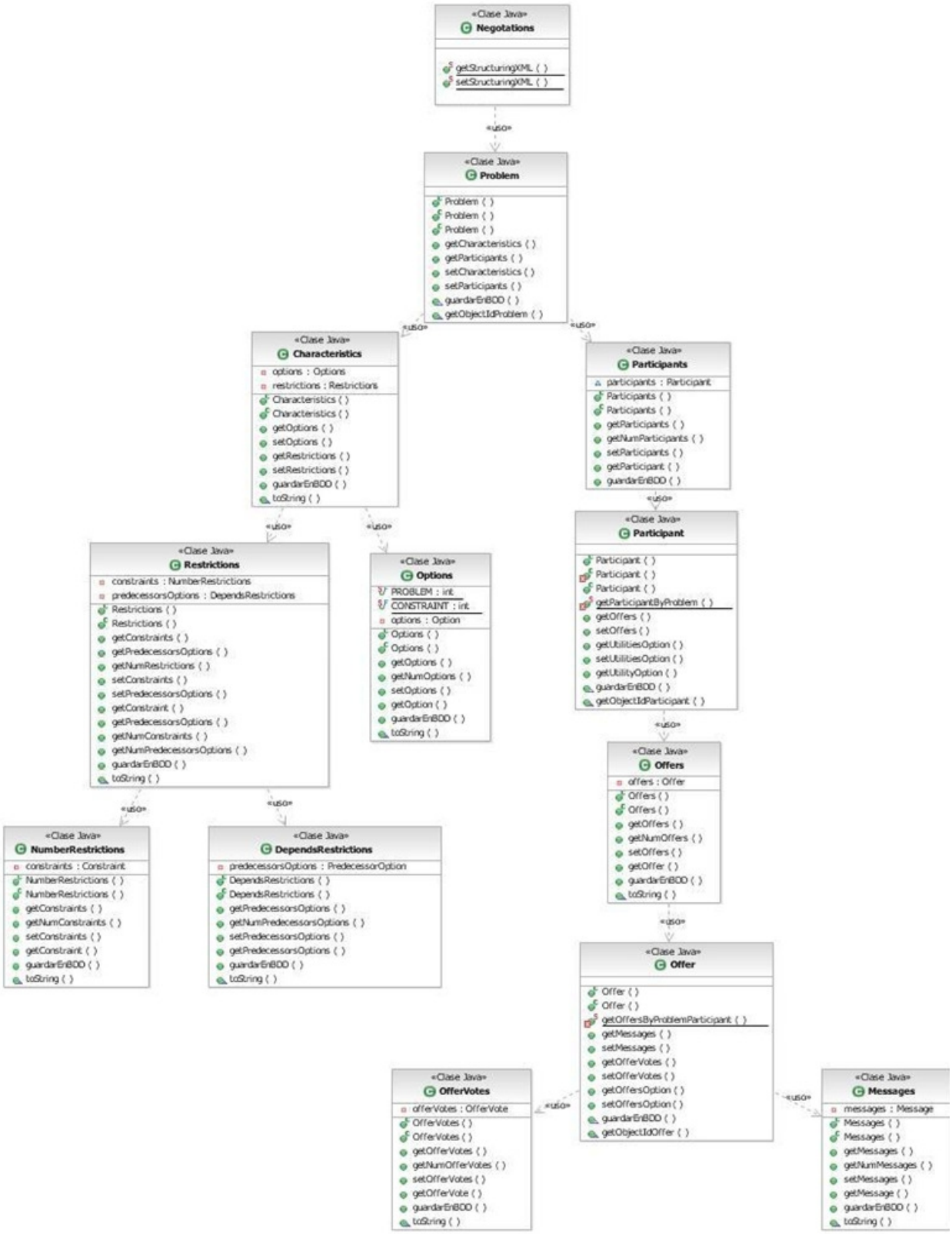

Ilustración 14: Diagrama de clases del Módulo de Negociación

En este módulo cada participante puede ofrecer soluciones al problema propuesto, que son votadas por el resto de participantes.

5.2.6. Módulo de votación

Al igual que el resto de módulos, éste verifica la estructura del XML mediante un esquema

XSD:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<!-- Participantes del problema -->
	<xs:element name="participant">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" />
				<xs:element name="surname" type="xs:string" />
				<xs:element name="DNI" type="xs:string" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_participant" use="required" />
		</xs:complexType>
	</xs:element>
	<xs:element name="participants">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="participant" minOccurs="0"
				maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="num_participants" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>
	<!-- Características (atributos, restricciones y opciones) del problema -->
	<xs:element name="characteristics">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="options" minOccurs="0" maxOccurs="1" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- Opciones del problema -->
	<xs:element name="option">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="name" type="xs:string" minOccurs="0" />
				<xs:element name="description" type="xs:string" minOccurs="0" />
				<xs:element name="cost" type="xs:integer" minOccurs="0" />
			</xs:sequence>
			<xs:attribute name="id" id="ID_option" use="required" />
		</xs:complexType>
	</xs:element>
	<xs:element name="options">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="option" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
			<xs:attribute name="num_options" type="xs:nonNegativeInteger" />
		</xs:complexType>
	</xs:element>
	<xs:element name="vote_type">
		<xs:simpleType>
			<xs:restriction base="xs:integer">
				<xs:enumeration value="0" />
				<xs:enumeration value="1" />
				<xs:enumeration value="2" />
			</xs:restriction>
		</xs:simpleType>
	</xs:element>
	<xs:element name="vote_option">
		<xs:complexType>
```

```
                <xs:sequence>
                        <xs:element name="mark_intensity" type="xs:integer" />
                        <xs:element name="option" type="string_option" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_vote_option" use="required" />
        </xs:complexType>
</xs:element>
<xs:element name="vote">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="vote_type" />
                        <xs:element ref="vote_option" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="id" use="required" />
        </xs:complexType>
</xs:element>
<xs:element name="votes">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="vote" minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="num_votes" type="xs:nonNegativeInteger" />
        </xs:complexType>
</xs:element>
<xs:element name="option_voting">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="option" type="string_option" />
                </xs:sequence>
                <xs:attribute name="id" use="required" />
        </xs:complexType>
</xs:element>
<xs:element name="options_voting">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="option_voting" minOccurs="0"
                        maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="num_options_voting" type="xs:nonNegativeInteger" />
        </xs:complexType>
</xs:element>
<xs:element name="participant_voting">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="participant" type="string_participant" />
                        <xs:element name="vote" type="string_vote" />
                </xs:sequence>
                <xs:attribute name="id" use="required" />
        </xs:complexType>
</xs:element>
<xs:element name="participants_voting">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="participant_voting" minOccurs="0"
                        maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="num_participants_voting" type="xs:nonNegativeInteger" />
        </xs:complexType>
</xs:element>
<xs:element name="voting">
        <xs:complexType>
```

```
                <xs:sequence>
                        <xs:element ref="final_budget" />
                        <xs:element ref="options_voting" />
                        <xs:element ref="participants_voting" />
                </xs:sequence>
                <xs:attribute name="id" use="required" />
            </xs:complexType>
        </xs:element>
        <xs:element name="final_budget_option">
            <xs:complexType>
                <xs:sequence>
                        <xs:element name="option" type="string_option" />
                </xs:sequence>
                <xs:attribute name="id" use="required" />
            </xs:complexType>
        </xs:element>
        <xs:element name="final_budget">
            <xs:complexType>
                <xs:sequence>
                        <xs:element name="num_votes" type="xs:nonNegativeInteger" />
                        <xs:element name="cost" type="xs:double" />
                        <xs:element ref="final_budget_option" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="id" use="required" />
            </xs:complexType>
        </xs:element>
        <!-- Problema a resolver -->
        <xs:element name="problem">
            <xs:complexType>
                <xs:sequence>
                        <xs:element name="name" type="xs:string" />
                        <xs:element name="description" type="xs:string" />
                        <xs:element ref="characteristics" />
                        <xs:element ref="participants" minOccurs="0" maxOccurs="1" />
                        <xs:element ref="votes" minOccurs="0" maxOccurs="1" />
                        <xs:element ref="voting" />
                </xs:sequence>
                <xs:attribute name="id" id="ID_problem" use="required" />
            </xs:complexType>
        </xs:element>
<!-- tipos de datos que creamos para poner solo el id del elemento y no todo -->
        <xs:complexType name="string_option">
            <xs:attribute name="id" use="required" />
        </xs:complexType>

        <xs:complexType name="string_participant">
            <xs:attribute name="id" use="required" />
        </xs:complexType>
        <xs:complexType name="string_vote">
            <xs:attribute name="id" use="required" />
        </xs:complexType>
</xs:schema>
```

Esquema XSD 6: Voting.xsd

El diagrama de clases de este módulo es similar al de los anteriores módulos, ya que se ha desarrollado toda la aplicación de modo que sea fácil el cambio de cualquier desarrollo entre los módulos.

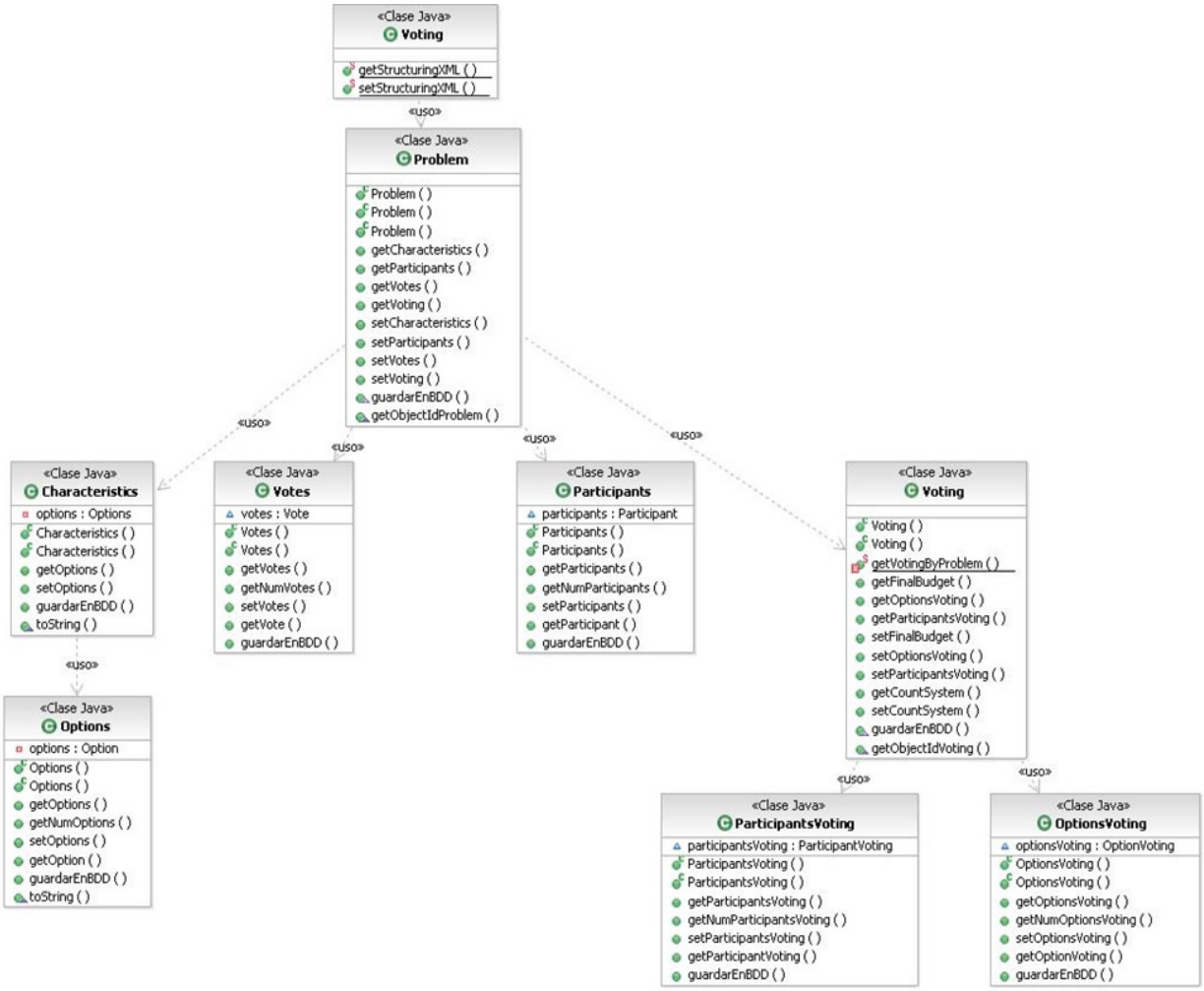

Ilustración 15: Diagrama de clases del Módulo de Votación

En este módulo se almacena la información de la votación, en caso de ser necesaria dicha votación. Se controla que la opción ganadora satisfaga los requisitos del problema. En caso contrario en módulo controlador realizara las gestiones necesarias y pasara al siguiente modulo según la configuración añadida inicialmente por el propietario del problema.

5.2.7. Módulo de controlador

Este es el módulo más sencillo en cuanto a base de datos se refiere, ya que se encarga de activar un módulo u otro. Lo único que hace es guardar información de cuando empieza una fase y cuando acaba. Su esquema XSD es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
        <!-- Fases -->
        <xs:element name="phase">
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:enumeration value="DEBATE" />
                                <xs:enumeration value="RE_STRUCTURING" />
```

```
                        <xs:enumeration value="PREFERENCES" />
                        <xs:enumeration value="NEGOTIATION" />
                        <xs:enumeration value="VOTING" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <!-- Controlador -->
            <xs:element name="controller">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="phase" minOccurs="0" />
                        <xs:element name="start" type="xs:dateTime"
                        minOccurs="0" />
                        <xs:element name="finish" type="xs:dateTime"
                        minOccurs="0" />
                    </xs:sequence>
                    <xs:attribute name="id" id="ID_controller" use="required" />
                </xs:complexType>
            </xs:element>
            <xs:element name="problem">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="name" type="xs:string" />
                        <xs:element name="description" type="xs:string" />
                        <xs:element ref="controller" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="id" id="ID_problem" use="required" />
                </xs:complexType>
            </xs:element>
</xs:schema>
```

Esquema XSD 7: controller.xsd

El diagrama de clases de este módulo es el siguiente:

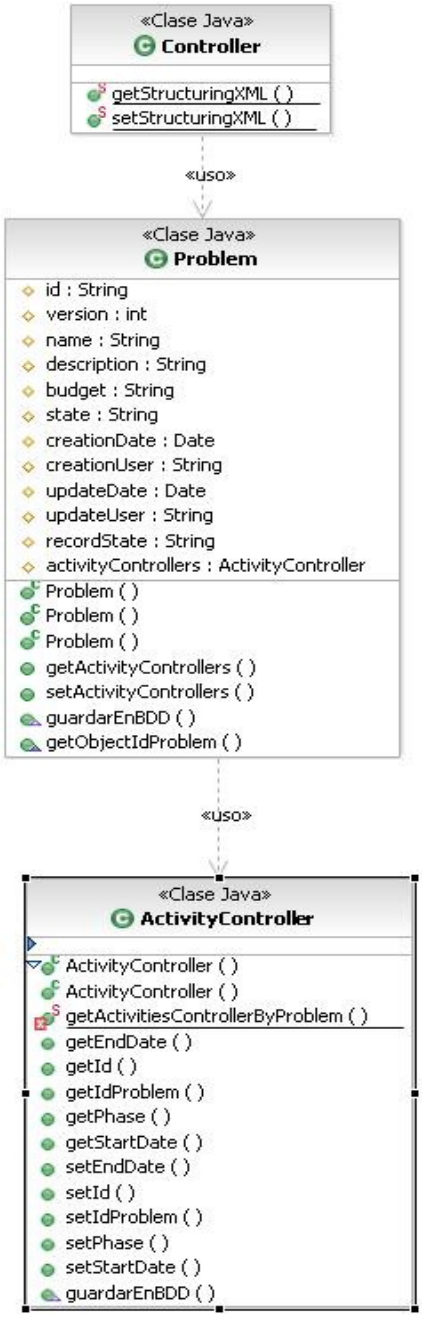

Ilustración 16: Diagrama de clase del Módulo Controlador

CONCLUSIONES Y MEJORAS

En este proyecto fin de carrera se ha trabajado en el campo de la participación electrónica, sus necesidades y las posibles soluciones que se le pueden dar a este tema. En nuestro caso se ha trabajado en un módulo de una aplicación mas extensa (NAML), la parte que ha cubierto este proyecto fin de carrera de esta aplicación incluye la obtención de datos de una base de datos y su transformación en un documento XML, y el tratamiento de un XML verificando su estructura con esquemas XSD y su posterior inserción en base de datos.

Como conclusiones principales se pueden obtener que la participación electrónica se puede volver un medio indispensable para el desarrollo de la sociedad actual hacia una sociedad mas participativa e involucrada en la democracia, ya que gracias a ella, las posibilidades de participación de todos los ciudadanos dentro de la sociedad, la política y la toma de decisiones dentro de cualquier ámbito en el que se quiera aplicar se elevan exponencialmente, remarcando sobre manera la posibilidad de que sean los propios ciudadanos los que decidan en que gastar sus impuestos.

También podemos llegar a la conclusión que, gracias a la polivalente y a la potencia que aporta el XML, y a la gran variedad de posibilidades inherentes a este lenguaje, se convierte en un lenguaje ideal para el intercambio de información entre las distintas partes que tiene la aplicación, aumentando la modularidad, y permitiendo en un futuro el intercambio de información entre esta aplicación y otras aplicaciones de muestreo o de informes.

Entre las mejoras que se pueden aplicar a este proyecto fin de carrera existen varias, como son:

- Mejora en la implementación de la base de datos. Evitando información redundante, así como almacenar información no útil.

- Implementar un esquema XSD que sea capaz de interpretar la información para cualquiera de los módulos. Unificando los distintos esquemas que se tienen actualmente en un único esquema.

BIBLIOGRAFIA

1. FONT, J. y BLANCO, I. (2003) "*Experiències de participaciò ciutadana. Polis, la ciutat participativa. Participar en els municipis: qui, com i per què?"* Barcelona: Centre per a la Participació Ciutadana, Organisme Autònom Flor de Maig, Diputació Barcelona. <http://www.diba.es/flordemaig/oafm/cpc/documents/Papers/09papers.pdf>

2. LÜHRS, R., ALBRECHT, S., et al. (2004) "Online Diskurse als Instrument politischer Partizipation - Evaluation der Hamburger Internetdiskussion zum Leitbild "Wachsende Stadt"", *Kommunikation@gesellschaft*, nº 5.

<http://www.soz.uni-frankfurt.de/K.G/B1_2004_L%FChrs_Albrecht_Hohberg_L%FCbcke.pdf>

3. Pedro Prieto Martín. Revista TEXTOS de la CiberSociedad ISSN 1577-3760 · Número 9 · Temática Variada. "Sistemas avanzados para la participación electrónica municipal: ejes conceptuales para su diseño".

<http://www.cibersociedad.net/textos/articulo.php?art=91>

4. Andres Nin Pérez. "Participación Electrónica: Ciudadanía y las reglas del juego"

<http://apunteselectronicos.wordpress.com/2009/03/01/%C2%BFcual-es-el-lugar-de-la-ciudadania-en-la-participacion-electronica/>

5. C. Alfaro, J.Gómez, J.M. Lavín, J.J. Molero (2009). A configurable architecture for e-Participatory budgeting support. Eighth International EGOV conference 2009.

6. C.Alfaro, J.Gómez, J.M. Lavín (2009). A configurable architecture for e-Participation support.

7. Keeney, R. and Raiffa, H (1976) Decisions with Multiple objectives: Preferences and Value Tradeoffs, Wiley.

8. French, S. and D. Rios Insua (2000) Statistical Decision Theory, Arnold.

9. Rios, J., Rios Insua, D. (2008) A methodology for participatory budget formation, Jour. Oper. Res. Soc., 59, 203-212.