UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Carrera de Ingeniero Informático

IMPLEMENTACIÓN, DESPLIEGUE Y ESTUDIO DE LA EFICIENCIA DE UNA LIBRERÍA DE GENERACIÓN AUTOMÁTICA DE PREGUNTAS

JOAQUÍN CAMPILLO MOLINA

Dirigido por: MANUEL LUQUE GALLEGO

Curso: 2011/2012. Convocatoria de Diciembre/Enero.

IMPLEMENTACIÓN, DESPLIEGUE Y ESTUDIO DE LA EFICIENCIA DE UNA LIBRERÍA DE GENERACIÓN AUTOMÁTICA DE PREGUNTAS

Proyecto de Fin de Carrera de modalidad *oferta específica*

Realizado por: JOAQUÍN CAMPILLO MOLINA

Dirigido por: MANUEL LUQUE GALLEGO

Tribunal calificador:

Presidente: D./D^{a}. .....
(firma)

Secretario: D./D^{a}. .....
(firma)

Vocal: D./D^{a}. .....
(firma)

Fecha de lectura y defensa: .....

Calificación: .....

Resumen

Las nuevas tecnologías están provocando cambios en muchos ámbitos, en particular están ayudando a una rápida extensión de la docencia a distancia a través de Internet. El Espacio Europeo de Educación Superior [EEES, 2012] considera al alumno el centro del proceso de aprendizaje y no la materia a impartir, por todo ello se plantea la necesidad de desarrollar herramientas capaces de generar actividades interactivas que estimulen al alumno, además de posibilitar la evaluación de sus avances académicos.

El proyecto presentado en esta memoria implementa una librería capaz de generar de forma automática y aleatoria un número ilimitado de preguntas tipo test de lógica proposicional y lógica de predicados de primer orden, en tiempo mínimo y predecible. Así mismo, se ha implementado en el entorno de evaluación de conocimientos SIETTE, al que podrán acceder los alumnos de las asignaturas " *Lógica y Estructuras Discretas*" y " *Lógica Computacional*", incluidas en las titulaciones de la UNED " *Grado en Ingeniería Informática* ", " *Grado en Ingeniería en Tecn*ologías *de la Información"* e " *Ingeniero en Informática*".

El software desarrollado es capaz de generar veintitrés modelos de preguntas tipo test, con sus respectivos conceptos teóricos y soluciones razonadas, incrementando de esta forma la interactividad y motivación del alumno.

El equipo docente también podrá utilizar la librería implementada para generar exámenes tipo test en su ordenador personal, con capacidad de ser incluida en páginas webs de creación propia.

Se ha primado la reusabilidad de los componentes, por tanto los conceptos relacionados con preguntas tipo test y lógica computacional podrían ser reutilizados en otros proyectos.

Finalmente, indicar que durante las distintas etapas del desarrollo se han explorado distintas posibilidades, llegando a soluciones técnicas y arquitectónicas, en particular relacionadas con SIETTE, que podrían ser de utilidad en futuros proyectos.

Palabras clave

Implementación

Despliegue

Eficiencia

Generación

Automática

Preguntas

Test

Lógica computacional

Lógica proposicional

Lógica de predicados de primer orden

Java

SIETTE

Ítems generativos

Librería

Abstract

Title

- Implementation, deployment and study of the efficiency of a library of automatic generation of questions.

New technologies are bringing about changes in many fields, in particular they are helping to a quick expansion of E- Learning by The Internet. The European Higher Education Area [EHEA, 2012] considers the student the center of the learning process and not subject to be taught, for all that there is a need to develop tools capable of create interactive activities that encourage students, in addition to enable the evaluation of their academic progress.

The project presented in this paper implements a library can automatically generate unlimited random and test questions of propositional logic and predicate logic of first order, in minimum time and predictable. It also has been deployed in the environment of knowledge assessment SIETTE, which students can access the courses "Logic and Discrete Structures" and "Computational Logic", included in the degrees of the UNED "Degree in Computer Engineering" "Degree in Information Technology" and "Computer Engineering".

The developed software is able to generate models twenty-three choice questions, with their theoretical concepts and reasoned solutions, thereby increasing interactivity and student motivation.

The teaching staff may also use the library implemented to generate multiple-choice exams on your personal computer, capable of being included in self-created websites.

The reusability of components has been given priority, so the concepts related to test questions and computational logic may be reused in other projects.

Finally, during the several stages of development, different technical and architectural solutions have been explored, particularly related to SIETTE, which could be useful in future projects.

Keywords

Implementation

Deployment

Efficiency

Generation

Automatic

Questions

Test

Computational logic

Propositional logic

Predicate logic of first order

Java

SIETTE

Generative items

Library

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1. Motivación

Son muchas las universidades, incluidas las de tipo presencial, así como centros de formación, la administración pública y empresas privadas que utilizan la docencia a distancia, en su modalidad e-learning, para formar a sus alumnos y empleados. Un buen ejemplo es la Universidad Nacional de Educación a Distancia [UNED, 2012] considerada como la universidad con más alumnos de España y su número va en aumento cada curso.

Existe una característica fundamental en la formación a distancia, la interactividad por parte del alumno tanto en el acceso a los contenidos, como en la ejecución de tareas que le estimulen y ayuden a evaluar su nivel de conocimientos. El Espacio Europeo de Educación Superior [EEES, 2012] hace énfasis en una nueva metodología, en la que el alumno es considerando el centro del proceso de aprendizaje y no la materia a impartir, permitiendo al estudiante poner en práctica todo lo aprendido. El uso de casos reales, trabajo colaborativo, campus online y otras metodologías innovadoras son la norma en un entorno en el que la clase magistral tradicional es la excepción. Un sistema de evaluación continua, en el que el conjunto de actividades y trabajos que realice el alumno durante el curso cobrarán mayor importancia frente al examen final tradicional. Por tanto el aprendizaje pasa a ser eminentemente activo, donde el alumno debe realizar gran cantidad de actividades y trabajos de manera autónoma, ayudado por la flexibilidad que aporta el estudiar a distancia desde cualquier lugar y momento del día.

Los hechos a los que se hacen referencia (auge de la docencia a través de Internet, el número creciente de alumnos de la UNED junto con un mayor número de actividades a realizar por cada uno de ellos) han desembocado en que los equipos docentes de las asignaturas les resulte cada vez más difícil y les lleve un mayor tiempo crear actividades para permitir que los alumnos consigan los objetivos del aprendizaje de una manera más eficaz. En muchos casos las tareas son repetitivas , por tanto se pueden automatizar, en especial en marco de las asignaturas de tipo matemático. Necesitándose herramientas capaces de generar actividades de forma automática, liberando al profesor de estas tareas repetitivas.

Con esta motivación surge la idea de crear una herramienta capaz de generar de forma automática un número ilimitado de preguntas tipo test de los temas lógica proposicional y lógica de predicados de primer orden, impartidas en las asignaturas " *Lógica y Estructuras Discretas*" y " *Lógica Computacional*", incluidas en las titulaciones de la UNED " *Grado en Ingeniería Informáti-*

ca “, “ *Grado en Ingeniería en Tecnologías de la Información”* e “ *Ingeniero en Informática*”.

Las áreas de conocimiento elegidas son de naturaleza matemática, donde los problemas se encuadran en un marco formal; suponiendo que será más fácil generar preguntas tipo test de forma automática que en otras asignaturas más teóricas, aunque no por ello exentos de retos, como el uso exhaustivo de notación matemática o el suministro de la información suficiente para ayudar al alumno a entender los razonamientos de las preguntas propuestas, todo ello en un tiempo mínimo y predecible, independientemente de la complejidad de los algoritmos utilizados para este fin.

1.2. SIETTE como ejemplo de uso

Una vez obtenida la librería capaz de generar las preguntas tipo test y respuestas razonadas objeto de este proyecto, el profesor se encuentra con la necesidad de distribuirlas a los estudiantes. Un entorno que se adapta a estos requisitos es SIETTE.

SIETTE es un sistema web de evaluación de conocimientos que permite la creación y mantenimiento de bancos de preguntas, así como la realización de exámenes tipo test. Entre sus características más relevantes son de especial importancia en este proyecto su fácil acceso vía Internet y la posibilidad de utilizar ítems generativos [ItemGenerativo, 2011].

El poder utilizar ítems generativos permitirá al profesor utiliza una plantilla en un test, como si fuera como un ítem más. Las plantillas se implementan mediante lenguajes embebidos en HTML, tales como JSP o PHP, por tanto toda la lógica de la generación de las preguntas y respuestas podría recaer en la librería implementada en un lenguaje de programación, como por ejemplo Java.

La posibilidad de poder agregar ítems generativos será sencilla, de forma que cuando un profesor quiera insertar un ítem generativo, sólo deberá seleccionar el tipo de ítem, al igual que los no generativos, ocultando la complejidad subyacente.

Quizá una traba pudiera ser la configuración de las respuestas autocorregidas y ayudas, ya que los profesores deberían tener ciertos conocimientos de programación para construir correctamente las plantillas, algo que también viene a subsanar éste proyecto. Unas plantillas de fácil configuración junto a una guía de despliegue sencilla reducirán al mínimo los conocimientos necesarios para su correcto uso.

1.3. Objetivos

Se exponen a continuación los **objetivos principales** del presente proyecto:

- Una librería (o conjunto de librerías) que permita generar preguntas tipo test de forma automática, respecto a las asignaturas ” *Lógica y Estructuras Discretas*“ y “*Lógica Computacional*”, incluidas en las titulaciones de la UNED “ *Grado en Ingeniería Informática* “, “ *Grado en Ingeniería en Tecnologías de la Información”* e “ *Ingeniero en Informática*”. Centrándose en temas *lógica proposicional* y *lógica de predicados de primer orden*.

- Como hecho medible que demuestre el funcionamiento adecuado de dicha librería se creará una asignatura de prueba en el sistema de evaluación de conocimientos SIETTE.

Otros **objetivos implícitos** a tener en cuenta son los siguientes:

- Establecer un modelo a seguir respecto a preguntas de tipo test. Además tener un enunciado y respuestas, deberá establecer las estructuras necesarias para gestionar conceptos teóricos, e indicar si las respuestas son verdaderas o falsas, así como un razonamiento de estos hechos.
- Debido al dominio de las preguntas elegido (lógica proposicional y lógica de predicados de primer orden), donde los problemas se encuadran en un marco formal, se implementarán los conceptos básicos de este área del conocimiento, con el ánimo de poder reutilizarlos en otros proyectos.
- Como consecuencia del punto anterior, será importante la búsqueda de programas o librerías que pudieran ayudar a implementar los conceptos básicos de lógica proposicional y lógica de predicados de primer orden.
- Los algoritmos utilizados para generar tanto las preguntas, como la solución deberán llevarse a cabo en tiempos razonables, siendo inaceptables en el caso de ser superiores a un segundo, de otra forma la interacción con el alumno se vería afectada negativamente.
- La cantidad de preguntas generadas debe ser ilimitada.
- Para aquellos alumnos que no pudieran acceder a los entornos de evaluación, se dará la posibilidad al profesor de poder generar colecciones de test en formatos fácilmente exportables, tales como LaTeX .
- La naturaleza de las preguntas incluye notación matemática, por tanto, se deberá buscar los formatos de salida más adecuados, capaces de ser mostrados tanto en entornos locales, como en entornos de evaluación de conocimientos.
- En general, es posible que el equipo de desarrollo, pudiera no tener a su alcance un servidor SIETTE para realizar pruebas antes de la implantación final. Se darán las pautas que permitan programar (incluyendo distintas particularidades de SIETTE) y probar librerías basadas en Java, de forma que cuando sean implantadas por parte de un administrador de sistemas, se haga con la seguridad de su correcto funcionamiento.
- Se realizará una guía sencilla y precisa de generación de ítems generativos soportados por la librería implementada, de forma que facilite reducir al mínimo tanto el trabajo administrativo, como los conocimientos de programación del equipo docente.

1.4. Descripción del resto de capítulos

En este capítulo 1 se ha pretendido ambientar al lector respecto a la motivación del proyecto y los objetivos propuestos, ilustrados con un ejemplo de uso (SIETTE).

El capítulo 2 está centrado en la gestión del proyecto, en el que se describen los conceptos utilizados, se realiza una estimación de costes y esfuerzos utilizando COCOMO II, para después

organizar las tareas mediante un diagrama de Gantt, detallando a continuación las herramientas utilizadas.

En el capítulo 3 se ha realizado un estudio de viabilidad, mostrando los requisitos funcionales y no funcionales, desde una perspectiva general, para después pasar a detallar distintas opciones respecto a arquitecturas y herramientas, razonando el motivo de su aceptación o descarte y la solución elegida.

El capítulo 4 se centra en describir los fundamentos teóricos en los que se basa el resto de proyecto, dividiéndose en tres grupos: lógica computacional, exámenes tipo test y patrones de diseño utilizados. Así mismo se estudia el comportamiento de Prover9 y Mace4, que como se podrá comprobar son herramientas clave en la construcción de la librería.

Debido al enfoque ingenieril con el que se han tratado las distintas etapas del desarrollo, en el capítulo 5 y 6, se muestran las fases de análisis y diseño. En estos capítulos basándose en el paradigma de la programación orientada a objetos, se pretende establecer las estructuras necesarias para la posterior construcción de exámenes tipo test en general, particularizando en los tipos de preguntas tratadas en este proyecto (lógica proposicional y lógica de predicados de primer orden), utilizando para este fin diagramas UML [UML, 2012].

Después del análisis y diseño, en el capítulo 7 se detallan los dos módulos construidos (evaluador y generador). Es importante destacar la descripción realizada, a tener en cuenta por parte del analista y programador, respecto a las particularidades de SIETTE a la hora de implementar ítems generativos basados en Java, considerándose de lectura recomendada para aquellos que den sus primeros pasos en la construcción de librerías con características similares a la descrita en este proyecto.

En el capítulo 8 se demuestra la consecución de los objetivos propuestos, describiendo las pruebas realizadas, descripción de la cantidad de preguntas distintas capaz de generar el sistema, un estudio de tiempos y una página JSP que utiliza todos los modelos de preguntas (requisito previo a la implantación en SIETTE).

En el capítulo 9 expone las conclusiones, mejoras y las posibles líneas de trabajo futuras.

Por último, se recomienda la lectura de los Apéndices A y B a aquellos usuarios que estuvieran interesados en la implantación y uso del producto final obtenido.

2. Gestión del proyecto

2.1. Conceptos generales

Debido a la naturaleza del proyecto se ha implementado según el **paradigma orientado a objetos**.

En el proceso de **desarrollo de software** se han seguido las pautas marcadas por el **proceso unificado de Rational** [ADMS, 2012], es decir el proyecto se ha ido dividiendo en una serie de partes o mini-proyectos, considerándose cada uno como una iteración.

El **ciclo de vida** de desarrollo de software utilizado ha sido desarrollo **iterativo e incremental** de forma que a medida que se aprenden nuevos conceptos se implementan en prototipos, comprobando que se adaptan a las necesidades planteadas.

La **metodología** utilizada se ha basado **Métrica v.3** [MetricaV3, 2012].

2.2. Estudio de costes y esfuerzo

Para la realizar el estudio de costes se utilizará el Modelo Constructivo de Costes II (o COCOMO II, acrónimo del inglés COnstructive COst MOdel II).

Se tendrán en cuenta los siguientes aspectos:

- Se utilizará el **modelo post-arquitectura**, debido a que es el modelo de estimación más detallado y la arquitectura del proyecto está definida completamente definida en el momento de realizar los cálculos.
- Se ha dividido en **cuatro módulos**: Preguntas Tipo Test, Conceptos Lógica Computacional, Generación Preguntas Lógica Computacional, Visualización Preguntas.

2.2.1. Líneas de código fuente

Para la estimación del tamaño de software se ha considerado como parámetro líneas estándar de código (SLOC):

Preguntas Tipo Test: 600 SLOC

Conceptos Lógica Computacional: 2100 SLOC

Generación Preguntas Lógica Computacional: 3000 SLOC

Visualización Preguntas: 300 SLOC

En todos los módulos se considera un Desperdicio de Código (Breakage) del 0 %, por considerarse los requisitos como no volátiles.

Respecto al sueldo mensual se han utilizado los siguientes en función de los perfiles: 3.700 € para el analista y 2.900 € para el programador semisenior. Por tanto, si consideramos que un 25 % del tiempo será trabajo de análisis y diseño, mientras que un 75 % serán labores de programación, finalmente se utilizará un sueldo medio ponderado de 3.100 €.

2.2.2. Factor exponencial de escala

Precedencia (PREC): Bajo
Flexibilidad en el desarrollo (FLEX:) Alto
Arquitectura/ Resolución de riesgo (RESL): Alto
Cohesión de equipo (TEAM): Extra
Madurez del proceso (PMAT): Bajo

2.2.3. Factores multiplicadores de esfuerzo

Factores (Effort Multipliers EM)	**Preguntas Tipo Test**	**Conceptos Lógica Computacional**	**Generación Preguntas Lógica Computacional**	**Visualización Preguntas**
RELY (Confiabilidad requerida)	Bajo	Bajo	Bajo	Bajo
DATA (Tamaño de la base de datos)	Bajo	Bajo	Bajo	Bajo
CPLX (Complejidad del producto)	Nominal	Alto	Muy alto	Bajo
RUSE (Requerimientos de reusabilidad)	Muy Alto	Muy Alto	Nominal	Bajo
DOCU (Documentación ciclo de vida)	Nominal	Nominal	Nominal	Nominal
TIME (Restricción tiempo ejecución)	Alto	Alto	Alto	Alto
STOR (Restricción almacenamiento)	Nominal	Nominal	Nominal	Nominal
PVOL (Volatilidad de la plataforma)	Bajo	Bajo	Bajo	Bajo
ACAP (Capacidad del analista)	Muy Alto	Muy Alto	Muy Alto	Muy Alto
PCAP (Capacidad del programador)	Muy Alto	Muy Alto	Muy Alto	Muy Alto
PCON (Continuidad del personal)	Nominal	Nominal	Nominal	Nominal
AEXP (Experiencia en la aplicación)	Bajo	Bajo	Bajo	Muy Alto
PEXP (Experiencia en la plataforma)	Alto	Bajo	Nominal	Nominal
LTEX (Experiencia herramientas)	Alto	Bajo	Nominal	Nominal
TOOL (Uso de herramientas)	Alto	Bajo	Alto	Alto
SITE (Ubicación Espacial)	Nominal	Nominal	Nominal	Nominal
SITE (Comunicación)	Muy Alto	Muy Alto	Muy Alto	Muy Alto
SCED (Cronograma desarrollo)	Nominal	Nominal	Nominal	Nominal

Tabla 2.1: Estimación de costes. Multiplicadores de esfuerzo.

2.2.4. Resultados y conclusiones

Aplicando los algoritmos propuestos por COCOMO II y los parámetros expuestos en los puntos anteriores obtenemos los siguientes resultados desglosados por módulos:

Factores	**Preguntas Tipo Test**	**Conceptos Lógica Computacional**	**Generación Preguntas Lógica Computacional**	**Visualización Preguntas**
Tamaño (SLOC)	600	2.100	3.000	300
Salario (€ / mes)	3.100,00	3.100,00	3.100,00	3.100,00
Factor Ajuste Tiempo (EAF)	0,38	0,77	0,53	0,24
Esfuerzo Nominal (meses/persona (PM))	2,00	7,00	10,00	1,00
Esfuerzo Estimado (meses/persona (PM))	0,80	5,40	5,30	0,20
Productividad	792,30	389,70	563,10	1.239,70
Coste (€)	2.347,69	16.705,10	16.517,15	750,18
Coste por Instrucción (€)	3,90	8,00	5,50	2,50
Desarrolladores	0,10	0,70	0,70	0,00
Nivel de Riesgo	1,00	2,00	0,00	0,00

Tabla 2.2: Estimación de costes. Resultados.

Podemos resumir los datos anteriores agrupando los resultados de los cuatro módulos, así mismo se presentan tres predicciones una optimista, otra más posible y por último la pesimista.

Parámetros	**Optimista**	**Más probable**	**Pesimista**
Esfuerzo (meses/persona - PM)	9,40	11,70	14,60
Tiempo (meses)	7,40	7,90	8,50
Productividad	640,10	512,10	409,70
Coste (€)	29.056,10	36.320,13	45.400,16
Coste por Instrucción (€)	4,80	6,10	7,60
Desarrolladores	1,30	1,50	1,70
Nivel de Riesgo	*******	3,00	*******

Tabla 2.3: Estimación de costes. Predicciones.

Como **conclusión**, para desarrollar la **6.000 líneas de código** del presente proyecto deberemos asignar **un analista (a tiempo parcial) y un programador (a tiempo completo)** durante **8 meses**, con una jornada laboral de **152 horas al mes**, generando un **coste de 36.320,13 €**.

2.3. Diagrama de Gantt

Tareas		**Fecha de inicio**	**Fecha de fin**	**Duración (Días)**
Preguntas Tipo Test		05/01/2012	13/01/2012	6
	Estudio Viabilidad	05/01/2012	06/01/2012	1
	Análisis	06/01/2012	07/01/2012	1
	Diseño	09/01/2012	10/01/2012	1
	Construcción	10/01/2012	11/01/2012	1
	Implantación	11/01/2012	12/01/2012	1
	Documentación	12/01/2012	13/01/2012	1
Conceptos Lógica Computacional		10/01/2012	31/05/2012	102
	Estudio Viabilidad	10/01/2012	11/01/2012	1
	Análisis	13/01/2012	24/01/2012	7
	Diseño	24/01/2012	18/02/2012	19
	Construcción	20/02/2012	28/04/2012	50
	Implantación	30/04/2012	26/05/2012	20
	Documentación	28/05/2012	31/05/2012	3
Generación Preguntas Lógica Computacional		20/02/2012	06/09/2012	143
	Estudio Viabilidad	20/02/2012	21/02/2012	1
	Análisis	21/02/2012	01/03/2012	7
	Diseño	01/03/2012	28/03/2012	19
	Construcción	28/05/2012	04/08/2012	50
	Implantación	06/08/2012	01/09/2012	20
	Documentación	03/09/2012	06/09/2012	3
Visualización Preguntas		28/03/2012	07/09/2012	117
	Estudio Viabilidad	28/03/2012	29/03/2012	1
	Análisis	29/03/2012	30/03/2012	1
	Diseño	30/03/2012	31/03/2012	1
	Construcción	03/09/2012	05/09/2012	2
	Implantación	05/09/2012	06/09/2012	1
	Documentación	06/09/2012	07/09/2012	1

Tabla 2.4: Diagrama de Gantt. Programación de tareas.

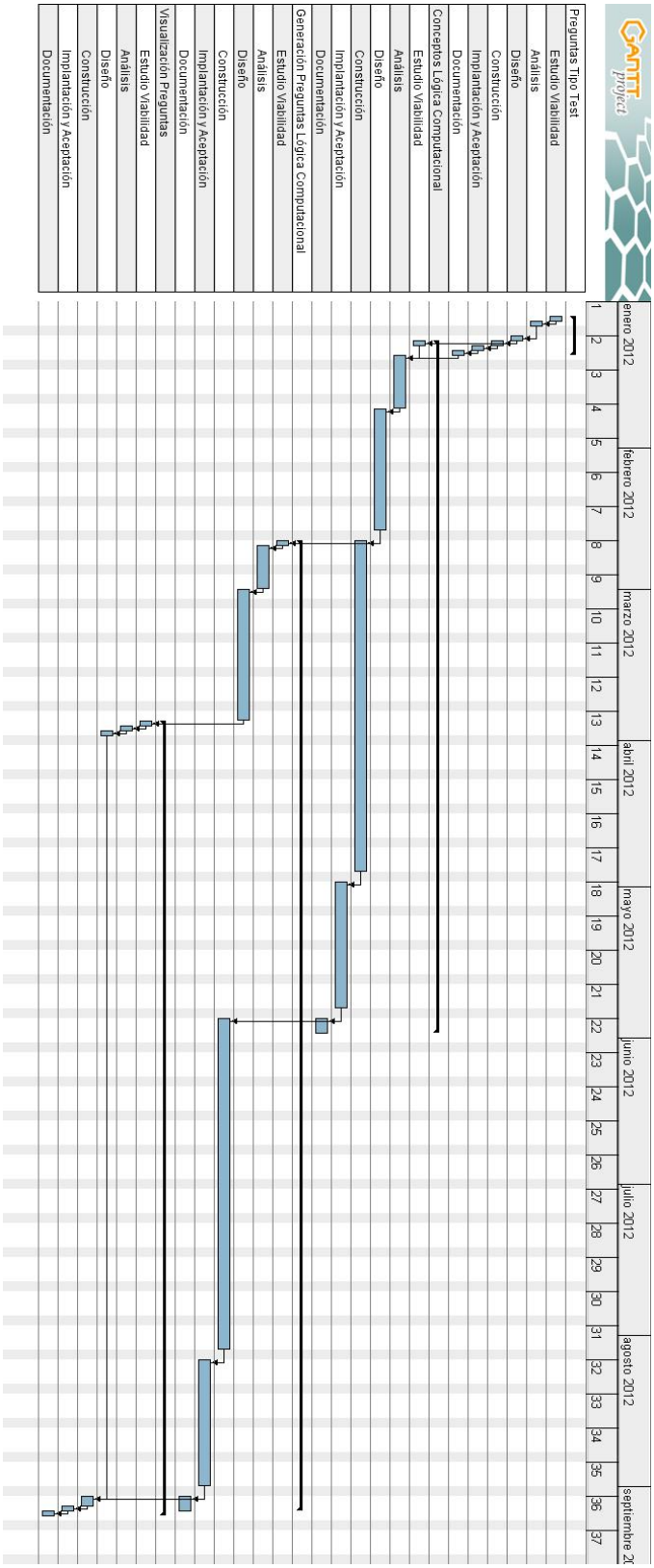


Figura 2.1: Gestión del Proyecto. Diagrama de Gantt.

2.4. Herramientas utilizadas

Destacar que el proyecto se ha realizado en su totalidad utilizando herramientas de **software libre**, entre las que cabe destacar:

- **JDK 1.6** [Oracle, 2012]: Kit de Desarrollo del lenguaje Java.
- **Eclipse 3.5 (Eclipse Galileo)** [Eclipse.org, 2012]: Utilizado como entorno de desarrollo integrado para la construcción, implementación y pruebas de las librerías propuestas en este proyecto, mediante el uso del lenguaje orientado a objetos Java.
- **JUnit 4** [JUnit, 2012]: Utilizado para la realización de módulos de prueba, así como cálculo de estadísticas.
- **Ant 1.8.2** [Ant, 2012]: Utilizado como herramienta para la automatización de las tareas de compilación del software desarrollado.
- **Apache Tomcat 6.0** [Tomcat, 2012]: Utilizado como servidor de aplicaciones JavaServer Pages en los procesos de prueba de la librería antes de su despliegue en el entorno de evaluación web (SIETTE).
- **StarUML 5.0** [StarUML, 2012]: Utilizado como diagramador UML.
- **GanttProject 2.0.10** [GanttProject, 2012]: Utilizado para crear diagramas de Gantt.
- **COCOMO II.2000.0** [Engineering, 2012]: Utilizado para la estimación de costes.
- **Bitbucket** [Bitbucket, 2012]: Utilizado como sistema de control colaborativo de versiones con doble finalidad: compartir ficheros con los administradores de SIETTE y gestionar las distintas versiones del proyecto.
- **MercurialEclipse** [JavaForge, 2012]: Utilizado como sistema de control de versiones integrado en Eclipse y compatible con Bitbucket.
- **Lyx 2.0** [Lyx, 2012]: Utilizado como editor de documentos para la escritura de la presente memoria.
- **JabRef 2.7** [JabRef, 2012]:Utilizado como administrador de referencias y bibliografía, fácilmente integrable en Lyx 2.0.
- **Evernote 3.1** [Evernote, 2012]: Utilizado como administrador de notas, apuntes y correos electrónicos.
- **Dropbox 1.2** [Dropbox, 2012]: Utilizado para almacenar y compartir ficheros con el equipo docente.

3. Estudio de viabilidad

A continuación se estudiarán distintas opciones arquitectónicas y tecnológicas para la consecución de un software que cumpla con los objetivos propuestos en la sección 1.3 Objetivos .

Una buena fuente para el estudio de los posibles modelos de preguntas que se pudieran generar son los exámenes correspondientes a las convocatorias de las asignaturas " *Lógica y Estructuras Discretas*“ y “ *Lógica Computacional*”, incluidas en las titulaciones de la UNED “ *Grado en Ingeniería Informática* “, “ *Grado en Ingeniería en Tecnologías de la Información*” e “ *Ingeniero en Informática*”. Se pueden encontrar todos los exámenes realizados de dicha asignatura en el depósito de exámenes de UNED Aragón [UNEDAragon, 2012].

3.1. Definición de requisitos

En esta sección se enumeran los requisitos funcionales y no funcionales desde una perspectiva de alto nivel.

3.1.1. Requisitos funcionales

1. El sistema deberá ser capaz de generar automáticamente preguntas tipo test de álgebra proposicional y álgebra de predicados de primer orden.

2. Las preguntas se adaptarán a los modelos presentados en los exámenes de las asignaturas " *Lógica y Estructuras Discretas*“ y “ *Lógica Computacional*”, incluidas en las titulaciones de la UNED “ *Grado en Ingeniería Informática* “, “ *Grado en Ingeniería en Tecnologías de la Información*” e “ *Ingeniero en Informática*”.

3. Además de las preguntas tipo test, el sistema devolverá al alumno una solución, informando si una respuesta es verdadera o falsa, así como un razonamiento de estos hechos.

4. El tiempo de generación de cada pregunta deberá ser predecible e inferior a un segundo .

5. El profesor de la asignatura podrá generar un número ilimitado de preguntas con el menor trabajo administrativo posible.

6. Los alumnos podrán acceder a los exámenes tipo test mediante Internet.

3.1.2. Requisitos no funcionales

1. El lenguaje de programación será Java.
2. Como producto final se obtendrá una librería (o conjunto de librerías) reutilizable en entornos Java.
3. Demostrar el correcto funcionamiento de la librería obtenida mediante pruebas de unitarias, integración y sistema.
4. Implementación de una página JSP que demuestre su correcto funcionamiento en un entorno web.
5. Implementar al menos un test de prueba en SIETTE que demuestre que la librería implementada funciona adecuadamente.
6. El sistema debe ser multiplataforma desde las perspectivas de profesores, alumnos y entorno de evaluación.

3.2. Alternativas a la solución

De la lectura de las características de SIETTE se desprende la posibilidad creación de plantillas para la generación de preguntas tipo test en tiempo real, dichas plantillas se implementan mediante lenguajes embebidos en HTML, tales como JSP o PHP [ItemGenerativo, 2011]. Por este motivo, la librería se construirá utilizando el **lenguaje de programación Java**, para posibilitar su integración mediante **JSP** en una asignatura de **SIETTE**.

Al ser Java un lenguaje ampliamente usado por la comunidad de programadores, facilitará la reutilización de otras librerías o programas, ya existentes, capaces de suministrar las funcionalidades necesarias para la gestión de los conceptos de lógica proposicional y lógica de predicados de primer orden.

Una parte importante de este proyecto será la reutilización de otros programas o librerías que permitan el uso de fórmulas proposicionales y lógica de predicado de primer orden, que convenientemente tratadas pudiera utilizarse para devolver al alumno un razonamiento de las respuestas correctas y erróneas . En este ámbito se han evaluado dos herramientas:

- Prover9 / Mace4 [Prover9Mace4, 2009]: son dos programas de software libre que se distribuyen conjuntamente. Prover9 permite probar de forma automatizada teoremas de lógica proposicional y lógica de predicados de primer orden. Mientras que Mace4 es capaz de buscar modelos finitos y contraejemplos.
- Orbital [Platzer, 2011]: es una librería implementada en Java que suministra, entre otras posibilidades, algoritmos para lógica, álgebra, algoritmos matemáticos, y pruebas de teoremas.

La elección del software base para los conceptos de lógica se han implementado sendos prototipos. Estimándose como requisitos mínimos que la librería o programa sea capaz de:

1. Deducir si un conjunto de fórmulas proposicionales es satisfacible o insatisfacible.

2. Deducir si un conjunto de fórmulas de predicados de primer orden es satisfacible o insatisfacible.

3. Devolver una interpretación verdadera en el caso de que un conjunto de fórmulas proposicionales sea satisfacible.

4. Devolver una interpretación verdadera en el caso de que un conjunto de fórmulas de predicados de primer orden sea satisfacible.

5. Devolver una demostración en el caso de que un conjunto de fórmulas proposicionales sea insatisfacible.

6. Devolver una demostración en el caso de que un conjunto de fórmulas de predicados de primer orden sea insatisfacible.

Mediante el uso de Prover9 / Mace4 se consiguieron los requisitos expuestos, aunque tienen el inconveniente de ser programas independientes que previamente deben ser instalados y configurados.

Respecto a Orbital, a pesar de que se pueden realizar inferencias sobre un grupo de fórmulas e interpretaciones, cuando se ha intentado utilizar de forma masiva los tiempos de obtención de resultados eran peores que con Prover9/Mace4, pero el problema más importante se ha presentado al intentar conseguir interpretaciones para lógica de predicado de primer orden, punto que no ha sido posible subsanarse. Por estos motivos, finalmente se ha decidido utilizar **Prover9/Mace4** como s**oftware base para la gestión de conceptos de lógica**, aunque en otros proyectos debería tenerse en cuenta Orbital como opción muy interesante.

Se van a implementar preguntas del ámbito de la lógica lo que conlleva una la búsqueda de los formatos de salidas oportunos para representar los símbolos y notación correspondiente a esta área de conocimiento. Además, el formato de salida debe cumplir con el requisito de poderse integrar en páginas JSP y más concretamente en SIETTE. Se han evaluado dos librerías:

- MathJax [MathJax, 2012]: es una biblioteca javascript que permite visualizar mediante navegadores web contenidos matemáticos generados en formato LaTeX [LaTeX, 2011], por tanto, servirá como soporte para poder implementar contenidos web, como por ejemplo páginas JSP, sin necesidad de instalación de componentes adicionales en los navegadores de los alumnos.

- MathML [MathLM, 2012]: es un lenguaje de marcas basado en XML cuyo objetivo es expresar notación matemática de forma que puedan intercambiarse notaciones matemáticas entre distintos sistemas. También es posible utilizarlo directamente en páginas web o previa instalación del complemento "*MathPlayer*".

Se ha decidido utilizar **MathJax** por la sencilla razón de que SIETTE soporta la notación generada en LaTeX mediante el uso interno de MathJax. No obstante, se ha probado la notación MathML, que no debe descartarse en otro tipos de proyectos, en especial aquellos que requieren una notación para el intercambio de fórmulas matemáticas, también se han realizado pruebas para visualizar

contenidos MathML, desde distintos navegadores, llegando a la conclusión que en alguno de ellos (por ejemplo Internet Explorer 7) requieren la instalación de componentes adicionales.

Debido a que MathJax requiere notación **LaTeX**, la librería deberá ser capaz de generar salidas en este formato, que al mismo tiempo puede ser útil con mínimo esfuerzo, para ser exportado a otros formatos, como pudiera ser **PDF**.

Otro requisito que puede condicionar la arquitectura de la aplicación son lo tiempos máximos en generar la pregunta de forma automática, así como las respuestas, incluyendo la información en la que se detalla las razones para determinar si éstas son correctas o falsas. Por tanto, se debe tener en cuenta los escenarios en los que será posible utilizar la librería construida:

1. En el caso que el profesor genere un test en formato LaTex o PDF, para posteriormente distribuirlo, el tiempo de generación de las preguntas no es crítico.

2. En el caso de que el alumno solicita la generación de exámenes tipo test en tiempo real, los tiempos de generación de las preguntas es crítico, considerándose inaceptable si es superior a un segundo, independientemente de la complejidad del tipo pregunta, además sería conveniente poder predecir dichos tiempos.

En ambos casos se podrían utilizar las mismas estructuras de datos, en el primero se generarían cada vez que se invocará la generación del examen, siendo los tiempos de generación imprevisibles, sin embargo en el segundo se podrían generar las estructuras necesarias, quedando almacenados en un fichero de datos preprocesados, de este modo cuando se requiriera la generación de una pregunta se cargaría dicha información y se utilizaría con la certeza de que dichas estructuras son las que se necesitan, evitando el uso de la librería o software base gestora de conceptos lógica. Por tanto, estaríamos hablando de dos arquitecturas distintas:

- Monolítica: El mismo subsistema que genera las estructuras de datos para un examen, generaría las preguntas.

- Distribuida: Debe existir un subsistema capaz de generar las estructuras de datos, que almacenará en un fichero de datos preprocesados, otro subsistema será el encargado de seleccionar de forma aleatoria las estructuras de datos preprocesados, a partir de las cuáles se generarían las preguntas.

Se utilizarán ambas arquitecturas, dependiendo del uso que se le dé, en el caso de que el profesor generé un examen completo, para después distribuirlo, en formato LaTeX o PDF se utilizará una arquitectura **monolítica**, pero si se necesitará utilizar en un aplicación web o entorno de evaluación, por ejemplo SIETTE, en el que alumno solicita la generación de una pregunta en tiempo real se utilizará una arquitectura **distribuida.**

Respecto a la estructuras de datos, a partir de las cuáles se generaran las preguntas, dado que el lenguaje será Java, podrían ser distintas listas (grupos de fórmulas satisfacibles, fórmulas insatisfacibles,...), se describirán a lo largo de esta memoria, a partir de fórmulas proposicionales o fórmulas de predicados de primer orden suministradas por el profesor y que cumplan ciertos requisitos que aseguren la generación de cualquiera de los modelos de preguntas. Por ejemplo, la estructura de datos para un examen de lógica proposicional o lógica de predicados de primer

orden, a partir de las fórmulas propuestas por el profesor, se deberían buscar cuatro fórmulas proposicionales que agrupadas de tres en tres tuvieran al menos dos grupos de fórmulas satisfacibles, dos grupos de fórmulas insatisfacibles, una de las fórmulas seleccionadas tuviera al menos otra equivalente. A la hora de generar las preguntas se dispondrán de los grupos de fórmulas que con toda seguridad cumplen con los requisitos de todos los modelos de preguntas.

El formato del fichero en el que se almacenarán las estructuras de datos preprocesados serán **XML**, otras alternativas posibles a este formato serían:

- Ficheros de texto: El lenguaje de programación que utilizará (Java) dispone de librerías que facilitan la gestión de de ficheros XML, por ello si se utilizarán ficheros de texto conllevaría la implementación de ciertas funcionalidades ya implementadas en librerías especializadas.
- Bases de datos: No podemos asegurar que vaya a existir un gestor de base de datos en los servidores web o los entornos de evaluación en los que se vaya a utilizar.

Se ha decidido utilizar **JDOM** [**?**] como librería de apoyo en el tratamiento de ficheros XML; el motivo de esta elección se debe su compatibilidad con Java y por ser software libre.

Por tanto, para buscar la mejor solución se deben tener en cuenta los siguientes atributos:

- Entorno de evaluación de conocimientos.
- Lenguaje de programación.
- Software o librería base reutilizable capaz de gestionar conceptos de lógica proposicional y lógica de predicados de primer orden.
- Arquitectura.
- Formato de salida.
- Formato de almacenamiento de datos preprocesados.
- Software o librería reutilizable capaz de gestionar los datos preprocesados.

Se darán las siguientes valoraciones para cada atributo de las posibles alternativas:

- 1: No tiene ningún beneficio.
- 2: Es buena pero el beneficio no es notable.
- 3: Es óptima con unos beneficios notables.

		VALORACIÓN		
ATRIBUTOS	OBSERVACIONES	1	2	3
Evaluación de Conocimientos				
SIETTE	Requisitos del proyecto sin alternativa posible			X
Fichero en formato LaTeX	Requisitos del proyecto sin alternativa posible			X
Lenguaje de programación				
Java	Requisitos del proyecto sin alternativa posible			X
Software Lógica				
Prover 9. Mace 4	Requiere instalación previa		X	
Orbital	No se ha conseguido realizar interpretaciones con lógica de primer orden	X		
Arquitectura				
Monolítica	Generación de exámenes en formato LaTeX			X
Distribuida	Utilizado en la generación de exámenes en SIETTE			X
Formato de Salida				
MathJax	Necesaria para en SIETTE o entornos web			X
PDF	Exportación desde la salida en formato LaTeX		X	
LaTeX	Soporte a las salidas en formato pdf y MathJax			X
MathML	Útil en entornos web, no es de aplicación en SIETTE	X		
Formato datos preprocesados				
Ficheros de Texto	Conlleva la implementación de la gestión de los datos preprocesados		X	
Bases de Datos	Dependencia del entorno de evaluación	X		
XML	Existen librerías reutilizables			X
Software gestión datos preprocesados				
JDOM	Librería muy extendida para la gestión del ficheros XML en Java			X

Tabla 3.1: Estudio de viabilidad. Alternativas a la solución.

3.3. Solución elegida

Como conclusión al estudio de las alternativas a la solución, finalmente se ha elegido la siguiente solución:

Se implementarán tanto una **arquitectura monolítica**, para la generación de colecciones de preguntas para su posterior distribución en formato LaTeX o PDF y otra **distribuida** para poder incluirlo en SIETTE o páginas JSP.

Los formatos de las salidas necesarios serán **LaTeX** y **MathJax**.

Debido a los requisitos de la plataforma de evaluación de conocimientos el lenguaje de programación será **Java**.

Como software reutilizable para la gestión de los conceptos de lógica proposicional y lógica de predicados de primer orden optaremos por **Prover9 / Mace4**. Está decisión implica la capacidad de interpretar los datos preprocesados sin que sea necesaria la instalación de este software en el servidor de evaluación de conocimientos.

Como formato de almacenamiento de datos procesados se optará por **XML**, y como librería de gestión **JDOM**.

En la siguiente figura, se deja patente como se integrarán los distintos componentes respecto a la solución elegida.

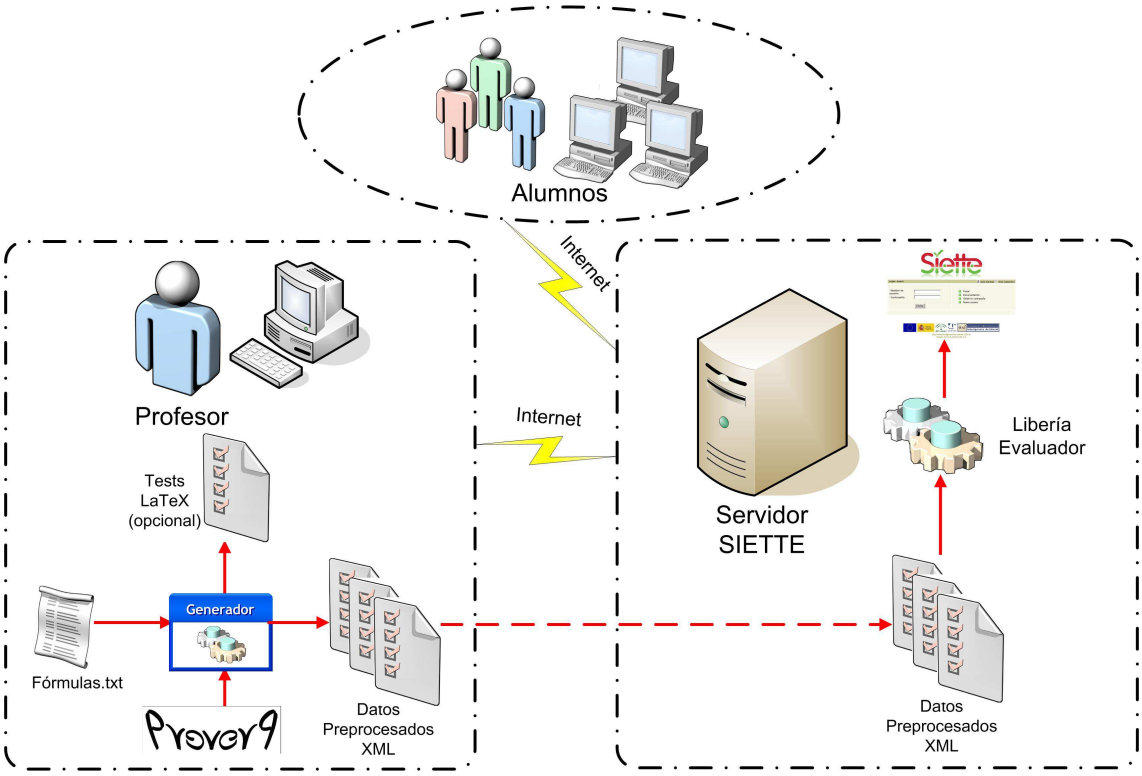


Figura 3.1: Estudio de Viabilidad. Componentes de despliegue.

4. Fundamentos

4.1. Lógica computacional

4.1.1. Conceptos preliminares

El objetivo principal de este proyecto es la generación automática de preguntas de lógica de predicados, más concretamente lógica proposicional y lógica de predicados de primer orden, por ello se describen los fundamentos teóricos y herramientas utilizadas para construir un conjunto de clases de Java que den soporte al resto de clases, con posibilidad de reutilizarse para otros proyectos.

Como se demostrará en la secciones siguientes cualquiera de las definiciones se basarán en evaluar la (in)satisfacibilidad de una o varias fórmulas, tanto proposicionales como de predicados de primer orden.

También se necesitará algún método capaz de devolver una interpretación en el caso de fórmulas satisfacibles o por el contrario una prueba en el caso de que sea insatisfacible.

Se ha confeccionado una estructura de capas (véase figura 4.1) con el fin de conseguir la construcción de los conceptos semánticos básicos (satisfacibilidad, validez, consecuencia y equivalencia). Esta estrategia tiene las siguientes ventajas:

- Trata la complejidad del problema utilizando el paradigma de diseño algorítmico *"Divide y vencerás"*.
- Es modular, de tal forma que podemos reemplazar cualquiera de las capas, siempre y cuando se mantengan la estructura de los métodos.

De esta forma, se podría muy fácilmente reemplazar el software utilizado en especial en las capas más bajas. Como ya se comentó en el estudio de viabilidad (véase capítulo 3), se van a realizar llamadas a dos programas externos Prover9 y Mace4 [Prover9Mace4, 2009], pero si se en el futuro se considerará más oportuno el uso de otros programas o librerías más adecuadas, se podría fácilmente reemplazar siempre y cuando se conservarán la estructura de los métodos, ocultando a la capas superiores la complejidad.

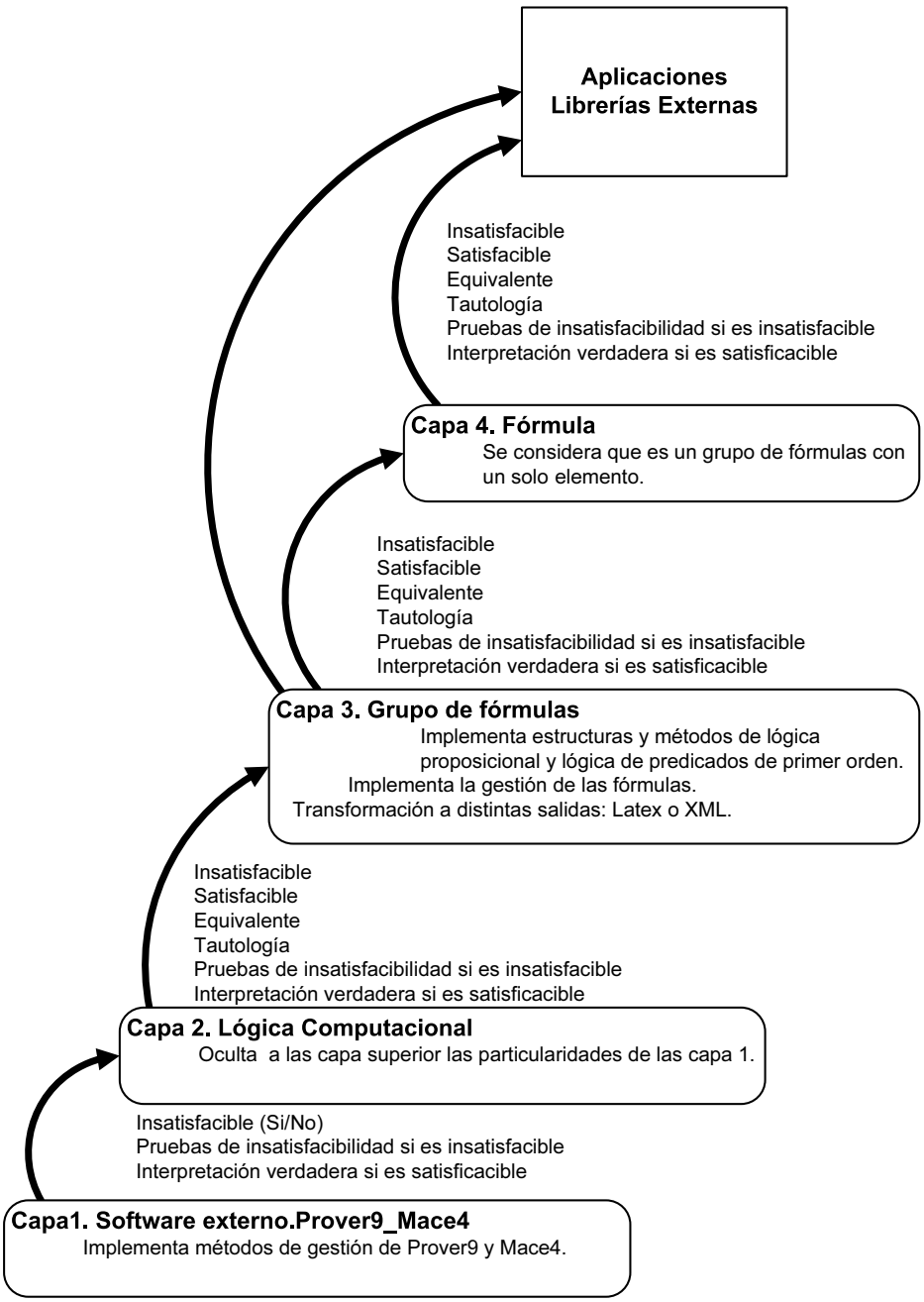


Figura 4.1: Conceptos Preliminares. Estructuras por capas de lógica computacional.

Es conveniente tener una visión global de la funcionalidad de cada capa que en secciones sucesivas se irán detallando:

1. Software externo. Prover9 | Mace4.
2. Lógica Computacional
3. Grupo de fórmulas.
4. Fórmula.

Capa 1. Software externo. Prover9/Mace4: Es la capa de más bajo nivel, se encarga de implementar los métodos necesarios para que a partir del software externo elegido, en nuestro caso Prover9 y Mace4 , se pueda suministrar a las capas superiores los métodos capaces de obtener a partir de un grupo de fórmulas si es insatisfacible, así como pruebas de insatisfacibilidad en el caso de fórmulas insatisfacibles y al menos una interpretación verdadera en el caso de fórmulas satisfacibles.

Capa 2. Lógica Computacional: Esta capa oculta a las capas superiores las particularidades de las capa 1, así mismo implementa nuevos métodos capaces de devolver a partir de un grupo de fórmulas si es insatisfacible, satisfacible, equivalente o tautología, además de pruebas de insatisfacibilidad en el caso de fórmulas insatisfacibles y al menos una interpretación verdadera en el caso de fórmulas satisfacibles.

Capa 3. Grupo de fórmulas: Esta capa es la utilizada el resto de clases o librerías externas y la capa 4 Fórmula. Utiliza los métodos proporcionados por la capa Lógica Computacional y además implementa todas aquellas estructuras y métodos con temas relacionados con lógica proposicional y lógica de predicados de primer orden. Además de los métodos ya mencionados anteriormente: insatisfacible, satisfacible, equivalente y tautología, obtener pruebas de insatisfacibilidad o interpretaciones satisfacibles, también implementa todos aquellos relacionado con la gestión de las fórmulas propiamente dicha, como pudieran ser agregar fórmulas a un grupo, o la transformación a las distintas salidas como pudiera ser LaTeX o XML.

Capa 4. Fórmula: Será utilizada por el resto de clases o librerías externas y en algunos casos, como por ejemplo la gestión de las salidas por la capa 3 Grupo de Fórmulas. En un intento de reutilización se ha considerado que una fórmula es un grupo de fórmulas con un sólo elemento, por ello utilizará los métodos proporcionados por la capa 3 Grupo de Fórmulas, aunque por motivos de optimización algunos métodos como por ejemplo el relacionado con conceptos de equivalencia son tratados, de forma excepcional, directamente con la capa 2 Lógica Computacional. También implementa todos aquellos relacionado con la gestión de las fórmulas individuales, como la transformación a las distintas salidas como pudiera ser LaTex o XML. En realidad, se puede considerar que las capas 3 Grupo de fórmulas y 4 Fórmula están al mismo nivel, colaborando entre sí.

4.1.2. Conceptos semánticos básicos utilizados

A continuación se describen los conceptos semánticos básicos utilizados de lógica proposicional y lógica de predicados de primer orden utilizados durante el desarrollo de la librería objeto de esta memoria, sin entrar, ni pretender ser un manual de estudio, por considerar la existencia de abundante bibliografía al respecto; la única pretensión es hacer referencia a aquellos aspectos que se han tenido en cuenta a lo largo del desarrollo y como se han aplicado de forma práctica.

Como documento base para la elaboración de las siguientes subsecciones se ha tomado como referencia J. L. Fernández Vindel [2003 - revisado en 2007], donde se podrá encontrar una selección de bibliografía comentada y ampliada.

Todos los conceptos que se exponen a continuación se pueden hacer extensibles tanto a lógica proposicional como a lógica de predicados de primer orden.

Merece la pena destacar que los **conceptos básicos de validez, consecuencia y equivalencia se pueden basar en el concepto de (in)satisfacibilidad**.

4.1.2.1. Satisfacibilidad

La **satisfacibilidad** es la potencialidad de ser satisfecho. En lógica, una interpretación satisface una o varias fórmulas cuando éstas se evalúan como verdaderas en esa interpretación.

Una interpretación ν satisface una fórmula φ si $\nu(\varphi) = 1$. Una interpretación ν satisface un conjunto de fórmulas $\Phi = \{\varphi_1, ..., \varphi_n\}$ si $\nu(\varphi_k) = 1$ para toda fórmula φ_k de Φ .

De esta forma, la **insatisfacibilidad** es la potencialidad de no ser satisfacible

Los métodos para deducir la satisfacibilidad pueden ser entre otros:

- Un método deductivo, en el caso de Lógica Proposicional también se puede utilizar tablas de verdad cuando el número de símbolos no sea muy elevado.
- Opcionalmente se puede utilizar un método de resolución llegando a soluciones no vacías,
- También se podría utilizar árboles semánticos llegando a la conclusión de que es satisfacible si este no se puede cerrar.

Los métodos para deducir la insatisfacibilidad pueden ser entre otros:

- Mediante tablas de verdad en lógica proposicional, siempre y cuando la cantidad de símbolos lo permita, comprobando que todas las valoraciones dan como resultado valores falsos (0).
- Cuando no disponemos del colchón de las tablas de verdad, (Lógica de Primer Orden o cardinalidad elevada en el caso de la Lógica Proposicional) se podrá llegar a una cláusula vacía (por resolución) ó a un árbol semántico cerrado.

4.1.2.2. Validez

Una fórmula válida es aquélla que es verdadera frente a cualquier interpretación. Las **tautologías** son fórmulas válidas. La satisfacibilidad divide en dos al conjunto de fórmulas: en insatisfacibles (contradicciones) y satisfacibles.

Dos observaciones cobran especial interés por utilizarse en el desarrollo de este proyecto:

- Si niega una fórmula insatisfacible, la fórmula resultante es una tautología.
- Si niega una tautología, la fórmula resultante es insatisfacible.

Para decidir la validez de una fórmula, el procedimiento semántico extensivo requiere recorrer toda la tabla de verdad. Los resultados negativos se pueden obtener más rápidamente: basta encontrar la primera interpretación que no satisface la fórmula, pero los resultados positivos requieren una comprobación completa. Se resalta que una fórmula φ es tautología si y sólo si $\neg\varphi$ es insatisfacible. Estos conceptos son extensibles a la lógica de predicados de primer orden. Un caso particular es el estudio de validez del conjunto de fórmulas $((\varphi_1 \wedge ... \wedge \varphi_n) \rightarrow \varphi)$, es equivalente a $(\neg(\varphi_1 \wedge ... \wedge \varphi_n) \vee \varphi)$. Por tanto, su negación es equivalente a la fórmula $(\varphi_1 \wedge ... \wedge \varphi_n \wedge \neg\varphi)$, si ésta es insatisfacible quedaría demostrado que el conjunto de fórmulas original es una tautología.

4.1.2.3. Consecuencia

Una fórmula ψ es consecuencia de un conjunto $\Phi = \{\varphi_1, ..., \varphi_n\}$ de fórmulas si toda interpretación que satisface a Φ también satisface a ψ. La notación para representar que ψ es consecuencia lógica de $\Phi = \{\varphi_1, ..., \varphi_n\}$ se suele emplear la notación $\Phi \models \psi$, ó $\{\varphi_1, ..., \varphi_n\} \models \psi$. De esta forma: $\varphi_1, ..., \varphi_n \models \psi$ si y sólo si $\varphi_1 \wedge ... \wedge \varphi_n \rightarrow \psi$ es tautología. Por tanto, existe una interdependencia formal entre consecuencia e insatisfacibilidad:

- $\varphi_1, ..., \varphi_n \models \psi$ y entonces el conjunto $\{\varphi_1, ..., \varphi_n, \neg\psi\}$ es insatisfacible.
- $\{\varphi_1, ..., \varphi_n, \neg\psi\}$ es insatisfacible entonces $\varphi_1, ..., \varphi_n \models \psi$.

Para verificar una relación de consecuencia se podría realizar un tabla de verdad, siempre y cuando fueran fórmulas proposicionales y la cardinalidad lo permitiera, procediendo a evaluar que en todas las líneas en que esas fórmulas coinciden en ser verdaderas, la última también lo es (y quizá en alguna más). Otro procedimiento extensible a la Lógica de Primer Orden sería demostrar que $\varphi_1 \wedge ... \wedge \neg\psi$ es insatisfacible, bien mediante un árbol semántico cerrado ó un método de resolución a partir del cuál se pudiera obtener una cláusula vacía.

4.1.2.4. Equivalencia

Aunque este concepto es extensible a la lógica proposicional y la lógica de predicados de primer orden, conviene distinguirlas en su definición.

Dos fórmulas proposicionales, ψ y Φ, son equivalentes si $\nu(\psi) = \nu(\Phi)$ para toda interpretación ν. Sobre la tabla de verdad, dos fórmulas equivalentes tienen exactamente los mismos valores de verdad sobre cada línea. Muy coloquialmente, son dos formas sintácticamente distintas de expresar lo mismo (puesto que semánticamente son indistinguibles). Existen distintos métodos para demostrar que dos fórmulas son equivalentes. En el caso de lógica proposicional bastaría con representar las tablas de verdad correspondientes a cada fórmula y comprobar que son idénticas.

Dos fórmulas ψ y Φ de primer orden serán equivalentes si $\psi \models \Phi$ y $\Phi \models \psi$. Existen distintos métodos para demostrar que dos fórmulas son equivalentes. Cuando la cardinalidad no lo permita como es el caso de la lógica de predicados de primer orden, donde no se tiene el colchón de las tablas de verdad, un método eficaz, también aplicable a la lógica proposicional, es demostrar que es insatisfacible la fórmula $\neg(\psi \leftrightarrow \Phi)$. Otro método, aplicado en ambos tipos de lógicas, sería partiendo de cualquiera de las fórmulas equivalentes y mediante reemplazos de equivalencias básicas llegar a la otra fórmula que se supone equivalente.

4.1.2.5. Interpretaciones

La interpretación en lógica de proposiciones. Para decidir si una fórmula proposicional es verdadera se requiere interpretarla. La interpretación se produce sobre un objeto matemático: sobre una asignación, sobre una función del conjunto de letras proposicionales en {0,1}. Bastará una asignación para que, de cualquier fórmula, se decida su valor de verdad. Es decir, basta una asignación para que se compruebe si esa interpretación de la fórmula la satisface. La definición recursiva de satisfacción facilita esta decisión, por muy compleja que sea la fórmula.

Es más, sobre una misma asignación se pueden interpretar todas las fórmulas de un conjunto de fórmulas dado. Se puede apreciar así su distinto 'comportamiento' (su valor de verdad) frente a un mismo 'estado de cosas' (una misma interpretación). Bastaba escribir la tabla de verdad conjunta, para todas las letras proposicionales que aparecían en esas fórmulas.

La interpretación en lógica de predicados de primer orden. Los lenguajes de primer orden tienen un alfabeto más expresivo: constantes, funciones, relaciones,. . . Para interpretar cualquiera de sus fórmulas es preciso un objeto matemático más complejo. De hecho, tanto más complejo cuántos más símbolos propios tenga el lenguaje. Para no perdernos, cuando la notación sea más farragosa, conviene ofrecer una descripción coloquial de este 'objeto matemático':

1. Se escoge primero un conjunto U no vacío, cualquiera.

2. Por cada predicado monádico, como $P(x)$, debe escoger un subconjunto de U. Por cada predicado diádico, como $R(x, y)$ debe escoger una relación binaria en U: un conjunto de pares de elementos de U. En general, por cada predicado n-ádico, un conjunto de n-tuplas de elementos de U.

3. Por cada símbolo constante en la fórmula debe escoger un elemento de U.

4. Por cada símbolo funcional, como $g(x, z)$, en la fórmula debe escoger una función sobre U con el mismo número de argumentos: 2, en el caso de g.

A una construcción como ésta se la denominará estructura. Por ejemplo, si una fórmula no contiene constantes, ni funciones y todos sus predicados son monádicos, una estructura adecuada es simplemente un conjunto y varios de sus subconjuntos. Cualquier sentencia se puede interpretar sobre una estructura adecuada a esa fórmula. Si existen apariciones de variables libres, la interpretación va a depender de 'quiénes se suponen que son esas variables sobre la estructura'. En este caso es preciso establecer una correspondencia entre variables y elementos de U, que denominaremos asignación.

Dada una fórmula Φ, una estructura adecuada a ella será una construcción matemática sobre la que se pueda interpretar Φ: un conjunto U y la 'elección de un representante' para cada símbolo propio empleado en Φ (constantes, funcionales, relacionales). En realidad, se ofrecerá una interpretación para cada símbolo propio del lenguaje, así se podrán interpretar varias fórmulas sobre una misma estructura aún cuando alguna utilice un símbolo propio que otra no usaba.

Una estructura adecuada al lenguaje $L(RFC)$ es un par $\langle U, I\rangle$ tal que:

1. U es un conjunto no vacío, denominado dominio o universo

2. I es una función sobre el conjunto de símbolos propios de $S = R \cup F \cup C$ que hace corresponder:

   a) a cada símbolo relacional n-ario $R \in S$, una relación n-aria sobre U

   b) a cada símbolo funcional n-ario $f \in S$, una función n-aria sobre U

   c) a cada constante $c \in S$, un elemento de U

Una interpretación satisface a una fórmula $\psi_1 \wedge ... \wedge \psi_n$, si satisface a cada una de las fórmulas por separado. Una interpretación hace insatisfacible a una fórmula $\psi_1 \wedge ... \wedge \psi_n$, si existe al menos una fórmula insatisfacible. Para determinar la insatisfacibilidad de una fórmula para una interpretación dada, se puede utilizar una tabla semántica y si ésta es cerrada implica que es insatisfacible, otro

método sería mediante resolución llegar a una cadena vacía. Para determinar la satisfacibilidad de ψ_i para una interpretación dada se puede deducir si $\neg\psi_i$ es insatisfacible. Un posible algoritmo para obtener interpretaciones que hacen satisfacibles ψ_i, sería previamente descartar que no es insatisfacible, si es así, analizar los valores del dominio que hacen verdad a ψ_i.

4.1.3. Prover9 y Mace4

Como ya se ha analizado el estudio de viabilidad (véase capítulo 3), se ha elegido Prover 9 y Mace 4 como aplicación de soporte para la construcción de los conceptos de lógica proposicional y de predicados de primer orden, de forma que se pudiera reutilizar software ya existente, de otro modo obligaría a su implementación que como se puede suponer escapa al desarrollo del presente proyecto.

Prover9 se utilizará para deducir si un conjunto de fórmulas es insatisfacible y **Mace4** para obtener interpretaciones verdaderas en el caso de que un conjunto de fórmulas fuera satisfacible.

Tanto Prover9 como Mace4 disponen de una interfaz gráfica además de sendas aplicaciones basadas en comandos (prover9.exe y mace4.exe). Durante el desarrollo del proyecto se han utilizado el formato comando por ser más apropiado a la hora de invocarse desde una clase java.

La sintaxis de invocación de ambos aplicaciones es la siguiente "*comando -t n -f fichero_entrada > fichero_salida*", donde comando puede ser *prover9* ó *mace4*, *-t n*, acota a n segundos el tiempo de búsqueda de una respuesta, -f indica que la entrada se obtendrá del fichero f*ichero_entrada* y la salida será almacenada en el fichero *fichero_salida*.

Si estudiamos el siguiente ejemplo *prover9 -t 10 -f input.in > output.out*, donde *input.in* contiene la siguiente información:

```
formulas(assumptions).
        humano(x) -> mortal(x).
        humano(joaquin).
end_of_list.
formulas(goals).
      mortal(joaquin).
end_of_list.
```

Si ejecutamos prove9, efectivamente es capaz de probar el teorema expuesto, en el que se dan unas premisas:

```
humano(x) -> mortal(x).
humano(joaquin).
```

y una conclusión:

```
mortal(joaquin).
```

Pero si el mismo fichero de entrada se utiliza para ejecutar mace4 se comprobaría que no es capaz de encontrar ningún contraejemplo.

Para mayor información sobre la sintaxis de los ficheros de entrada se recomienda lectura del manual de Prover9 y Mace4 [Prover9Sintaxis, 2009], también puede consultar el resumen incluido en el Apéndice B.1.1.2.

4.1.3.1. Estudio de comportamiento

Al estudiar el comportamiento del uso de una fórmula insatisfacible, por ejemplo $q \wedge -q$, como premisa y sin fórmula conclusión, traducido a lenguaje Prover9 y ejecutando el comando *prover9 -t 10 -f input.in > output.out*:

```
formulas(assumptions).
        q & -q.
end_of_list.
formulas(goals).
end_of_list.
```

Se obtiene como salida una prueba de que es capaz de llegar por deducción a una cadena vacía, en 0,06 segundos.

La misma entrada pero en este caso utilizando Mace4, *mace4 -t 10 -f input.in > output.out*, no es capaz de encontrar ningún, modelo o contraejemplo como era de esperar, llegando a esta conclusión una vez agotado todo el tiempo.

Ahora bien, si se utiliza una fórmula satisfacible, por ejemplo $q \vee -q$, con premisa y sin fórmula de conclusión, traducido al lenguaje Prover9 y ejecutando *prover9 -t 10 -f input.in > output.out*:

```
formulas(assumptions).
        q | -q.
end_of_list.
formulas(goals).
end_of_list.
```

En la salida indica que no puede obtener ninguna prueba en 0,03 segundos.

La misma entrada usada con Mace4, *mace4 -t 10 -f input.in > output.out*, encuentra una interpretación en la que la fórmula es satisfacible (q=0), invirtiendo en ello 0,01 segundos.

Tomando como referencia el comportamiento expuesto y después de realizar numerosas pruebas se llega a las siguientes conclusiones:

- Si al ejecutar Prover9 con un grupo de fórmulas (premisas) que en su conjunto son insatisfacibles, sin fórmula conclusión, devolverá siempre al menos una prueba y además no contendrá en la salida la cadena “Exiting with failure.”. Todo ello en un tiempo razonable (menor de 1 segundo en la mayoría de los casos).

- Si la ejecutar Prover9 con un grupo de fórmulas (premisas) que en su conjunto son satisfacibles, sin fórmula conclusión, no devolverá ninguna prueba y además contendrá la cadena "Exiting with failure.". Todo ello en un tiempo razonable (menor de 1 segundo en la mayoría de los casos).
- Si al ejecutar Mace4 con un grupo de fórmulas (premisas) que en su conjunto son insatisfacibles, sin fórmula conclusión, no devolverá un modelo que demuestre la satisfacibilidad y unos caso agotará el tiempo asignado y en otros la salida contendrá la cadena "Exiting with failure.". Los tiempos para obtener la salida no suelen ser razonables (varios segundos).
- Si al ejecutar Mace4 con un grupo de fórmulas (premisas) que en su conjunto son satisfacibles, sin fórmula conclusión, un modelo que demuestre la satisfacibilidad . Los tiempos para obtener la salida no suelen ser razonables (menores o ligeramente superiores a 1 segundo), pero mayores los manejados por Prover9.

Aplicando las conclusiones anteriores se puede establecer un **algoritmo para deducir si un grupo de fórmulas es (in)satisfacible** en tiempos razonables.

1. Se utilizará Prover9 para deducir si un grupo de fórmulas es insatisfacible.
2. Se utilizará Prover 9 para deducir si un grupo de fórmulas es satisfacible, debido a que si no es insatisfacible, tal y como se indica en el paso 1, entonces quiere decir que es satisfacible.
3. Se utilizará Prover9 para obtener una prueba en el caso de que el grupo de fórmulas sea insatisfacible, deducido este hecho a partir del paso 1.
4. Se utilizará Mace4 para obtener al menos una interpretación válida en el caso de que el grupo de fórmulas sea satisfacible, deducido este hecho a partir del paso 2.

4.2. Examen tipo test

4.2.1. Definición de test

Si se estudia como percibe en la realidad, tanto el profesor como el alumno, un examen tipo test, se puede observar que:

- Un test tiene un grupo de preguntas.
- Las preguntas se pueden agrupar en tipos o categorías.
- El test organiza tanto las preguntas, como los datos comunes a todas ellas.

4.2.2. Datos comunes

Las preguntas de un examen tipo test podría tener datos comunes, lo que evitaría la redundancia a la hora de formular las preguntas, con un ejemplo se clarificará esta idea: Imaginemos que se está evaluando al alumno respecto a la comprensión escrita de un texto en inglés, se intuye que no

sería lógico repetir el texto completo cada vez que se realizará una pregunta, sería más conveniente hacer referencia al texto y expresarlo éste de forma común a todas las preguntas. Pero también podría suceder que existiesen preguntas individuales (sin datos comunes) .

Por ello, se debe gestionar si una pregunta se basa sobre un dato común o es individual. Más adelante se estudiará como se ha resuelto esta problemática en el caso tratado en este proyecto (exámenes de lógica proposicional y lógica de predicados de primer orden 4.3.

4.2.3. Preguntas tipo test

Las preguntas deberán cumplir con los siguientes requisitos:

- Se confeccionarán en base datos comunes del examen o de forma individual.
- Incorporarán los conceptos teóricos en los que se basan.
- Almacenarán la información necesaria para determinar si una respuesta es verdadera o falsa.
- Generarán y almacenarán la información necesaria para dar información sobre el razonamiento de las respuestas tanto verdaderas como falsas.

De esta forma las estructuras de datos necesarias para gestionar una pregunta serán:

- Título de la pregunta.
- Datos comunes a la pregunta.
- Lista de respuestas.
- Conceptos teóricos del tipo de pregunta.

A su vez cada respuesta de la lista de respuestas definirá:

- Enunciado de una respuesta.
- Si es correcta o incorrecta.
- Solución razonada, tanto si es verdadera como si es falsa.

4.2.4. Tipos de salidas

Los formatos de acceso de los exámenes deben ser lo suficientemente extendidos para que sin apenas requisitos técnicos los alumnos puedan visualizarlos y realizarlos, por ello se han considerado los siguientes tipos de salida:

- LaTeX.
- PDF,s.
- MathJax.
- JSP.
- SIETTE.

4.2.4.1. LaTeX

Gracias al uso de LaTeX [LaTeX, 2011] se subsana la problemática de la visualización de símbolos y fórmulas matemáticas, dejando de ser una restricción, con independencia de su complejidad, a la hora de realizar preguntas.

La gestión de documentos en formato LaTex, conlleva el uso de una distribución, para lo cuál se ha elegido **MiKTeX** [MiKTeX, 2012], así como un editor **TeXnicCenter** [TeXnicCenter, 2012], facilitando las herramientas para la visualización (no será necesaria la composición del texto, ya que la librería nos proporcionará un examen o preguntas individuales en formato LaTeX), así como la compilación y la exportación a otros formatos como pueda ser PDF.

En realidad, el alumno no visualizará las preguntas en este formato, sino que se utilizará como base para otros. De esta forma los test en formato LaTeX serán exportados a otros formatos como PDF, aunque también se utilizará como formato base de MathJax, gracias a la cual se podrá utilizar en páginas JSP ó en entornos de evaluación de conocimientos, como por ejemplo SIETTE.

Destacar que durante el desarrollo del proyecto se ha utilizado con éxito la distribución de LaTeX MiKTeX [MiKTeX, 2012].

4.2.4.2. PDF

A partir de la librería implementada se podrá obtener una salida en LaTeX, formato que se podrá exportar a PDF, para este fin, durante el desarrollo del proyecto, se ha utilizado con éxito el editor **TeXnicCenter** [TeXnicCenter, 2012] .

4.2.4.3. MathJax

MathJax [MathJax, 2012] es una biblioteca javascript que permite visualizar mediante navegadores web contenidos matemáticos generados en formato LaTeX, por tanto, servirá como soporte para poder implementar contenidos web capaces de gestionar la notación de las fórmulas de lógica, de esta forma los navegadores no necesitarán la instalación de componentes adicionales.

La instalación de está biblioteca consiste básicamente en descargarla de su página web oficial y posterior descompresión el la carpeta *webapps* de un servidor Apache-Tomcat. Además en la cabecera (<HEAD>) de las páginas webs implementadas para la visualización de los contenidos LaTeX deberá incluirse una cláusula <SCRIPT> que haga referencia a la biblioteca, a continuación se muestra un ejemplo:

```
<SCRIPT SRC="../MathJax.js">
        MathJax.Hub.Config({extensions:["tex2jax.js"],
        jax:["input/TeX","output/HTML-CSS"],tex2jax:{inlineMath:
        [["$","$"],["\\(","\\)"]]}});
</SCRIPT>
```

Como primer paso en el estudio de ejemplos se puede acceder a los incluidos en la distribución de la biblioteca, aunque también sería conveniente consultar el código fuente de la página web `http://www.mathjax.org/mathjax/test/`.

4.2.4.4. JSP

El uso de este formato es nos permitirá:

- La correcta visualización de las preguntas en formato MathJax.
- Demostrar el uso de la librería en contenidos webs implementados por terceros.
- Como entorno de preproducción y pruebas antes de realizar despliegues en servidores SIETTE.

Se considera está última opción como la más importante de este formato, posibilitando el desarrollo de la librería de forma local, sin necesidad de tener que estar constantemente instalando versiones de la librería en servidores SIETTE para comprobar su funcionamiento, llevando a cabo los despliegues definitivos en dichos servidores con la seguridad de su correcto funcionamiento, tanto tecnológico, como funcional.

4.2.4.5. SIETTE

El objetivo principal de este proyecto es poder realizar preguntas tipo test en SIETTE . Una de las características de este sistema de evaluación es la creación de ítems generativos basados en JSP, compatibles con MathJax; por tanto una vez conseguido que la librería implementada funcione en páginas JSP, es relativamente sencillo exportarla a servidores SIETTE.

4.3. Preguntas tipo test de lógica

El objetivo de la librería implementada es la generación automática de preguntas de tipo test tanto de lógica proposicional y lógica de predicados de primer orden, este hecho tiene las siguientes implicaciones:

- Debido a la complejidad de los cálculos necesarios deberán dotarse de estructuras ayudantes capaces de auxiliar en la generación de las preguntas.
- Gestionarán fórmulas proposicionales, fórmulas de predicados de primer orden, así como sus interpretaciones.
- En el caso de ser preguntas autónomas, es decir sin datos comunes, todos los datos deberán expresarse en la pregunta, sin embargo si el examen tuviera datos comunes se ubicarán en una sección a las que se harán referencia desde la pregunta, para mayor claridad vea dos ejemplos, en la que se realiza la misma pregunta con y sin datos comunes:

Pregunta con datos comunes (las fórmulas proposicionales $\{X_1, X_2, X_3\}$ será proporcionadas en una sección común al resto de preguntas
El conjunto satisfacible es:

(a) $\{X_1, X_2, \neg X_3\}$

(b) $\{X_1, \neg X_2, X_3\}$

(c) $\{X_1, X_2, X_3\}$

Pregunta con sin datos comunes (las fórmulas proposicionales se muestran en la pregunta)
Dadas las siguientes fórmulas proposicionales, el conjunto satisfacible es:
$X_1 : \neg (t \vee (\neg q \rightarrow p))$
$X_2 : p \vee q \rightarrow r \wedge s$
$X_3 : \neg (\neg t \rightarrow p)$

(a) $\{X_1, X_2, \neg X_3\}$

(b) $\{X_1, \neg X_2, X_3\}$

(c) $\{X_1, X_2, X_3\}$

- Será más costoso la generación de preguntas autónomas, ya que conllevará la generación de datos cada vez que se invoque una pregunta, sin embargo, en el caso de tener datos comunes, estos se calcularán una sola vez por test y podrán ser reutilizados por todas las preguntas del mismo.

- Respecto al número de fórmulas proposicionales o de predicados de primer orden, diferirá en el caso de tener preguntas con referencias a datos comunes o preguntas autónomas. En el caso de preguntas con sección de datos comunes se ha determinado cuatro fórmulas de cada tipo, mientras que si no existe tal sección se estima ideal un número de tres. La razón principal es que si se utilizarán cuatro fórmulas para las preguntas sin datos comunes, tendría que implementarse la lógica necesaria para eliminar todas aquellas fórmulas que no se utilizaran en la pregunta, así como renumeración de los subíndices de los nombres de las fórmulas, para mostrar un aspecto coherente.

4.4. Patrones utilizados

Se ha utilizado el Paradigma Orientado a Objetos durante las distintas fases del proyecto, por ello se han aplicado algunos patrones de diseño [Gamma et al., 2005] basados en la experiencia y que se ha demostrado su correcto funcionamiento [Gracia, 2005].

- **Singleton**. Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.
- **Facade**. Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar.
- **Abstract Factory**. Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.

5. Análisis

5.1. Objetivos del sistema

De la sección Objetivos (véase 1.3) se pueden catalogar los siguientes objetivos de sistema:

OBJ-1	**Generación de automática de preguntas**
Descripción	El sistema deberá ser capaz de generar de forma automática preguntas de tipo test de lógica computacional. Las preguntas incluirán enunciado, respuestas, soluciones y conceptos generales.

OBJ-2	**Presentación de preguntas**
Descripción	El sistema mostrará las preguntas y posibles respuestas.

OBJ-3	**Realización de preguntas**
Descripción	El sistema permitirá seleccionar de cada pregunta la respuesta que el alumno considere más adecuada.

OBJ-4	**Presentación de soluciones y conceptos generales**
Descripción	El sistema mostrará las soluciones y los conceptos generales como resultado de la selección de las respuestas.

5.2. Requisitos funcionales

5.2.1. Diagrama de casos de usos

El sistema se subdividirá en dos subsistemas (véase 3): un sistema generador de test (Generador Preguntas) y otro para la realización por parte de los alumnos (Evaluador Preguntas).

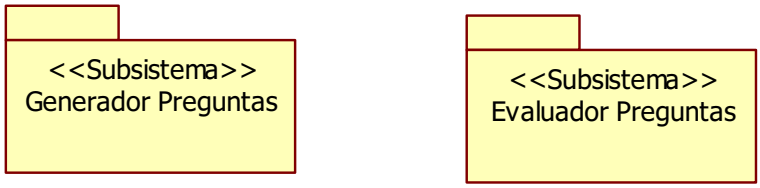


Figura 5.1: Análisis. Diagrama de subsistemas.

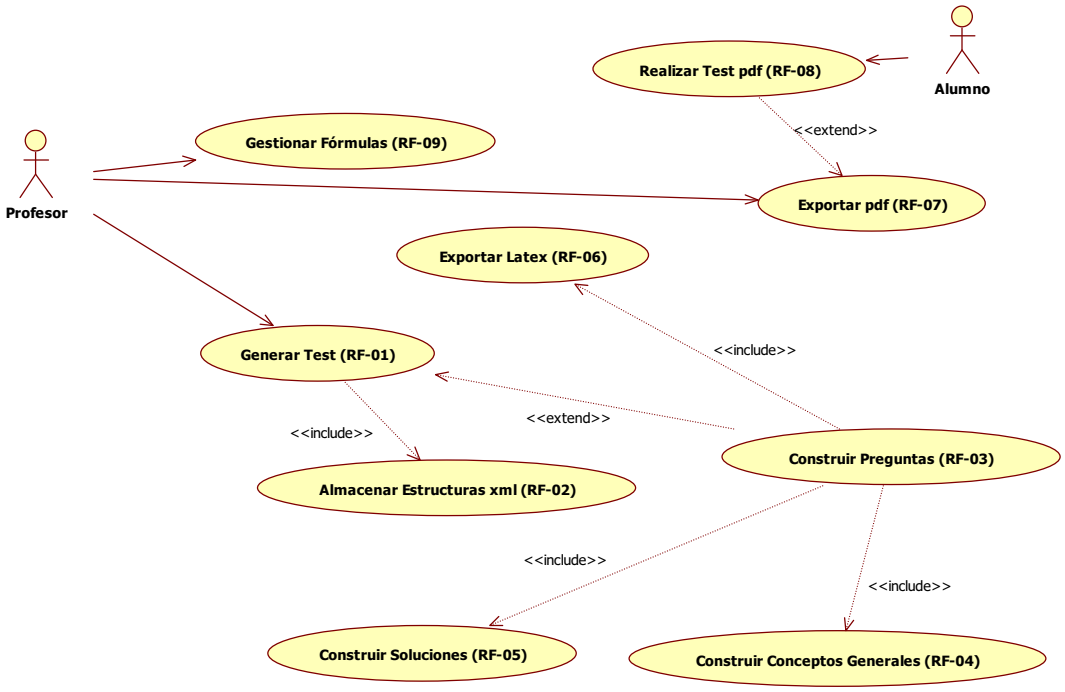


Figura 5.2: Análisis. Diagrama de casos de uso generador preguntas.

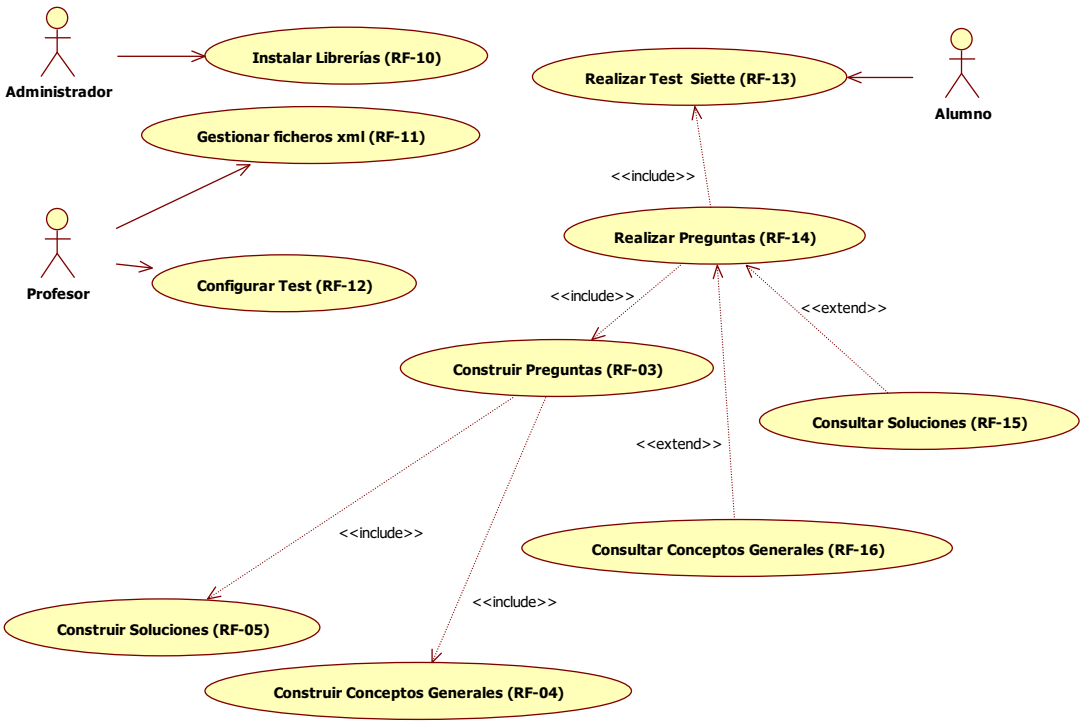


Figura 5.3: Análisis. Diagrama de casos de uso evaluador preguntas.

5.2.2. Definición de actores

A continuación se definen los actores que interaccionarán con los subsistemas.

ACT-1	**Profesor**
Descripción	Profesor de alguna de las asignaturas " *Lógica y Estructuras Discretas*" y "*Lógica Computacional*", incluidas en las titulaciones de la UNED "*Grado en Ingeniería Informática* ", "*Grado en Ingeniería en Tecnologías de la Información*" e "*Ingeniero en Informática*".

ACT-2	**Alumno**
Descripción	Alumno de alguna de las asignaturas " *Lógica y Estructuras Discretas*" y " *Lógica Computacional*", incluidas en las titulaciones de la UNED " *Grado en Ingeniería Informática* ", " *Grado en Ingeniería en Tecnologías de la Información*" e " *Ingeniero en Informática*".

ACT-2	**Administrador**
Descripción	Administrador del entorno de evaluación conocimientos SIETTE .

5.2.3. Casos de uso del subsistema generador

A continuación se pasan a definir los distintos casos de usos del subsistema Generador de Preguntas, dicho subsistema se traducirá en aplicación de línea de comandos con distintos parámetros que modificarán su comportamiento y ejecutará el profesor de la asignatura en su ordenador personal.

Dicha aplicación tendrá doble finalidad:

- La **finalidad más importante**: **Generación de los ficheros .XML con datos preprocesados**, que posteriormente serán importados a SIETE.
- Poder generar exámenes tipo test automáticamente en formato LaTeX, fácilmente exportables a otros como PDF. Esta finalidad no es requisito del proyecto, pero debido al amplio estudio de viabilidad (véase capítulo 3) , donde todas las pruebas se realizaron en un ordenador personal, ha proporcionado la posibilidad de dotar de esta funcionalidad al proyecto, que puede ser de gran utilidad para probar el comportamiento cuando el profesor agregue nuevas fórmulas ó la propia generación de preguntas sin necesidad de entorno de evaluación de conocimientos.

RF-01. Generar Test.

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un profesor solicite la generación de un test.

Precondición. El profesor habrá actualizado las fórmulas tal y como se describe en el requisito funcional RF-09 Insertar Fórmulas.

Secuencia normal.

1. El profesor ejecuta el programa con los parámetros adecuados para la generación de un test.

2. El sistema busca los conjuntos de fórmulas proposicionales y de lógica de predicados de primer orden con las condiciones descritas a continuación que asegurarán la generación de preguntas basadas en los modelos de preguntas (véase 6.3.5).

- Se obtendrá dos conjuntos de fórmulas uno de lógica proposicional y otro de lógica de predicados de primer orden.
- Cada uno de los conjuntos tendrán tres fórmulas como mínimo, es decir, tres de lógica proposicional y tres de lógica de predicado de primer orden.
- Cada uno de los conjuntos tendrán una fórmula que tendrán al menos dos fórmulas equivalentes.
- Cada uno de los conjuntos tendrán al menos tres grupos de tres fórmulas insatisfacibles.
- Cada uno de los conjuntos tendrán al menos tres grupos de tres fórmulas satisfacibles.

3. El sistema generará las estructuras necesarias utilizadas para la generación de las preguntas según se expone en el RF-02 Almacenar XML. Dichas estructura se obtendrán a partir de los dos conjuntos de fórmulas deducidas en el punto 2, de esta forma por cada conjunto (lógica proposicional y lógica de predicados de primer orden) se almacenarán la siguiente información:

- Conjunto de tres fórmulas (determinadas en el punto 2).
- Todos los grupos posibles de tres fórmulas satisfacibles (incluyendo su forma negada), además de una interpretación satisfacible para cada uno.
- Todos los grupos de tres fórmulas insatisfacibles (incluyendo su forma negada), además de una prueba de insatisfacibilidad para cada uno.
- Por cada par de fórmulas equivalentes halladas (ψ y Φ) se almacenarán dos pruebas, de forma que se pueda demostrar dicha equivalencia ($\psi \models \Phi$ y $\Phi \models \psi$).
- Por cada par de fórmulas no equivalentes halladas (ψ y Φ) se almacenarán dos interpretaciones, de forma que se pueda demostrar la no equivalencia ($\psi \models \Phi$ y $\Phi \models \psi$).
- Todas las fórmulas equivalentes entre sí.
- Todas las fórmulas no equivalentes entre si.

- Además en el caso de las fórmulas de predicado de primer orden se almacenarán todas las interpretaciones posibles del conjunto de fórmulas de este tipo, incluyendo su forma negada.

4. Las estructuras se exportan a un fichero .XML que será posteriormente añadido por parte del profesor al entorno de evaluación (SIETTE) según se expone en el RF-11 Gestionar Ficheros XML.

Secuencia alternativa.

3. Dependiendo de los parámetros de ejecución el sistema generará un test completo, incluyendo preguntas, respuestas, soluciones y conceptos generales en formato LaTeX, sin necesidad de guardar las estructuras en ficheros XML.

Rendimiento. No existen restricciones de tiempo, dado que el profesor solicitará la ejecución de la generación del test fuera de línea respecto a la interactividad por parte del alumno.

RF-02. Almacenar Estructuras XML.

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se almacenan las estructuras necesarias para poder realizar posteriormente un test de forma desconectada, es decir la no existencia de librerías o programas gestores de lógica como pudiera ser Prover9/Mace4. Las estructuras que se almacenarán serán las ya indicadas en el RF-01. Generar Test.

Precondición. El profesor habrá ejecutado el programa generador con los parámetros adecuados, indicando la generación de estructuras XML.

Secuencia normal.

1. El sistema exportará las estructuras de datos preprocesados a ficheros .XML que serán utilizados para la generación de las preguntas en la plataforma de evaluación (SIETTE).

Secuencia alternativa.

No contempladas.

Rendimiento. Las estructuras de datos de cada test se almacenarán en un fichero independiente, de esta forma los ficheros no serán de un tamaño excesivo y el parser realizado en la apertura del fichero XML no retrase la generación de las preguntas. Por tanto, deberá haber tantos ficheros XML como exámenes tipo test se hayan generado.

RF-03. Construir Preguntas.

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el sistema genera una pregunta.

Precondición. Ya existirán las estructuras de datos generadas en el RF01 Generar Test.

Secuencia normal.

1. Se importan desde un fichero XML las estructuras de un test seleccionado de forma aleatoria.

2. Dependiendo del tipo de pregunta se generan los conceptos generales según RF04 - Construir Conceptos Generales.

3. Dependiendo del tipo de pregunta se generarán las respuestas.

4. Dependiendo del tipo de respuesta y si son correctas o incorrectas se generan las soluciones razonadas según RF05 - Construir Soluciones.

Secuencia alternativa.

1. Este paso no será necesario en el caso de que el profesor hubiera decidido exportar directamente a formato LaTeX.

Rendimiento.

Independientemente del tipo de preguntas no superará segundo.

RF-04. Construir Conceptos Generales.

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el sistema genere conceptos generales relacionados con la pregunta en curso.

Precondición. Se ha iniciado la construcción de una pregunta de un tipo determinado.

Secuencia normal.

1. Dependiendo del tipo de pregunta se le asociará un texto estático predefinido (puede incluir notación de lógica).

Secuencia alternativa.

No contempladas.

Rendimiento.

No contemplado.

RF-05. Construir Soluciones.

Objetivos asociados.

OBJ2-Generación de automática de preguntas.

Descripción. El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el sistema genere las soluciones asociadas a cada respuesta de la pregunta en curso.

Precondición. Se ha iniciado la construcción de una pregunta de un tipo determinado.

Secuencia normal.

1. Como demostración, dependiendo del tipo de respuesta y si es correcta o incorrecta se le asociará un texto dinámico.

Secuencia alternativa.

No contempladas.

Rendimiento.

No contemplado.

RF-06. Exportar LaTeX.

Objetivos asociados.

1. OBJ2-Presentación de preguntas.

2. OBJ4-Presentación de soluciones y conceptos generales.

Descripción. El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el sistema genere un test completo en un fichero con formato LaTeX.

Precondición. Se ha generado todas las preguntas (enunciados, respuestas, soluciones y conceptos generales).

Secuencia normal.

1. Se abre un fichero con extensión .tex.

2. Se exportan a formato LaTeX, todas la preguntas (enunciados y repuestas tipo test), escribiéndose a continuación en el fichero abierto en paso 1.

3. Se exportan a formato LaTeX los datos comunes del examen (fórmulas proposicionales, fórmulas de predicados de primer orden e interpretaciones de las fórmulas de primer orden), escribiéndose a continuación en el fichero abierto en paso 1.

4. Por cada pregunta del paso 2, se exportan a formato LaTeX conceptos generales y soluciones, escribiéndose a continuación en el fichero abierto en paso 1.

5. Se cierra el fichero abierto en el paso1.

Secuencia alternativa.

No contempladas.

Rendimiento.

No contemplado.

RF-07. Exportar PDF.

Objetivos asociados.

1. OBJ2-Presentación de preguntas.

2. OBJ4-Presentación de soluciones y conceptos generales.

Descripción. El profesor podrá de forma opcional exportar el fichero LaTeX con el test completo (enunciado, respuestas tipo test, datos comunes tales como fórmulas e interpretaciones, conceptos generales y soluciones) a formato PDF.

Precondición. El profesor podrá habrá generado el fichero .tex con el examen completo tal y como se muestra en el caso de uso RF-06. Exportar LaTeX.

Secuencia normal.

1. El profesor ejecuta una aplicación capaz de exportar fichero en formato LaTeX a formato PDF.

2. El profesor abre un fichero con extensión .tex. que contiene el test completo en formato LaTeX.

3. El profesor exporta el fichero abierto en el paso 1 a formato pdf.

4. El profesor guarda el fichero PDF.

5. El fichero cierra la aplicación abierta en el paso 1.

Secuencia alternativa.

1. La aplicación utilizada es indiferente siempre y cuando soporte conversión de LaTeX a PDF, un ejemplo puede ser MiKTeX [MiKTeX, 2012]

Rendimiento.

No contemplado.

RF-08. Realizar Test PDF

Objetivos asociados.

1. OBJ3-Realización de Preguntas.

2. OBJ4-Presentación de soluciones y conceptos generales.

Descripción. El alumno podrá realizar el test completo en formato PDF (enunciado, respuestas tipo test, datos comunes tales como fórmulas e interpretaciones, conceptos generales y soluciones).

Precondición. El profesor podrá habrá generado el fichero PDF con el examen completo tal y como se muestra en el caso de uso RF-07. Exportar pdf.

Secuencia normal.

1. El alumno abre el fichero PDF que el profesor puso a su disposición.

2. El alumno realiza el test.

Secuencia alternativa.

No contemplada.

Rendimiento.

No contemplado.

RF-09. Gestionar Fórmulas

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El profesor podrá actualizar las fórmulas proposicionales y de lógica de primer orden a partir de las cuales se generarán los test de forma automática.

Precondición.

No contemplada.

Secuencia normal.

1. El profesor abre el fichero en el que se almacenan las fórmulas.

2. El profesor podrá agregar nuevas fórmulas en formato Prover9/Mace4 [Prover9Mace4, 2009].

3. El profesor podrá eliminar las fórmulas que crea necesario.

Secuencia alternativa.

No contemplada.

Rendimiento.

No contemplado.

5.2.4. Casos de uso del subsistema evaluador

A continuación se describe los casos de usos llevados a cabo en el entorno de evaluación de conocimientos (SIETTE). La idea básica es poder instalar una librería capaz de a partir de los datos .XML generados en el Subsistema Generador de Preguntas poder realizar gran número de preguntas en un tiempo predecible, corto (menor de un segundo) y sin necesidad de tener instalado en dicho entorno de evaluación ninguna aplicación gestora de lógica (por ejemplo Prover9/Mace4), que dificultaría el despliegue y posteriores migraciones.

Los casos de uso RF-03. Construir Preguntas, RF-04. Construir Conceptos Generales y RF-05. Construir Soluciones, son prácticamente idénticos a los del Subsistema Generador de Preguntas, la única diferencia será que en RF-03. Construir Preguntas, las estructuras y datos necesarias para la generación de las preguntas son importadas de los ficheros XML.

RF-10. Instalar Librerías

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El administrador de SIETTE deberá instalar la librería capaz de gestionar los ficheros XML y la realización de las preguntas (enunciado, respuestas tipo test, datos comunes tales como fórmulas e interpretaciones, conceptos generales y soluciones).

Precondiciones.

No contempladas.

Secuencia normal.

1. El administrador de la plataforma de evaluación web, en nuestro caso SIETTE, instala la librería según sus procedimientos internos.

Secuencia alternativa.

No contemplada.

Rendimiento.

No contemplado.

RF-11. Gestionar Ficheros XML .

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El profesor podrá importar todos los ficheros XML que crea oportunos creados en su ordenador personal según RF-02. Almacenar Estructuras XML.

Precondiciones.

1. El administrador del entorno de evaluación habrá instalado la librería necesaria para importar los ficheros XML y generación de las preguntas.

2. El profesor deberá haber importado los ficheros XML creados con el subsistema generador.

Secuencia normal.

1. El profesor accede a SIETTE.

2. El profesor selecciona editar la asignatura Lógica Computacional.

3. El profesor selecciona la opción Gestión de Archivos.

4. El profesor añade los archivos.

Secuencia alternativa.

4. Desde la misma ubicación también se podrían eliminar archivos.

Rendimiento.

No contemplado.

RF-12. Configurar Test.

Objetivos asociados.

OBJ1-Generación de automática de preguntas.

Descripción. El profesor podrá configurar el test con distintas opciones que proporcionarán el comportamiento adecuado en la realización.

Precondiciones.

No contempladas.

Secuencia normal.

1. El profesor accede a SIETTE.
2. El profesor selecciona editar la asignatura Lógica Computacional.
3. El profesor selecciona la opción Test.
4. El profesor selecciona un test determinados.
5. El profesor modifica las distintas opciones del test seleccionado

Secuencia alternativa.

4. El profesor podría dar de alta un test.

Rendimiento.

No contemplado.

RF-13. Realizar Test SIETTE.

Objetivos asociados.

1. OBJ2-Presentación de preguntas
2. OBJ3-Realización de Preguntas.

Descripción. El alumno podrá seleccionar un test que contendrá las preguntas tipo test a realizar.

Precondiciones.

1. El administrador habrá instalado la librería.
2. El profesor habrá configurado el test.
3. El profesor habrá añadido los ficheros XML.

Secuencia normal.

1. El alumno accede a SIETTE
2. El alumno selecciona Hacer un test.
3. El alumno selecciona la asignatura Lógica Computacional.
4. El alumno selecciona un test.
5. El alumno realiza el test.

Rendimiento.

No contemplado.

RF-14. Realizar Preguntas.

Objetivos asociados.

1. OBJ3-Realización de Preguntas.

Descripción. El alumno podrá ir avanzando por las distintas preguntas del test.

Precondiciones.

No contempladas

Secuencia normal.

1. El alumno selecciona una pregunta.
2. El alumno contesta una pregunta.
3. El alumno avanza a la siguiente pregunta.

Secuencia alternativa.

3. El alumno puede volver a preguntas ya contestadas.

Rendimiento.

No contemplado.

RF-15. Consultar Soluciones.

Objetivos asociados.

OBJ3-Presentación de soluciones y conceptos generales.

Descripción. El alumno una vez contestada la preguntas o al final del test, dependiendo de como lo haya configurado el profesor podrá consultar las solución que incluirán un razonamiento del motivo por lo que se consideran correctas o incorrectas.

Precondiciones.

No contempladas.

Secuencia normal.

1. El alumno al terminar el test consulta las soluciones razonadas.

Secuencia alternativa.

1. Dependiendo como configure el profesor el test dichas soluciones podrán ser consultadas una vez contestada cada pregunta.

Rendimiento.

No contemplado.

RF-16. Consultar Conceptos Generales.

Objetivos asociados.

OBJ3-Presentación de soluciones y conceptos generales.

Descripción. El alumno una vez contestada la preguntas o al final del test, dependiendo de como lo haya configurado el profesor podrá consultar los conceptos generales teóricos que le ayudarán a comprender las soluciones adoptadas.

Precondiciones.

No contempladas.

Secuencia normal.

1. El alumno al terminar el test consulta los conceptos generales.

Secuencia alternativa.

1. Dependiendo como configure el profesor el test dichos conceptos generales podrán ser consultadas una vez contestada cada pregunta.

Rendimiento.

No contemplado.

5.3. Requisitos no funcionales

RNF-01. Lenguaje de programación Java.

Descripción. Debido a los lenguajes soportados por SIETTE Java parece el lenguaje capaz de reunir las características para cumplir con éxito el resto de requisitos.

RNF-02. Sistema multiplataforma.

Descripción. El sistema debe ser multiplataforma desde los puntos de vista de profesores, alumnos y entorno de evaluación.

5.4. Definición de un test como factoría de preguntas

Un test se compone de preguntas en las que habrá un título y se propondrán varias respuestas de la cuales sólo una será válida. Además cada una de las preguntas debería de almacenar un razonamiento de su validez o invalidez.

Gracias a la adecuada combinación de los principios de la programación orientada a objetos: herencia, polimorfismo y abstracción, se podrá conseguir, un patrón de preguntas tipo test de forma que reuniese tres características, intrínsecas a la calidad del software y siempre deseables en cualquier desarrollo:

- Independiente: Se debe limitar a la implementación de las preguntas, eliminando todo lo accesorio.
- Extensible: Si se implementasen nuevos tipos de preguntas deberían poderse utilizar con las mínimas modificaciones posibles.
- Mantenible: Si en el futuro se necesitase una nueva funcionalidad, por ejemplo un nuevo tipo de salida, debería permitir analizarse el impacto de forma rápida y con la menor cantidad de esfuerzo posible.

En una primera aproximación, pero errónea, se podría implementar un atributo por cada tipo de pregunta, es decir el test sería una **composición de preguntas**. Una posible representación en UML sería:

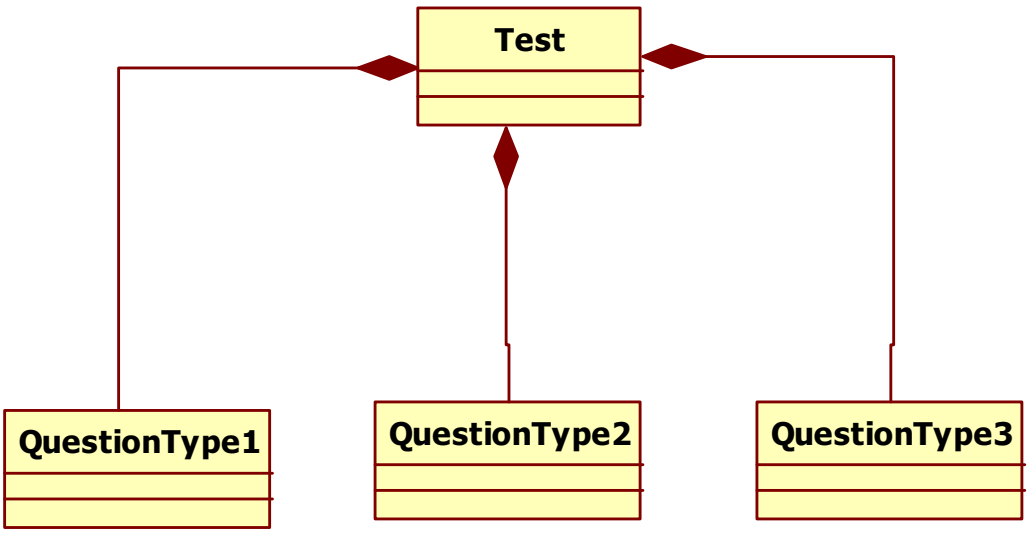


Figura 5.4: Patrón de Preguntas Tipo Test. Composición de preguntas.

Este modelo presenta varios problemas difícil de resolver:

- La creación de una nueva categoría de pregunta provocaría modificación del código, se debería agregar un nuevo atributo, así como su tratamiento.
- No tendría la flexibilidad suficiente para determinar y variar cuantas preguntas se quieren de cada categoría.
- Si se quisiera implementar una nueva especificación, por ejemplo una salida distinta a las ya existentes, se debería revisar toda la implementación.
- Sería bastante difícil incorporar preguntas de distintos temas o asignaturas, provocando la inclusión de nuevos atributos y su tratamiento correspondiente.

En un segunda aproximación, se podría utilizar un patrón "*Factoría Simple*", de esta forma la clase test instanciará objetos de las clases correspondiente a las distintas categorías de preguntas, almacenándolos en un vector o lista, para su posterior tratamiento. A diferencia de la primera aproximación ahora estamos tratando una **agregación de preguntas**, en vez una composición, quedando almacenada una estructura de datos dinámica, que podrá tener más o menos elementos dependiendo del número de preguntas deseadas.

Al ser el la clase test una factoría de preguntas almacenadas en una estructura de datos, las clases de preguntas instanciadas se relacionarán con un interfaz lógico (QuestionIF), que definirá un tipo genérico, así como los métodos obligatorios que deben tener todas las clases de las distintas categorías de preguntas. Como ejemplo se muestra un código Java que representa está idea:

```
private List<QuestionIF> groupQuestionsLogic= new ArrayList<QuestionIF>();
.....
groupQuestionsLogic.add(new QPCLogicPropositionalSatisfacible());
groupQuestionsLogic.add(new QPCLogicPropositionalInSatisfacible());
.....
```

También se implementa una clase madre (Question), que contendrá las estructuras de datos y métodos comunes a cualquier tipo de pregunta, definidos en el interfaz lógico (QuestionIF). Su representación en UML sería:

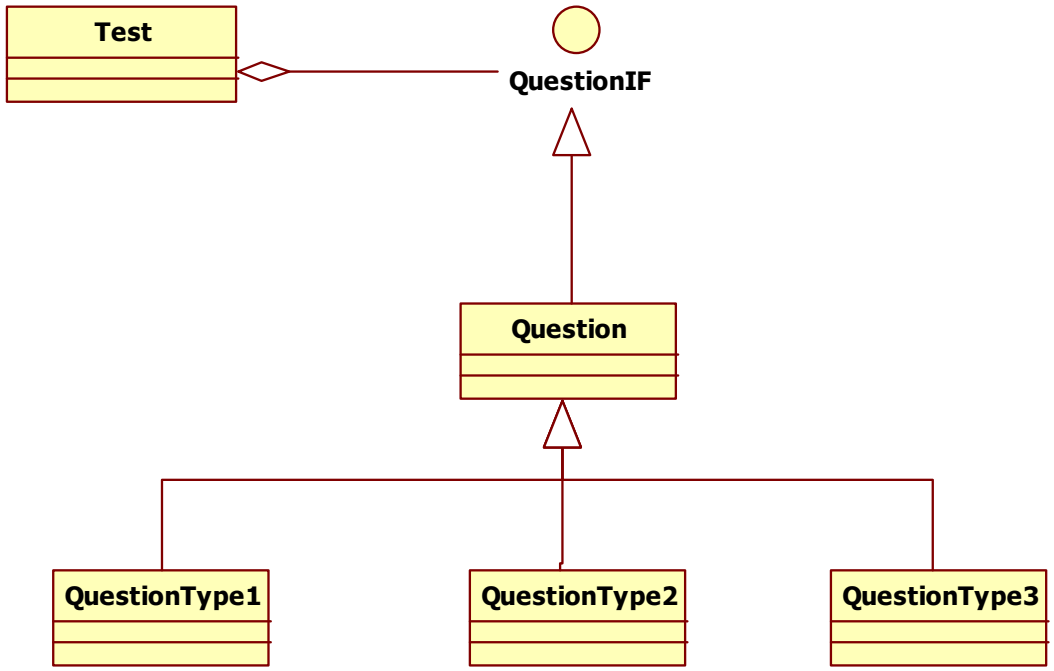


Figura 5.5: Patrón de Preguntas Tipo Test. Factoría de preguntas.

Se puede observar que gracias al interfaz QuestionIF, la clase test podrá almacenar y tratar cualquier tipo de pregunta mientras está se implemente dicho interfaz. Desde un test se crearán y tratarán todas las preguntas, independientemente del tipo que sean.

Con el modelo expuesto se solucionarían los problemas detectados en la primera aproximación:

- La creación de una nueva categoría de pregunta no provocaría prácticamente modificación del código, bastaría con agregar la nueva pregunta al vector o lista.
- Tendría la flexibilidad suficiente para determinar y variar cuantas preguntas se quieren de cada categoría.
- Sería fácil incluir y gestionar una nueva especificación.
- Sería sencillo incorporar preguntas de distintos temas o asignaturas, siempre y cuando se basará en el interfaz QuestionIF.

5.5. Diagramas de estados

Para mayor claridad del funcionamiento habitual del sistema se incluyen a continuación dos diagramas de estados. El primero representa los estados en la configuración de un test en SIETTE,

el segundo representa los estados en la realización un test en SIETE .

5.5.1. Diagrama de estados configuración de un test en SIETTE

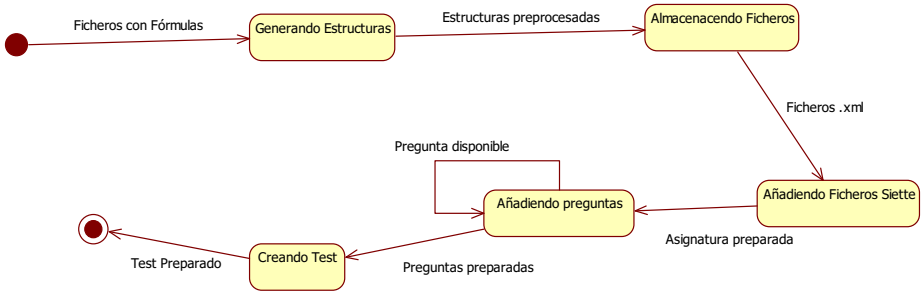


Figura 5.6: Análisis. Diagrama de estado configuración de un test en SIETTE.

5.5.2. Diagrama de estados realización de un test en SIETTE

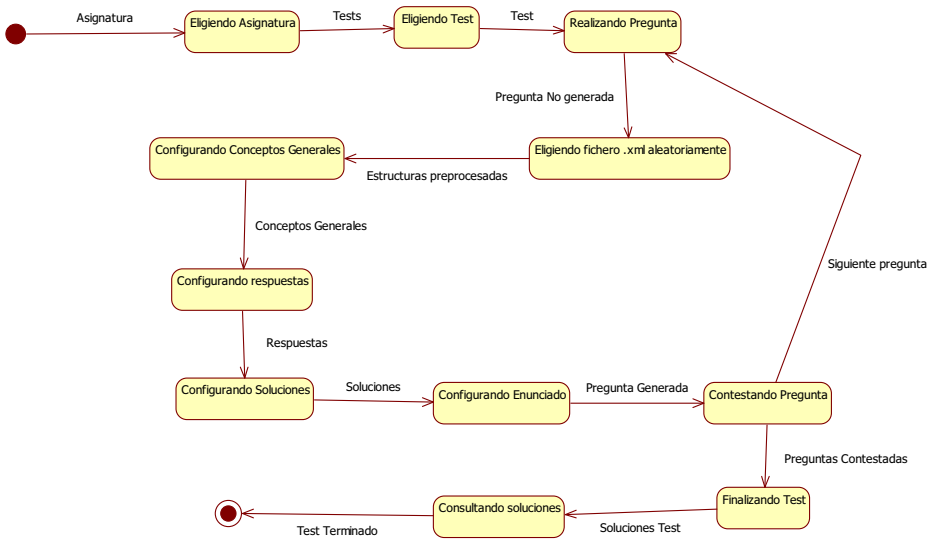


Figura 5.7: Análisis. Diagrama de estados realización de un test en SIETTE.

5.6. Determinación de paquetes de análisis

A pesar de haber subdivido el sistema en dos subsistemas (generador y evaluador), con el ánimo de simplificar la complejidad se fragmentarán a su vez en cuatro paquetes:

1. Paquete Preguntas Tipo Test. Implementa los conceptos referentes a los exámenes tipo test (véase sección 4.2).
2. Paquete Conceptos Lógica Computacional. Implementa los conceptos referentes a lógica computacional (véase sección 4.1).
3. Paquete Generación Preguntas Lógica Computacional. Implementa los conceptos referentes a preguntas de lógica computacional (véase sección 4.3)
4. Paquete Visualización Preguntas. Gracias a este paquete el uso de la librería será lo más sencillo posible.

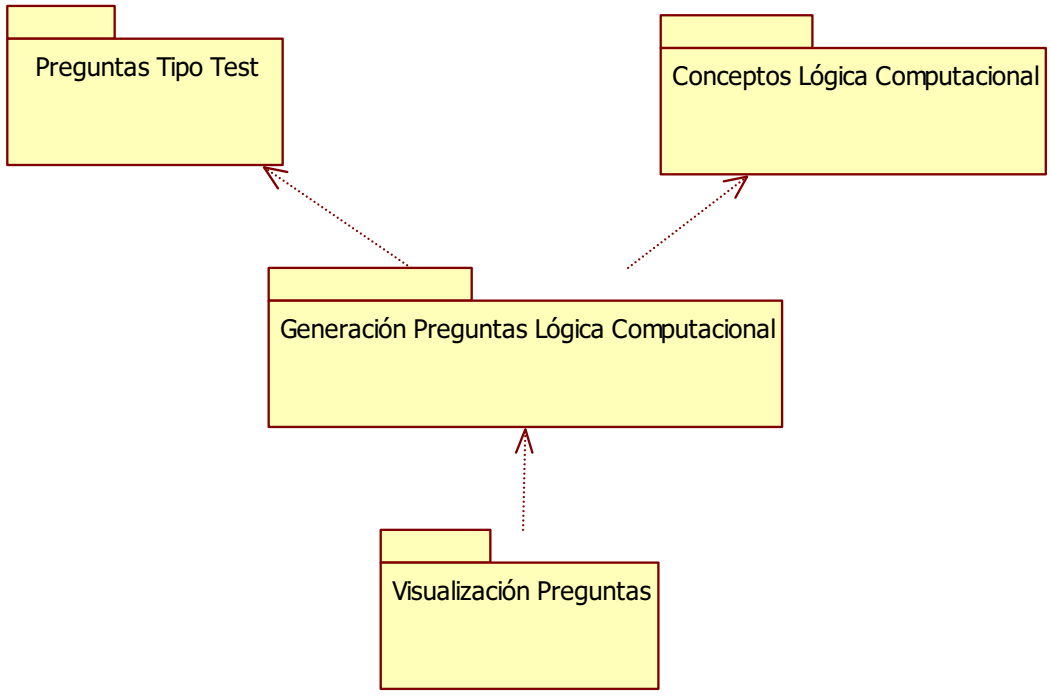


Figura 5.8: Análisis. Diagrama de paquetes.

En el siguiente diagrama se clarifica el uso de los paquetes respecto a los subsistemas generador y evaluador:

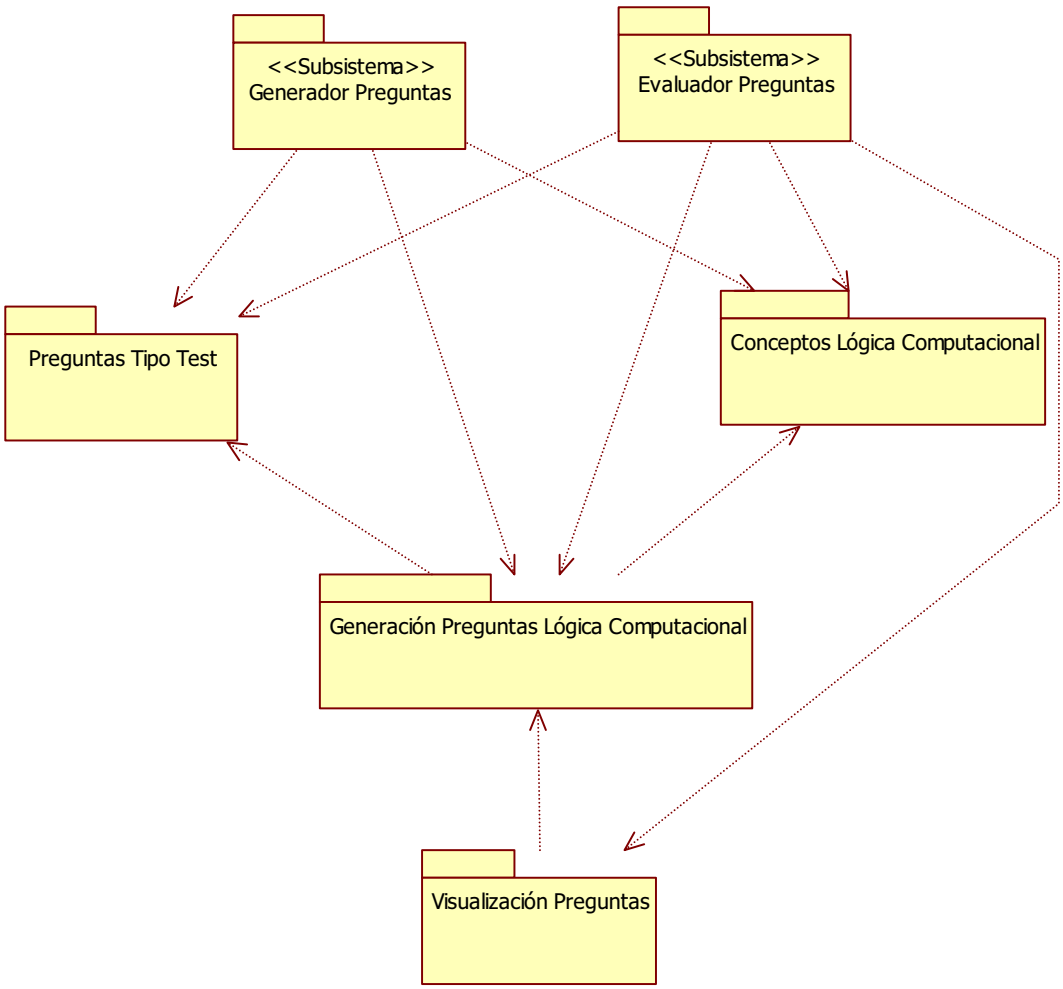


Figura 5.9: Análisis. Diagrama de dependencias subsistemas paquetes.

5.7. Diagrama de despliegue de alto nivel

El despliegue de la librería se llevará a cabo en dos ubicaciones: el ordenador del profesor y en el servidor de evaluación de cocimientos (SIETTE). El alumno únicamente deberá disponer de un navegador web, así como una conexión a Internet.

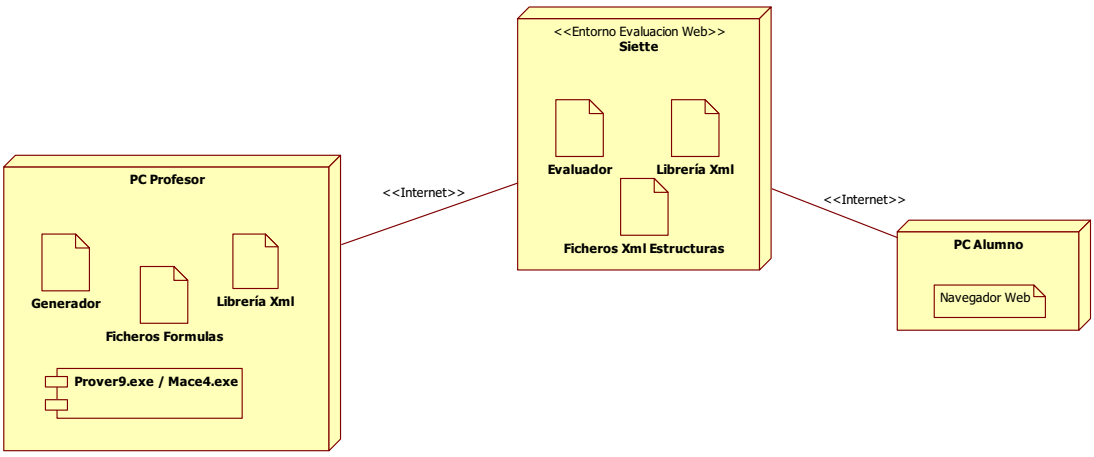


Figura 5.10: Análisis. Diagrama de despliegue.

5.8. Diagramas de secuencias

Para mayor claridad se exponen dos diagramas de secuencias de alto nivel respecto a los principales procesos.

5.8.1. Diagrama de secuencias generación de estructuras

Se muestra un diagrama (véase figura 5.11) que describe la generación de las estructuras desde la que posteriormente se obtendrán las preguntas (corresponden a los descritos en los requisitos funcionales **RF-01. Generar Test** y **RF-02. Almacenar Estructuras XML**).

5.8.2. Diagrama de secuencias generación de preguntas

Se muestra un diagrama (véase figura 5.12) que describe la realización del test por parte de un alumno (corresponden a los descritos en los requisitos funcionales **RF-03. Construir Preguntas**, **RF-04. Construir Conceptos Generales**, **RF-05. Construir Soluciones**, **RF-13. Realizar Test SIETTE**, **RF-14. Realizar Preguntas**, **RF-15. Consultar Soluciones y RF-16. Consultar Conceptos Generales**).

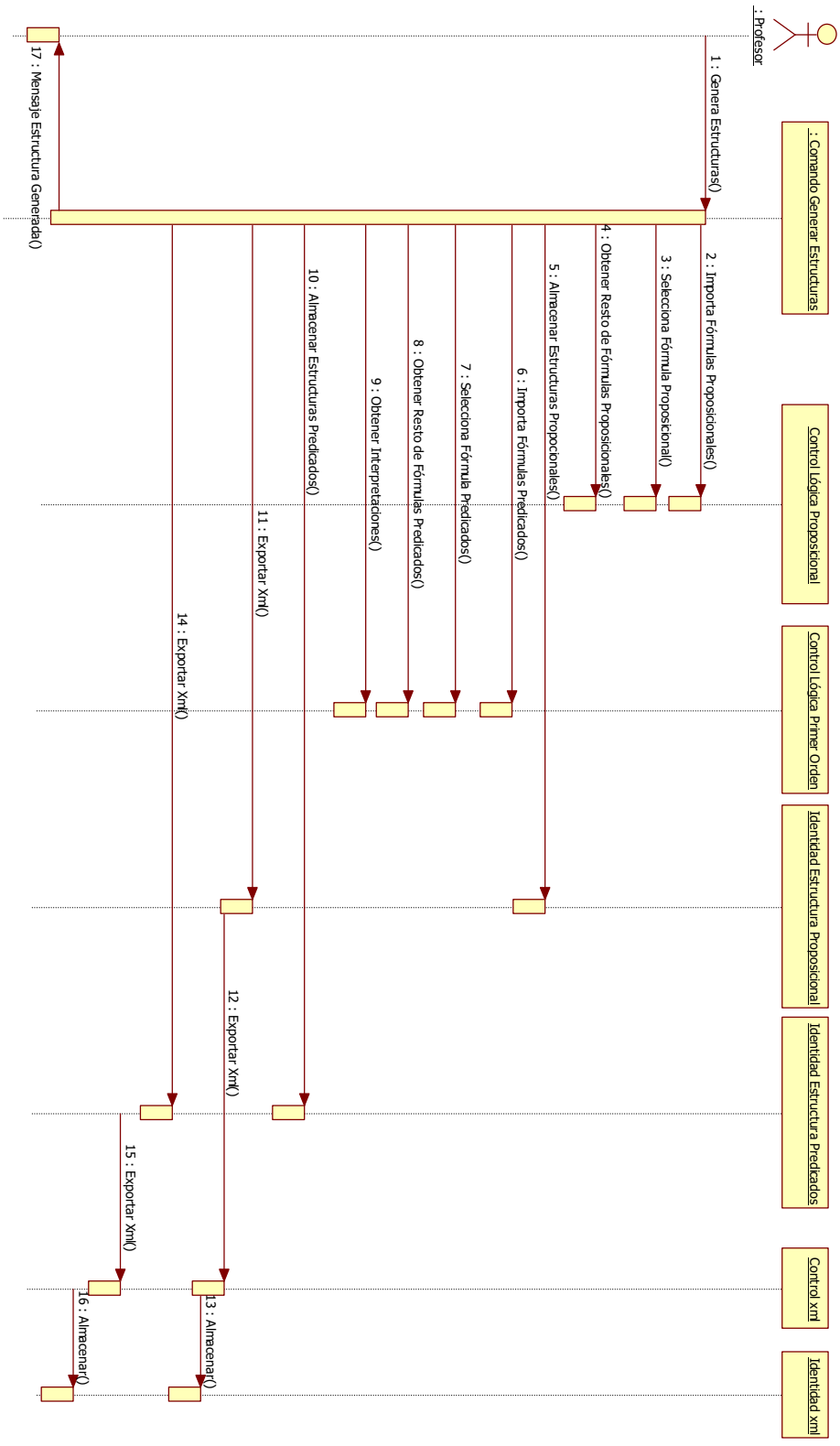


Figura 5.11: Análisis. Diagrama de secuencia generación estructura.

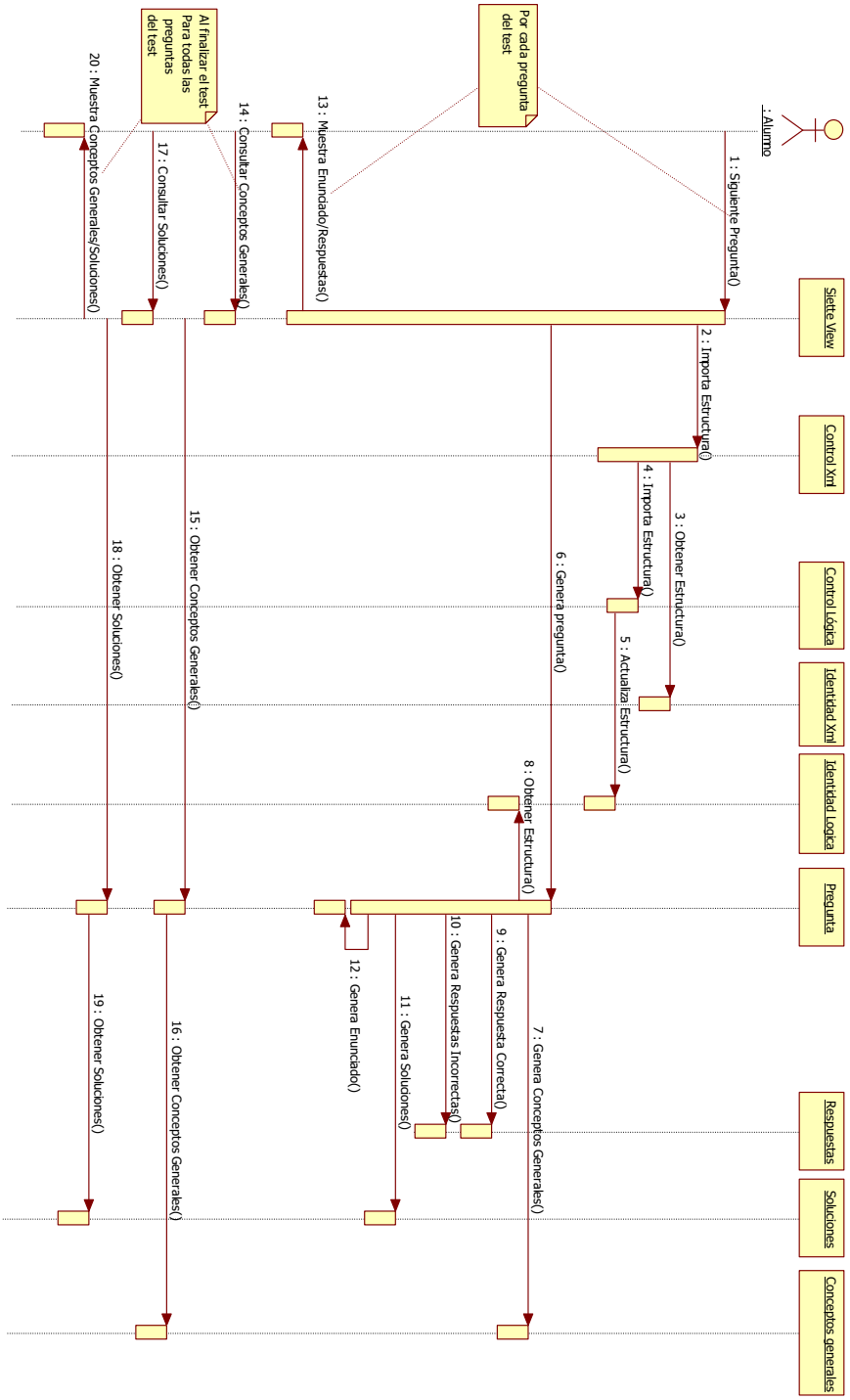


Figura 5.12: Análisis. Diagrama de secuencia generación preguntas.

6. Diseño

6.1. Especificación del entorno tecnológico

Como ya se indicó en el estudio del diagrama de casos de usos (véase subsección 5.2.1) el proyecto se divide en dos subsistemas (generador y evaluador) que junto al solución elegida en el estudio de viabilidad (véase sección 3.3)se puede deducir que el sistema tendrá dos entornos tecnológicos:

- **Entorno tecnológico generador**. Se ubicará en el ordenador personal del profesor, deberá tener instalado el siguiente software:
  - Prover 9 / Mace 4.
  - Máquina virtual de Java 1.5 ó superior.
  - Librería JDOM 1.1 para tratamiento de ficheros XML.
  - Además de forma adicional si el profesor deseara gestionar un examen en formato LaTeX, así como su exportación a formato PDF, debería instalar algún software que le facilitase esta tarea. Durante el desarrollo del proyecto se ha utilizado con éxito la distribución de LaTeX **MiKTeX** [MiKTeX, 2012], así como el editor **TeXnicCenter** [TeXnicCenter, 2012], aunque se debe destacar que dicho software sólo es soportado en sistemas operativos Windows.
  - Respecto al sistema operativo, no se han detectado restricciones a la hora de poder utilizar aquél que el profesor estime oportuno debido a que existen distribuciones de Prover 9 y Mace 4 para Linux, Windows y otros; además de la posibilidad de disponer de Máquinas Virtuales de Java 1.5 para múltiples plataformas. No obstante, si el profesor requiriera de un editor de LaTeX (opcional) debería buscar alguno que se adaptase a la versión de sistema operativo que tuviera instalado.
- **Entorno tecnológico evaluador**. Se ubicará en el servidor web del sistema de evaluación de conocimientos.
  - SIETTE. Este entorno de evaluación de conocimientos no dependerá del profesor, siendo transparente su configuración en el desarrollo del proyecto. No obstante, es de especial interés tener en cuenta que las librerías generadas con Java deben de ser compiladas para una versión de Máquina Virtual 1.5.

- Librería JDOM 1.1 para tratamiento de ficheros XML.
- El alumno bastará con que disponga un acceso a Internet y un navegador web.

6.2. Identificación de subsistemas de diseño

A partir de los paquetes de análisis (véase sección 5.6) se obtienen los paquetes de diseño que se muestran en la figura.

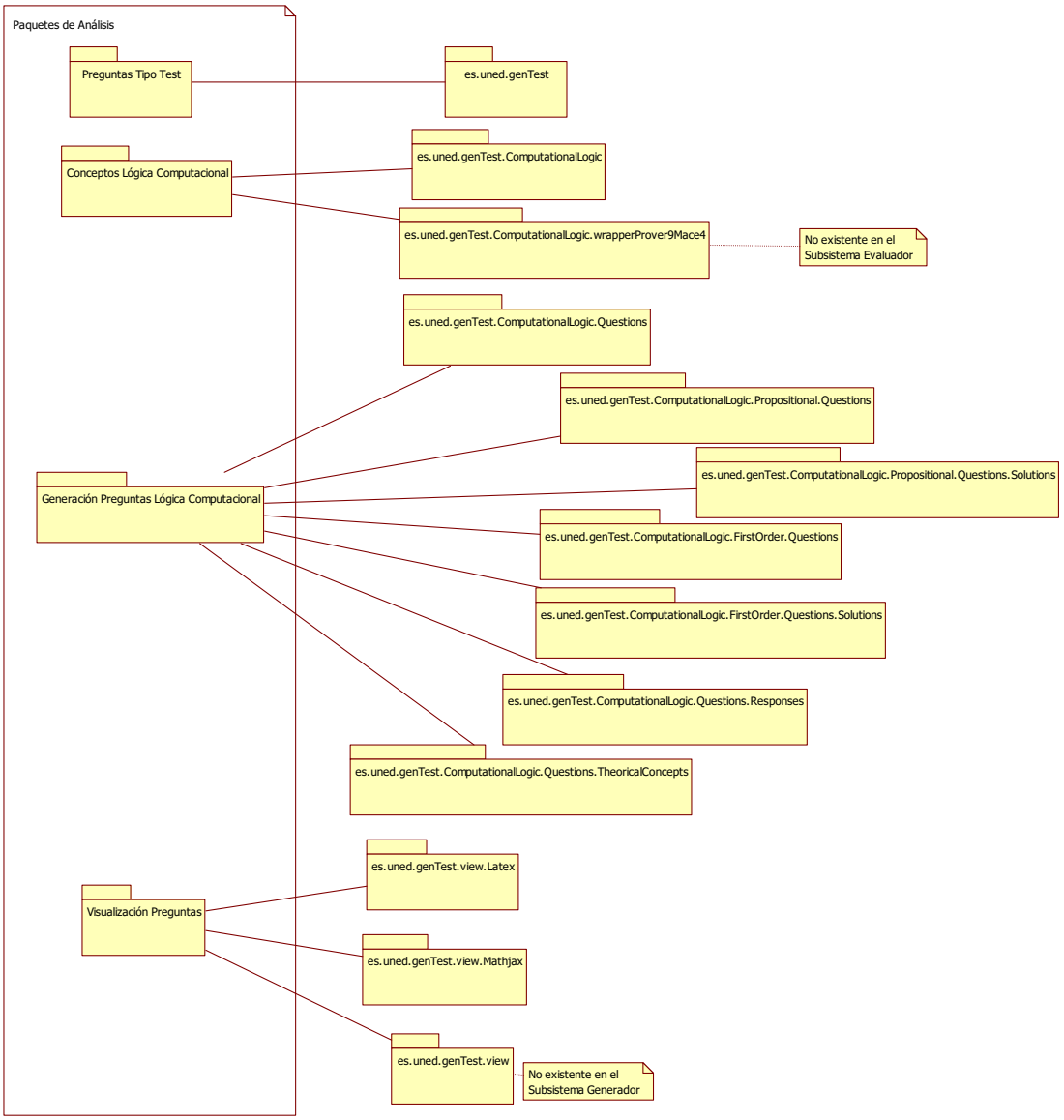


Figura 6.1: Análisis. Diagrama de paquetes de diseño.

6.3. Identificación de clases

Del estudio del comportamiento de los casos de uso y la subdivisión en paquetes de diseño se deducen las clases que se exponen a continuación. Para mayor claridad se han representado las clases, así como las asociaciones, composiciones y agregaciones mediante diagramas de clases, indicando la existencia de los atributos y métodos más importantes o representativos.

6.3.1. Diagrama de clases de preguntas tipo test

Para aprovechar al máximo las características de la programación orientada a objetos (polimorfismo y herencia), se puede observar que tanto las respuestas como las soluciones se implementan según los interfaces ResponseIF y SolutionIF, que tienen sus propias clases Reponse y Solution, que implementan los métodos comunes, las cuales se especializarán mediante clases que heredan de éstas.

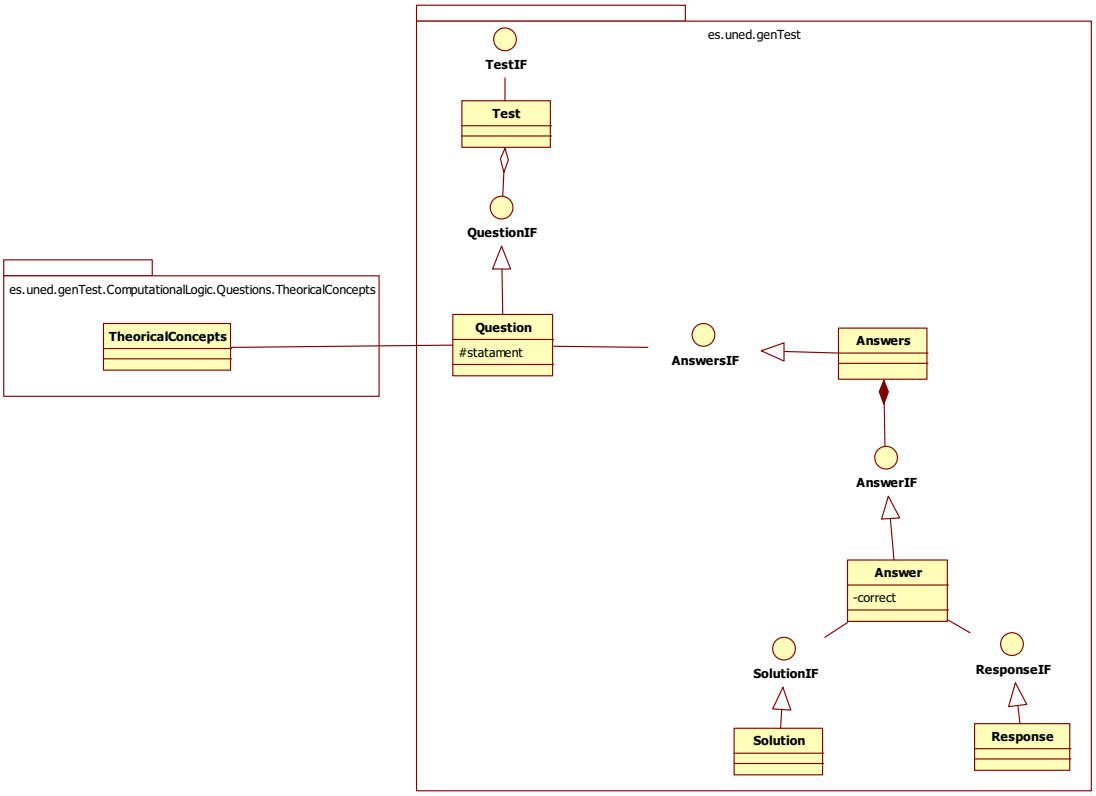


Figura 6.2: Diseño. Diagrama de clases preguntas tipo test.

6.3.2. Descripción del paquete gestor de Prover9 y Mace4

Se ha diseñado un paquete denominado **wrapperProver9Mace4**, que contiene dos clases, gestoras de Prover9 y Mace4, a las que se les aplica el patrón de diseño singleton, con la intención de garantizar que cada una de las clases que tratan los ejecutables de Prover9 ó Mace4 sólo tengan una instancia y proporcionar un punto de acceso global a ellas; dichas clases se han denominado SingletonProver9 y SingletonMace4. Destacar que el paquete wrapperProver9Mace4, correspondería a la implantación "*Capa 1. Software externo. Prover9 / Mace4*", (véase subsección 4.1.1).

La clase **SingletonProver9** tiene cuatro métodos públicos:

- getSingletonProver9: Permite crear una instancia de la clase en cuestión aplicando el patrón de diseño singleton.
- exec: Ejecuta Prover9, devolviendo un valor verdadero en el caso de que el grupo de fórmulas sea insatisfacible y falso en el caso de que sea satisfacible.
- getClausifyFormulas: Devuelve el grupo de fórmulas en forma clausulada.
- getProofs: Devuelve una prueba de insatisfacibilidad, demostrando mediante resolución que se puede deducir una cláusula vacía.

La clase **SingletonMace4** tiene tres métodos públicos:

- getSingletonMace4: Permite crear una instancia de la clase en cuestión aplicando el patrón de diseño singleton.
- getDomainSize: Devuelve el tamaño del dominio de la interpretación que hace verdadera al grupo de fórmulas estudiado.
- getRelations: Devuelve una interpretación que hace verdadera al grupo de fórmulas estudiado.

6.3.3. Descripción del paquete gestor de lógica computacional

Se ha diseñado un paquete denominado **ComputationalLogic** que contiene todas las clases necesarias para gestionar los conceptos básicos de lógica computacional utilizados en este proyecto.

A grandes rasgos se divide en dos conjuntos de clases: una clase base, denominada singletonComputerLogic, que implementa los métodos de los conceptos básicos ((in)satisfacibilidad, equivalencia, tautología,...) y otras clases de más alto nivel que se encargan de gestionar fórmulas proposicionales ó de predicados de primer orden.

6.3.3.1. La clase singleton de lógica computacional

Se ha diseñado la clase **singletonComputerLogic** que dará como resultado la construcción de la "*Capa 2. Lógica Computacional*" (véase subsección 4.1.1). Implementa los métodos correspondientes a los conceptos básicos de lógica computacional, invocando los métodos públicos ofrecidos por las clases SingletonProver9 y SingletonMace4.

Se le ha aplicado el patrón de diseño singleton para restringir la creación de más de un objeto, de esta forma por ser un recurso común se garantiza que sólo tenga una instancia y asegurando un punto de acceso global a ellas.

Tiene ocho métodos públicos:

- isInsatisfacible: Devuelve un valor verdadero en el caso de que un grupo de fórmulas pasado como parámetro sea insatisfacible y falso en el caso de que sea satisfacible.
- isSatisfacible: Devuelve un valor verdadero en el caso de que un grupo de fórmulas pasado como parámetro sea satisfacible y falso en el caso de que sea insatisfacible.
- isEquivalent: Devuelve un valor verdadero en el caso de que dos fórmulas pasadas como parámetros sean equivalentes y falso en caso contrario.
- isTautologia: Devuelve un valor verdadero en el caso de que un grupo de fórmulas pasado como parámetro sea tautología y falso en caso contrario.
- getDomainSize: En el caso de que el grupo de fórmulas estudiado sea satisfacible, devuelve el tamaño del dominio de una interpretación que hace verdadera.
- getRelations: En el caso de que el grupo de fórmulas estudiado sea satisfacible, devuelve una interpretación que hace verdadera al grupo de fórmulas estudiado.
- getClausifyFormulas: En el caso de que el grupo de fórmulas estudiado sea insatisfacible, devuelve el grupo de fórmulas en forma clausulada.
- getProofs: En el caso de que el grupo de fórmulas estudiado sea insatisfacible, devuelve una prueba de insatisfacibilidad, demostrando mediante resolución que se puede deducir una cláusula vacía.

6.3.3.2. Las clases fórmula y grupo de fórmulas

Se han diseñado las clases **CLogicGroupFormulas** y **CLogicFormula** que darán como resultado la construcción de *" Capa 3. Grupo de fórmulas"* y *" Capa 4. Fórmulas"*, respectivamente (véase subsección 4.1.1).

La clase **CLogicGroupFormulas** gestiona un grupo de fórmulas de lógica proposicional ó de lógica de predicados de primer orden, por ello tendrá las propiedades necesarias para almacenar una prueba (clase *PCLogicListProofs*), así como su forma clausulada (clase *PCLogicListClausifyFormula*) en el caso de que fueran insatisfacibles ó una interpretación verdadera (clase *PCLogicListRelations*) en el caso de ser satisfacible.

Implementa los métodos públicos necesarios para gestionar las fórmulas del grupo:

- addFormula: Agrega fórmula(s) al grupo.
- deny: Denegar el grupo.
- equal: Devuelve verdadero en el caso de que sea igual a otro grupo de fórmulas pasado como parámetro.

- disorder: Desordena el grupo de fórmulas.
- isInsatisfacible: Devuelve un valor verdadero en el caso de que un grupo de fórmulas pasado como parámetro sea insatisfacible y falso en el caso de que sea satisfacible.
- isSatisfacible: Devuelve un valor verdadero en el caso de que un grupo de fórmulas pasado como parámetro sea satisfacible y falso en el caso de que sea insatisfacible.
- isTautologia: Devuelve un valor verdadero en el caso de que un grupo de fórmulas pasado como parámetro sea tautología y falso en caso contrario.
- Distintos métodos para transformar el grupo de fórmulas a LaTeX y XML.
- Distintos métodos para la gestión pruebas, fórmulas clausuladas e interpretaciones (puts y gets).

La clase **CLogicFormula** gestiona una fórmula de lógica proposicional ó de lógica de predicados de primer orden, cabe destacar que se apoya en los métodos de la clase *CLogicGroupFormulas*, por considerarse que una fórmula es un grupo de fórmulas con un sólo elemento.

Para una mejor comprensión de las propiedades de esta clase en primer lugar se debe estudiar las partes de una fórmula, si por ejemplo tenemos la fórmula proposicional $X_1 = p \vee -q$, diremos que:

- X es la propiedad *alias*.
- $_1$ es la propiedad *subalias*.
- $p \vee -q$ es la propiedad *formula*.
- Además tendrá una propiedad *deny* que estará a falso si no está denegada, en el ejemplo estará a falso.

Implementa los métodos públicos necesarios para gestionar la fórmula:

- isDeny: Devuelve falso en el caso de que la fórmula esté negada y verdadero en caso contrario.
- isInsatisfacible: Devuelve un valor verdadero en el caso de que una fórmula sea insatisfacible y falso en el caso de que sea satisfacible.
- isSatisfacible: Devuelve un valor verdadero en el caso de que una fórmula sea satisfacible y falso en el caso de que sea insatisfacible.
- isEquivalent: Devuelve un valor verdadero en el caso de que una fórmula pasada como parámetro sea equivalente y falso en caso contrario.
- Distintos métodos para transformar la fórmula a LaTeX y XML.
- Distintos métodos para la gestión de las propiedades (puts y gets).

6.3.3.3. Diagrama de clases de lógica computacional

Se expone el diagrama de clases que representa los conceptos generales de lógica computacional.

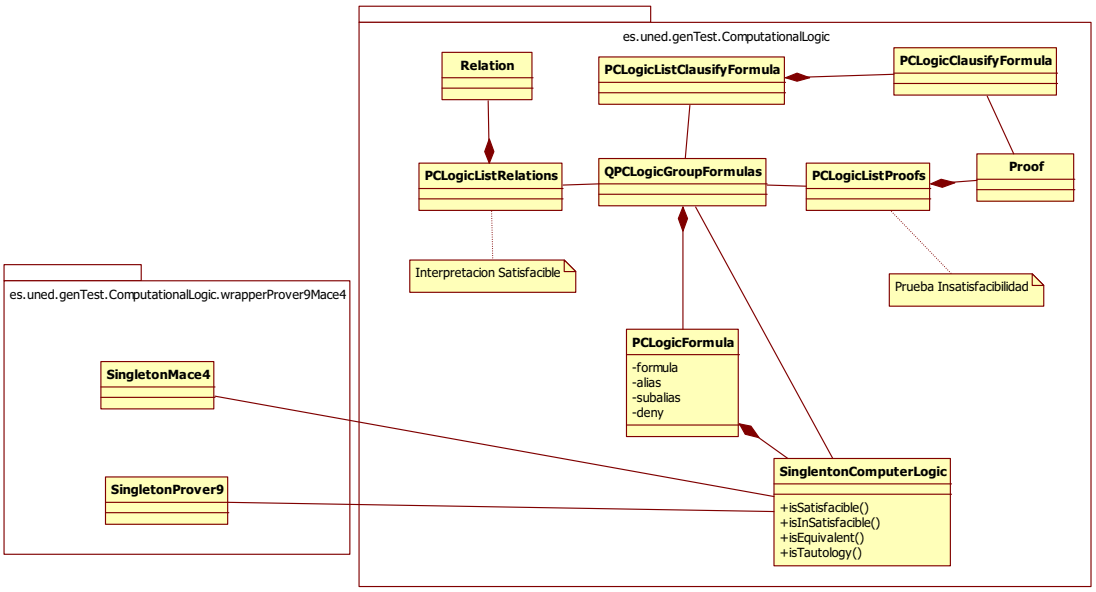


Figura 6.3: Diseño. Diagrama de clases conceptos básicos de lógica computacional.

6.3.4. Descripción del paquete preguntas de lógica computacional

Se ha diseñado un paquete denominado **ComputationalLogic.Questions.** Este paquete contiene los interfaces y las clases especializadas necesarias capaz de gestionar preguntas de lógica. También incluye las clases ayudantes en las que se apoyarán los algoritmos de generación automática de datos.

Dicha especialización se consigue mediante la implantación de la clase **QuestionComputerLogic**, estableciendo herencia de métodos comunes y atributos de la clase **Question** (véase subsección 6.3.1) e implementando los interfaces **QuestionComputerLogicIF**, **QuestionComputerLogicLatexIF**, **QuestionComputerLogicMathjaxIF**, asegurando la implementación de los métodos obligatorios tanto en su gestión propiamente dichas, como en sus salidas.

Además se necesita, una mayor especialización, ya que dentro de la categoría de lógica computacional, tenemos dos grandes tipos:

- Lógica Proposicional.
- Lógica de Predicados de Primer Orden.

Por tanto, se han creado dos clases **QuestionComputionalLogicPropositional** y **QuestionComputionalLogicFirstOrder**, que heredan métodos y atributos de la clase **QuestionComputerLogic**. Dichas clases serán las responsables de hacer uso de sus respectivas clases ayudantes,

así como de la gestión de las clases que almacenarán los datos generados automáticamente. Servirán de base a cada uno de los modelos de preguntas finales que se describen en la subsección 6.3.5.

Para asistir a las clases que implementan los distintos tipos de preguntas se ha implementado dos ayudantes:

- **singletonHelperCLPropositionalTest**: Ayudante de las preguntas de lógica proposicional.
- **singletonHelperCLFirstOrderTest**: Ayudante de las preguntas de lógica de predicados de primer orden.

Dichos ayudantes almacenarán los datos generados, en sus respectivas clases de datos **singletonDataCLPropositionalTest** será utilizada por **singletonHelperCLPropositionalTest** y **singletonDataCLFirstOrderTest** corresponde a **singletonHelperCLFirstOrderTest**. Tanto **singletonDataCLPropositionalTest** como **singletonDataCLFirstOrderTest** implementan el interfaz **singletonDataCLTestIF** para conseguir establecer la obligatoriedad de la implementación de los métodos de ambas clases.

En la implementación, tanto las clases ayudantes, como las que almacenan los datos generados, se ha utilizado el patrón de diseño singleton para restringir la creación de más de un ayudante o clases de datos, de esta forma por ser un recurso común se garantiza que sólo tenga una instancia y asegurando un punto de acceso global a ellas.

Se ha implementado un ayudante (**singletonHelperXml**), cuya misión será la exportación de los datos almacenados en las clases **singletonDataCLPropositionalTest** y **singletonDataCLFirstOrderTest** a un fichero XML, así como su posterior importación desde dicho fichero XML a las clases de datos. De esta forma, los datos generados de forma automática se podrán ser portables y utilizados de forma desconectada. Como en casos anteriores y por las mismas razones, su implantación, se ha basado en un patrón de diseño singleton.

6.3.4.1. Diagrama de clases preguntas de lógica computacional

Se expone el diagrama de clases que representa las preguntas de lógica computacional, así como los ayudantes necesarios.

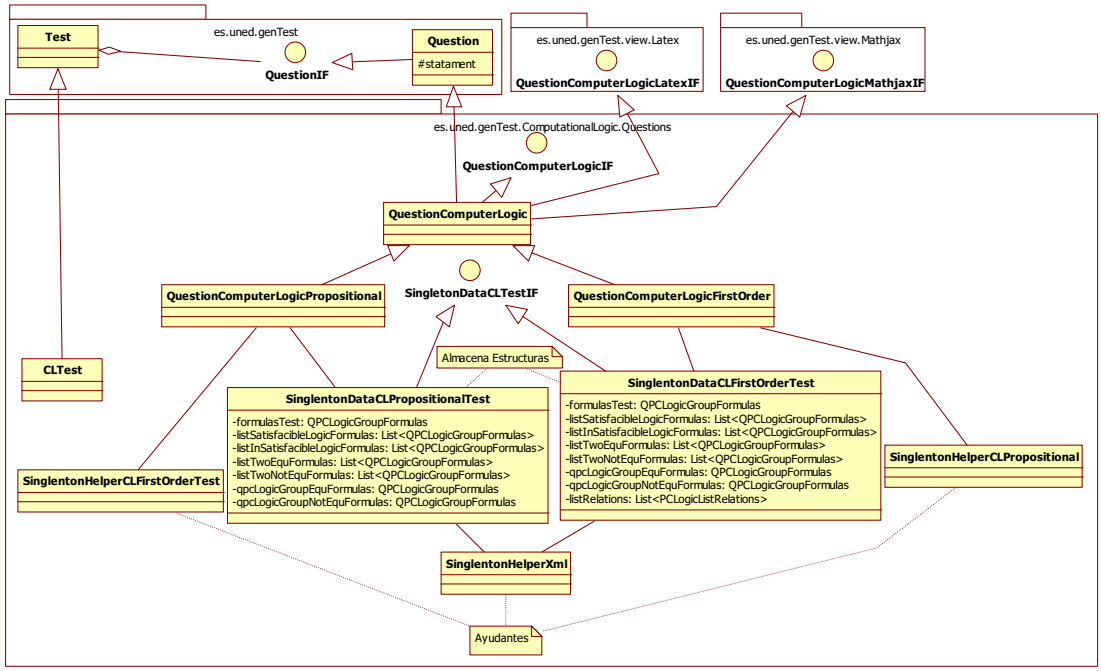


Figura 6.4: Diseño. Diagrama de clases de preguntas de lógica computacional.

6.3.5. Modelos de preguntas

Tomando como referencia la teoría y exámenes de años anteriores de la asignatura de cuarto curso "Lógica Computacional" de la carrera "*Ingeniería Técnica Superior en Informática*" de la Universidad Nacional a Distancia (UNED), se han seleccionado modelos de preguntas, que serán generadas de forma automática por la librería implementada en este proyecto.

Antes de pasar a detallar cada uno de los veintitrés modelos construidos, indicar que se han implementado dos paquetes:

- **ComputationalLogic.Propositional.Questions**.
- **ComputationalLogic.FirstOrder.Questions**.

En ellos se incluirán las clases necesarias para implementar los modelos de preguntas que a continuación se detallarán que además serán especializaciones de las clases **QuestionComputionalLogicPropositional** y **QuestionComputionalLogicFirstOrder** (véase subsección 6.3.4).

También existen otros paquetes especializados en las distintas partes de una pregunta:

- Soluciones:
  - **ComputationalLogic.Propositional.Questions.Solutions**
  - **ComputationalLogic.FirstOrder.Questions.Solutions**
- Respuestas:
  - **ComputationalLogic.Questions.Responses**
- Conceptos Teóricos:
  - **ComputationalLogic.Questions.TheoricalConcepts**

6.3.5.1. Lógica proposicional

Incluidas en el paquete **ComputationalLogic.Propositional.Questions** se han implementado las clases que corresponden a los distintos modelos de preguntas de lógica proposicional:

QPCLogicPropositionalSatisfacible

Se presenta tres grupos de tres fórmulas proposicionales cada uno, preguntando cual de ellos es satisfacible (véase pregunta 1 del Apéndice C). En la solución, fundamentada en el concepto básico de satisfacibilidad (véase subsección 4.1.2.1), se presentan los siguientes conceptos teóricos:

Los métodos para deducir la satisfacibilidad pueden ser entre otros: Un método deductivo, en el caso de Lógica Proposicional también se puede utilizar tablas de verdad cuando el número de símbolos no sea muy elevado. Opcionalmente se puede utilizar un método de resolución llegando a soluciones no vacías, también se podría utilizar árboles semánticos llegando a la conclusión de que es satisfacible si este no se puede cerrar. Los métodos para deducir la insatisfacibilidad pueden ser entre otros: Mediante tablas de verdad en Lógica Proposicional, siempre y cuando la cantidad de símbolos lo permita, comprobando que todas las valoraciones dan como resultado valores falsos (0). Cuando no disponemos del colchón de las tablas de verdad, (Lógica de Primer Orden o cardinalidad elevada en el caso de la Lógica Proposicional) se podrá llegar a una cláusula vacía (por resolución) ó a un árbol semántico cerrado.

Indicando para cada respuesta si es correcta o incorrecta, así como una interpretación que satisfaga el grupo de fórmulas proposicionales ó el razonamiento de la obtención de una cláusula vacía mediante resolución en caso de que sean insatisfacibles.

Para mayor claridad se muestra un ejemplo de solución para este tipo de preguntas:

(a) $\{X_1, \neg X_2, \neg X_3\}$. Incorrecta.

En este caso se puede comprobar en la resolución expuesta respecto a $\{X_1, \neg X_2, \neg X_3\}$, se llega a una claúsula vacía.

En primer lugar se hayan las fórmulas en su forma clausulada:

$X_1 = \{\{\neg t\}, \{\neg q\}, \{\neg p\}\}$

$\neg X_2 = \{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$

$\neg X_3 = \{\{p, q\}, \{\neg r, \neg s\}\}$

Resolución:

1. $\{\{\neg t\}, \{\neg q\}, \{\neg p\}\}$ # Premisa
2. $\{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$ # Premisa
3. $\{\{p, q\}, \{\neg r, \neg s\}\}$ # Premisa
4. $\{\neg p\}$ # Cláusula de la premisa 1
5. $\{p\}$ # Cláusula de la premisa 2
6. [ ] # Cláusula vacía (5,4)

(b) $\{\neg X_1, \neg X_2, X_3\}$. Incorrecta.

En este caso se puede comprobar en la resolución expuesta respecto a $\{\neg X_1, \neg X_2, X_3\}$, se llega a una claúsula vacía.

En primer lugar se hayan las fórmulas en su forma clausulada:

$\neg X_1 = \{\{t, q, p\}\}$

$\neg X_2 = \{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$

$X_3 = \{\{\neg p, r\}, \{\neg p, s\}, \{\neg q, r\}, \{\neg q, s\}\}$

Resolución:

1. $\{\{t, q, p\}\}$ # Premisa
2. $\{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$ # Premisa
3. $\{\{\neg p, r\}, \{\neg p, s\}, \{\neg q, r\}, \{\neg q, s\}\}$ # Premisa
4. $\{p\}$ # Cláusula de la premisa 2
5. $\{\neg r\}$ # Cláusula de la premisa 2
6. $\{\neg p, r\}$ # Cláusula de la premisa 3
7. [ ] # Cláusula vacía (6,4,5)

(c) $\{\neg X_1, X_2, \neg X_3\}$. Correcta.

En este caso, se podría realizar la tabla de verdad verificando, por ejemplo, que la interpretación $I(p, q, r, s, t) = (f, v, f, f, f)$ satisface $\{\neg X_1, X_2, \neg X_3\}$.

QPCLogicPropositionalInSatisfacible

Similar a la anterior, pero en este caso se pregunta que grupo es insatisfacible (véase pregunta 2 del Apéndice C).

QPCLogicPropositionalTautology

Se presentan tres grupos de fórmulas con el formato $\{formula_1 \wedge formula_2 \rightarrow formula_3\}$ y se pide que se señale la tautología (véase pregunta 3 del Apéndice C) . En la solución, fundamentada en el concepto básico de validez (véase subsección 4.1.2.2), se presentan los siguientes conceptos teóricos:

Una fórmula válida es aquélla que es verdadera frente a cualquier interpretación. Las tautologías son fórmulas válidas. Se puede observar que:
- Si niega una fórmula insatisfacible, la fórmula resultante es una tautología.
- Si niega una tautología, la fórmula resultante es insatisfacible.

Para decidir la validez de una fórmula, de nuevo, el procedimiento semántico extensivo requiere recorrer toda la tabla de verdad. Los resultados negativos se pueden obtener más rápidamente: basta encontrar la primera interpretación que no satisface la fórmula. Pero los resultados positivos requieren una comprobación completa..

Se resalta que:una fórmula ψ es tautología si y sólo si $\neg\psi$ es insatisfacible..

Estos conceptos son extensibles a la Lógica de Primer Orden: de esta forma cualquier método de decisión de la (in)satisfacibilidad permite decidir la validez, y viceversa.

Un caso particular es el estudio de validez del conjunto de fórmulas $((\varphi_1 \wedge ... \wedge \varphi_n) \rightarrow \varphi)$, es equivalente a $(\neg(\varphi_1 \wedge ... \wedge \varphi_n) \vee \varphi)$. Por tanto, su negación es equivalente a la fórmula $(\varphi_1 \wedge ... \wedge \varphi_n \wedge \neg\varphi)$, si ésta es insatisfacible quedaría demostrado que el conjunto de fórmulas original es una tautología.

Indicando para cada respuesta si es correcta o incorrecta, así como una respuesta razonada basada en la obtención de una cláusula vacía mediante resolución de un grupo de fórmulas negado en caso de que sea tautología ó la obtención de una interpretación que satisfaga un grupo de fórmulas negado en el caso no ser tautología.

Para mayor claridad se muestra un ejemplo de solución para este tipo de preguntas :

(a) $\{\neg X_1 \wedge \neg X_2 \rightarrow \neg X_3\}$. Incorrecta.
Existen varias posibilidades para demostrar que es una tautología, por ejemplo si fuese insatisfacible la negación de $\{\neg X_1 \wedge \neg X_2 \rightarrow \neg X_3\}$, en este caso en concreto, quizá otro método más sencillo sería llegar a una claúsula vacía de $\{\neg X_1 \wedge \neg X_2 \wedge X_3\}$
En primer lugar se hayan las fórmulas en su forma clausulada:
$\neg X_1 = \{\{t, q, p\}\}$
$\neg X_2 = \{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$
$X_3 = \{\{\neg p, r\}, \{\neg p, s\}, \{\neg q, r\}, \{\neg q, s\}\}$
Resolución:
1. $\{\{t, q, p\}\}$ # Premisa
2. $\{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$ # Premisa
3. $\{\{\neg p, r\}, \{\neg p, s\}, \{\neg q, r\}, \{\neg q, s\}\}$ # Premisa
4. $\{p\}$ # Cláusula de la premisa 2
5. $\{\neg r\}$ # Cláusula de la premisa 2
6. $\{\neg p, r\}$ # Cláusula de la premisa 3
7. [ ] # Cláusula vacía (6,4,5)

(b) $\{\neg X_1 \wedge X_2 \rightarrow X_3\}$. Correcta.
Existen varias posibilidades para demostrar que no es una tautología, por ejemplo si fuese satisfacible la negación de $\{\neg X_1 \wedge X_2 \rightarrow X_3\}$, en este caso en concreto, quizá otro método más sencillo sería obtener una interpretación verdadera para la negación de las fórmulas, para ello se podría realizar la tabla de verdad de $\{\neg X_1 \wedge X_2 \wedge \neg X_3\}$ verificando, por ejemplo, que se puede obtener una interpretación verdadera para $I(p, q, r, s, t) = (f, v, f, f, f)$. Al ser satifacible no será tautología.

(c) $\{X_1 \wedge \neg X_2 \rightarrow X_3\}$. Incorrecta.
Existen varias posibilidades para demostrar que es una tautología, por ejemplo si fuese insatisfacible la negación de $\{X_1 \wedge \neg X_2 \rightarrow X_3\}$, en este caso en concreto, quizá otro método más sencillo sería llegar a una claúsula vacía de $\{X_1 \wedge \neg X_2 \wedge \neg X_3\}$
En primer lugar se hayan las fórmulas en su forma clausulada:
$X_1 = \{\{\neg t\}, \{\neg q\}, \{\neg p\}\}$
$\neg X_2 = \{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$
$\neg X_3 = \{\{p, q\}, \{\neg r, \neg s\}\}$
Resolución:
1. $\{\{\neg t\}, \{\neg q\}, \{\neg p\}\}$ # Premisa
2. $\{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$ # Premisa
3. $\{\{p, q\}, \{\neg r, \neg s\}\}$ # Premisa
4. $\{\neg p\}$ # Cláusula de la premisa 1
5. $\{p\}$ # Cláusula de la premisa 2
6. [ ] # Cláusula vacía (5,4)

QPCLogicPropositionalNotTautology

Similar a la anterior, pero en este caso se pregunta que consecuencia no es tautología (véase pregunta 4 del Apéndice C).

QPCLogicPropositionalEquivalent

Dada una fórmula proposicional, se pide que seleccione la fórmula equivalente a elegir en tres

posibles repuestas. (véase pregunta 5 del Apéndice C). En la solución, fundamentada en el concepto básico de equivalencia (véase subsección 4.1.2.4), se presentan los siguientes conceptos teóricos:

Dos fórmulas proposicionales, ψ y Φ, son equivalentes si $v(\psi) = v(\Phi)$ para toda interpretación v. Sobre la tabla de verdad, dos fórmulas equivalentes tienen exactamente los mismos valores de verdad sobre cada línea. Muy coloquialmente, son dos formas sintácticamente distintas de expresar lo mismo (puesto que semánticamente son indistinguibles). Estos conceptos son extensibles a la Lógica de Primer Orden de tal forma que dos fórmulas ψ y Φ serán equivalentes si $\psi \models \Phi$ y $\Phi \models \psi$. Existen distintos métodos para demostrar que dos fórmulas son equivalentes. En el caso de Lógica Proposicional bastaría con representar las tablas de verdad correspondientes a cada fórmula y comprobar que son idénticas. Cuando la cardinalidad no lo permita o en el caso de Lógica de Primer Orden, donde no se tiene el colchón de las tablas de verdad, un método eficaz es demostrar que es insatisfacible la fórmula $\neg(\psi \leftrightarrow \Phi)$. Otro método sería partiendo de cualquiera de las fórmulas equivalentes y mediante reemplazos de equivalencias básicas llegar a la otra fórmula que se supone equivalente.

Indicando para cada respuesta si es correcta o incorrecta, así como una respuesta razonada basada en la obtención de una cláusula vacía mediante resolución en caso de que ser equivalentes ó la obtención de una interpretación que satisfaga un grupo de fórmulas en el caso no ser equivalentes.

Para mayor claridad se muestra un ejemplo de solución para este tipo de preguntas:

(a) $p \vee q \rightarrow r \wedge s$. Incorrecta.
En este caso, tenemos las fórmulas $\psi = \{\neg\,(t \vee (\neg q \rightarrow p))\}$ y $\Phi = \{p \vee q \rightarrow r \wedge s\}$. Se podría demostrar que no son equivalentes si $\psi \models \Phi$ ó $\Phi \models \psi$ son satisfacibles.
Se puede comprobar que $\neg\psi \wedge \Phi$ es satisfacible para la interpretación $I(p, q, r, s, t) = (f, f, f, f, v)$.

(b) $\neg\,(\neg t \rightarrow (q \vee p))$. Correcta.
En este caso, tenemos las fórmulas $\psi = \{\neg\,(t \vee (\neg q \rightarrow p))\}$ y $\Phi = \{\neg\,(\neg t \rightarrow (q \vee p))\}$. Se podría demostrar demostrar que son equivalentes si $\psi \models \Phi$ y $\Phi \models \psi$. Por tanto si son equivalentes $\neg\psi \wedge \Phi$ y $\psi \wedge \neg\Phi$ serían insatisfacibles.
Se puede comprobar en la resolución expuesta, se llega a una claúsula vacía para $\neg\psi \wedge \Phi$.
En primer lugar se haya su forma clausulada:
$\neg\psi \wedge \Phi = \{\{t, q, p\}, \{\neg t\}, \{\neg q\}, \{\neg p\}\}$
Resolución:
1. $\{\{t, q, p\}, \{\neg t\}, \{\neg q\}, \{\neg p\}\}$ # Premisa
2. $\{t, q, p\}$ # Cláusula de la premisa 1
3. $\{\neg t\}$ # Cláusula de la premisa 1
4. $\{\neg q\}$ # Cláusula de la premisa 1
5. $\{\neg p\}$ # Cláusula de la premisa 1
6. [ ] # Cláusula vacía (2,3,4,5)
Se puede comprobar en la resolución expuesta, se llega a una claúsula vacía para $\psi \wedge \neg\Phi$.
En primer lugar se haya su forma clausulada:
$\psi \wedge \neg\Phi = \{\{\neg t\}, \{\neg q\}, \{\neg p\}, \{t, q, p\}\}$
Resolución:
1. $\{\{\neg t\}, \{\neg q\}, \{\neg p\}, \{t, q, p\}\}$ # Premisa
2. $\{t, q, p\}$ # Cláusula de la premisa 1
3. $\{\neg t\}$ # Cláusula de la premisa 1
4. $\{\neg q\}$ # Cláusula de la premisa 1
5. $\{\neg p\}$ # Cláusula de la premisa 1
6. [ ] # Cláusula vacía (2,3,4,5)

(c) $p \wedge q \rightarrow r \vee s$. Incorrecta.
En este caso, tenemos las fórmulas $\psi = \{\neg\,(t \vee (\neg q \rightarrow p))\}$ y $\Phi = \{p \wedge q \rightarrow r \vee s\}$. Se podría demostrar que no son equivalentes si $\psi \models \Phi$ ó $\Phi \models \psi$ son satisfacibles.
Se puede comprobar que $\neg\psi \wedge \Phi$ es satisfacible para la interpretación $I(p, q, r, s, t) = (f, f, f, f, v)$.

QPCLogicPropositionalNotEquivalent

Similar a la anterior, pero en este caso se pregunta que fórmula no es equivalente (véase pregunta 6 del Apéndice C).

QPCLogicPropositionalConsequence

Se presentan tres consecuencias y se pide que se señale la correcta (véase pregunta 7 del Apéndice C) . En la solución, fundamentada en el concepto básico de consecuencia (véase subsección 4.1.2.3), se presentan los siguientes conceptos teóricos:

Una fórmula ψ es consecuencia de un conjunto $\Phi = \{\varphi_1, ..., \varphi_n\}$ de fórmulas si toda interpretación que satisface a Φ también satisface a ψ.
La notación para representar que ψ es consecuencia lógica de $\Phi = \{\varphi_1, ..., \varphi_n\}$ se suele emplear la notación $\Phi \models \psi$, ó $\{\varphi_1, ..., \varphi_n\} \models \psi$.
De esta forma: $\varphi_1, ..., \varphi_n \models \psi$ si y sólo si $\varphi_1 \wedge ... \wedge \rightarrow \psi$ es tautología.
Por tanto, existe una interdependencia formal entre consecuencia e insatisfacibilidad:
1. $\varphi_1, ..., \varphi_n \models \psi$ y entonces el conjunto $\varphi_1, ..., \varphi_n \neg\psi$ es insatisfacible.
2. $\varphi_1, ..., \varphi_n \neg\psi$ es insatisfacible entonces $\varphi_1, ..., \varphi_n \models \psi$.
Para verificar una relación de consecuencia se podría realizar un tabla de verdad, siempre y cuando fueran fórmulas proposicionales y la cardinalidad lo permitiera, procediendo a evaluar que en todas (todas, todas) las líneas en que esas fórmulas coinciden en ser verdaderas, la última también lo es (y quizá en alguna más).Otro procedimiento extendible a la Lógica de Primer Orden sería demostrar que $\varphi_1 \wedge ... \wedge \neg\psi$ es insatisfacible, bien mediante un árbol semántico cerrado ó un método de resolución del cual se pudiera llegar a una cláusula vacía.

Indicando para cada respuesta si es correcta o incorrecta, así como una respuesta razonada basada en la obtención de una cláusula vacía mediante resolución de un grupo de fórmulas en caso de ser consecuencia ó la obtención de una interpretación que satisfaga un grupo de fórmulas en el caso no ser consecuencia.

Para mayor claridad se muestra un ejemplo de solución para este tipo de preguntas:

(a) $\{\neg X_1, X_2 \models X_3\}$. Incorrecta.
En este caso, se podría realizar la tabla de verdad de $\{\neg X_1, X_2, \neg X_3\}$ verificando, por ejemplo, que se puede obtener una interpretación verdadera para $I(p,q,r,s,t) = (f,v,f,f,f)$. Al ser satifacible no será una consecuencia válida.

(b) $\{\neg X_1, \neg X_2 \models X_3\}$. Incorrecta.
En este caso, se podría realizar la tabla de verdad de $\{\neg X_1, \neg X_2, \neg X_3\}$ verificando, por ejemplo, que se puede obtener una interpretación verdadera para $I(p,q,r,s,t) = (v,v,f,f,f)$. Al ser satifacible no será una consecuencia válida.

(c) $\{X_1, \neg X_2 \models X_3\}$. Correcta.
En este caso se puede comprobar que la resolución expuesta respecto a $\{X_1, \neg X_2, \neg X_3\}$, se llega a una claúsula vacía, por tanto es insatisfacible.
En primer lugar se hayan las fórmulas en su forma clausulada:
$X_1 = \{\{\neg t\}, \{\neg q\}, \{\neg p\}\}$
$\neg X_2 = \{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$
$\neg X_3 = \{\{p, q\}, \{\neg r, \neg s\}\}$
Resolución:
1. $\{\{\neg t\}, \{\neg q\}, \{\neg p\}\}$ # Premisa
2. $\{\{p\}, \{q\}, \{\neg r\}, \{\neg s\}\}$ # Premisa
3. $\{\{p, q\}, \{\neg r, \neg s\}\}$ # Premisa
4. $\{\neg p\}$ # Cláusula de la premisa 1
5. $\{p\}$ # Cláusula de la premisa 2
6. [ ] # Cláusula vacía (5,4)

QPCLogicPropositionalNotConsequence

Similar a la anterior, pero en este caso se pregunta que fórmula no es equivalente (véase pregunta 8 del Apéndice C).

6.3.5.2. Lógica de predicados de primer orden

Incluidas en el paquete **ComputationalLogic.FirstOrder.Questions** se han implementado las clases que corresponden a los distintos modelos de preguntas de lógica de predicados de primer Orden. Se han implementado ocho modelos similares a las realizadas en lógica proposicional, además se han realizado otros siete basados en los conceptos de interpretación (véase subsección 4.1.2.5):

Respecto a las clases **QPCLogicFirstOrderSatisfacible**, **QPCLogicFirstOrderInSatisfacible**, **QPCLogicFirstOrderTautology**, **QPCLogicFirstOrderNotTautology**, **QPCLogicFirstOrderEquivalent**, **QPCLogicFirstOrderNotEquivalent**, **QPCLogicFirstOrderConsequence** y **QPCLogicFirstOrderNotConsequence** (véase preguntas 9, 10, 11, 12 ,13, 14 ,15 y 16 del Apéndice C) tienen el mismo formato que el descrito para las preguntas de lógica proposicional.

En las soluciones de las preguntas de lógica de proposicional, no se han usado tablas de verdad, por ello se han podido reutilizar tantos los conceptos teóricos como los algoritmos de demostración de corrección o incorrección.

QPCLogicFirstOrderInterpretationInSatisfacible, a partir de un universo dado y un grupo de fórmulas, se pregunta que es la interpretación correcta (véase pregunta 17 Apéndice C), en ese modelo el grupo de fórmulas es insatisfacible. En la solución, fundamentada en el concepto básico de interpretación (véase subsección 4.1.2.5), se presentan conceptos los siguientes teóricos:

Una interpretación satisface a una fórmula $\psi_1 \wedge ... \wedge \psi_n$, si satisface a cada una de las fórmulas por separado.
Una interpretación hace insatisfacible a una fórmula $\psi_1 \wedge ... \wedge \psi_n$, si existe al menos una fórmula insatisfacible.
Para determinar la insatisfacibilidad de una fórmula para una interpretación dada, se puede utilizar una tabla semántica y si ésta es cerrada implica que es insatisfacible, otro método sería mediante resolución llegar a una cadena vacía.
Para determinar la satisfacibilidad de ψ_i para una interpretación dada se puede deducir si $\neg\psi_i$ es insatisfacible.
Un posible algoritmo para obtener interpretaciones que hacen satisfacibles ψ_i, sería previamente descartar que no es insatisfacible, si es así, analizar los valores del dominio que hacen verdad a ψ_i.

Indicando para cada respuesta si es correcta o incorrecta, así como una respuesta razonada basada en la obtención de una cláusula vacía mediante resolución de un grupo de fórmulas en caso de ser insatisfacible.

Para mayor claridad se muestra un ejemplo de solución para este tipo de preguntas :

(a) $S = \emptyset, Q = \emptyset$. Incorrecta.
$\{Y_1 \wedge Y_2 \wedge \neg Y_3\}$No puede ser satisfacible, ya que es insatisfacible.

(b) $S = \{(0,0),(0,1)\}, Q = \emptyset$. Incorrecta.
$\{Y_1 \wedge Y_2 \wedge \neg Y_3\}$No puede ser satisfacible, ya que es insatisfacible.

(c) es insatisfacible. Correcta.
En este caso se puede comprobar en la resolución expuesta respecto a $\{Y_1 \wedge Y_2 \wedge \neg Y_3\}$, se llega a una claúsula vacía.
En primer lugar se hayan las fórmulas en su forma clausulada:
$Y_1 = \{\{Sax\}\}$
$Y_2 = \{\{\neg Qxy, Qxf(x,y)\}, \{\neg Qxy, Qf(x,y)y\}\}$
$\neg Y_3 = \{\{\neg Qbx, \neg Qxc\}, \{Qbc\}\}$
Resolución:
1. $\{\{Sax\}\}$ # Premisa
2. $\{\{\neg Qxy, Qxf(x,y)\}, \{\neg Qxy, Qf(x,y)y\}\}$ # Premisa
3. $\{\{\neg Qbx, \neg Qxc\}, \{Qbc\}\}$ # Premisa
4. $\{\neg Qxy, Qxf(x,y)\}$ # Cláusula de la premisa 2
5. $\{\neg Qxy, Qf(x,y)y\}$ # Cláusula de la premisa 2
6. $\{\neg Qbx, \neg Qxc\}$ # Cláusula de la premisa 3
7. $\{Qbc\}$ # Cláusula de la premisa 3
8. $\{\neg Qf(b,x)c, \neg Qbx\}$ # (6,4)
9. [ ] # Cláusula vacía (8,5,7)

QPCLogicFirstOrderInterpretationSatisfacible, similar a la anterior en su formato (véase pregunta 18 del Apéndice C), la diferencia es que el grupo de fórmulas es satisfacible. En la solución, fundamentada en el concepto básico de interpretación (véase subsección 4.1.2.5), se presentan conceptos teóricos, indicando para cada respuesta si es correcta o incorrecta. Las respuestas planteadas son la interpretación para el grupo de fórmulas (correcta), interpretación para la negación del grupo de fórmulas (incorrecta) y su insatisfacibilidad (incorrecta), por tanto en la solución se indica por descarte cuál es la correcta, es decir si es satisfacible no puede ser insatisfacible, además si es satisfacible para una interpretación, la interpretación no puede ser satisfacible para la negación del grupo de fórmulas.

QPCLogicFirstOrderInterpretationNoSatisfacibleFormula, se pregunta que interpretación no satisface a una fórmula (véase pregunta 19 del Apéndice C). En la solución, fundamentada en el concepto básico interpretación (véase subsección 4.1.2.5), se presentan conceptos los mismos fundamentos teóricos que el modelo *QPCLogicFirstOrderInterpretationInSatisfacible*.

Indicando para cada respuesta si es correcta o incorrecta. Las respuestas planteadas son tres fórmulas, dos son satisfacibles respecto a la interpretación dada en el enunciado, mientras que

otra no puede ser satisfacible para dicha interpretación por ser la resultante de negar una de las fórmulas anteriores, por tanto en la solución se indica por descarte cuál es la correcta, es decir si tiene una fórmula tiene una interpretación válida, la misma interpretación no puede ser valida para su fórmula negada.

Para mayor claridad se muestra un ejemplo de solución para este tipo de preguntas:

(a) Y_3. Incorrecta.
En este caso, se podría verificar, por ejemplo, que la interpretación I^Y: Dominio $\{0, 1\}$, con $Q = \emptyset$ satisface Y_3.

(b) $\neg Y_2$. Correcta.
No puede ser una interpretación verdadera para $\neg Y_2$, ya que la interpretación del enunciado hace verdadera a Y_2.

(c) Y_2. Incorrecta.
En este caso, se podría verificar, por ejemplo, que la interpretación I^Y: Dominio $\{0, 1\}$, con $Q = \emptyset$ satisface Y_2.

QPCLogicFirstOrderInterpretationSatisfacibleFormula, similar a la anterior, pero en este caso se pregunta que interpretación satisface a una fórmula (véase pregunta 20 del Apéndice C). En la solución, fundamentada en el concepto básico de interpretación (véase subsección 4.1.2.5), se presentan los mismos conceptos teóricos que el modelo *QPCLogicFirstOrderInterpretationInSatisfacible*.

Indicando para cada respuesta si es correcta o incorrecta. Las respuestas planteadas son tres fórmulas, una es satisfacible respecto a la interpretación dada en el enunciado, mientras las otras dos no puede ser satisfacibles para dicha interpretación por ser la resultante de negar una de la fórmula anteriores, por tanto en la solución se indica por descarte cuál es la correcta, es decir si tiene una fórmula tiene una interpretación válida, la misma interpretación no puede ser valida para su fórmula negada.

QPCLogicFirstOrderInterpretationSatisfacibleFormulaOnlyOne, se pregunta que interpretación satisface a una fórmula ó dos fórmulas (véase pregunta 21 del Apéndice C). En la solución, fundamentada en el concepto básico de interpretación (véase subsección 4.1.2.5), se presentan los mismos conceptos teóricos que el modelo *QPCLogicFirstOrderInterpretationInSatisfacible*.

Indicando para cada respuesta si es correcta o incorrecta. Las respuestas planteadas son una fórmula, dos formulas ó ninguna de las fórmulas, una es satisfacible respecto a la interpretación dada en el enunciado (correcta), mientras que otra no puede ser satisfacible para dicha interpretación por ser la resultante de negar la fórmula anterior, por tanto en la solución se indica por descarte cuál es la correcta, es decir si tiene una fórmula tiene una interpretación válida, se puede observar que las otras no son validas para su fórmula negada.

Para mayor claridad se muestra un ejemplo de solución para este tipo de preguntas :

(a) Solo Y_2. Correcta.
Efectivamente, la interpretación propuesta hace satisfacible a la fórmula Y_2, pero no a $\neg Y_1$, ya que satisface a su negada Y_1.

(b) a Y_2 y $\neg Y_1$. Incorrecta.
La interpretación propuesta hace satisfacible sólo a la fórmula Y_2, pero no a $\neg Y_1$, ya que satisface a su negada Y_1.

(c) ni a Y_2 ni a $\neg Y_1$. Incorrecta.
La interpretación propuesta hace satisfacible al menos a la fórmula Y_2, aunque no a $\neg Y_1$, ya que satisface a su negada Y_1.

QPCLogicFirstOrderInterpretationSatisfacibleFormulaBoth, similar a la anterior en su planteamiento (véase pregunta 22 del Apéndice C), pero en este caso la respuesta correcta sería la que presenta ambas fórmulas.

QPCLogicFirstOrderInterpretationSatisfacibleFormulaNeither, similar a la anterior en su planteamiento (véase pregunta 23 del Apéndice C), pero en este caso la respuesta correcta sería aquella hace referencia a ninguna de las fórmulas.

6.3.5.3. Diagrama de clases modelos de preguntas

Se expone el diagrama de clases que representa cada uno de los modelos de preguntas capaz de gestionar la librería. En total serán veintitrés modelos (ocho respecto a lógica proposicional y quince respecto a lógica de predicados de primer orden).

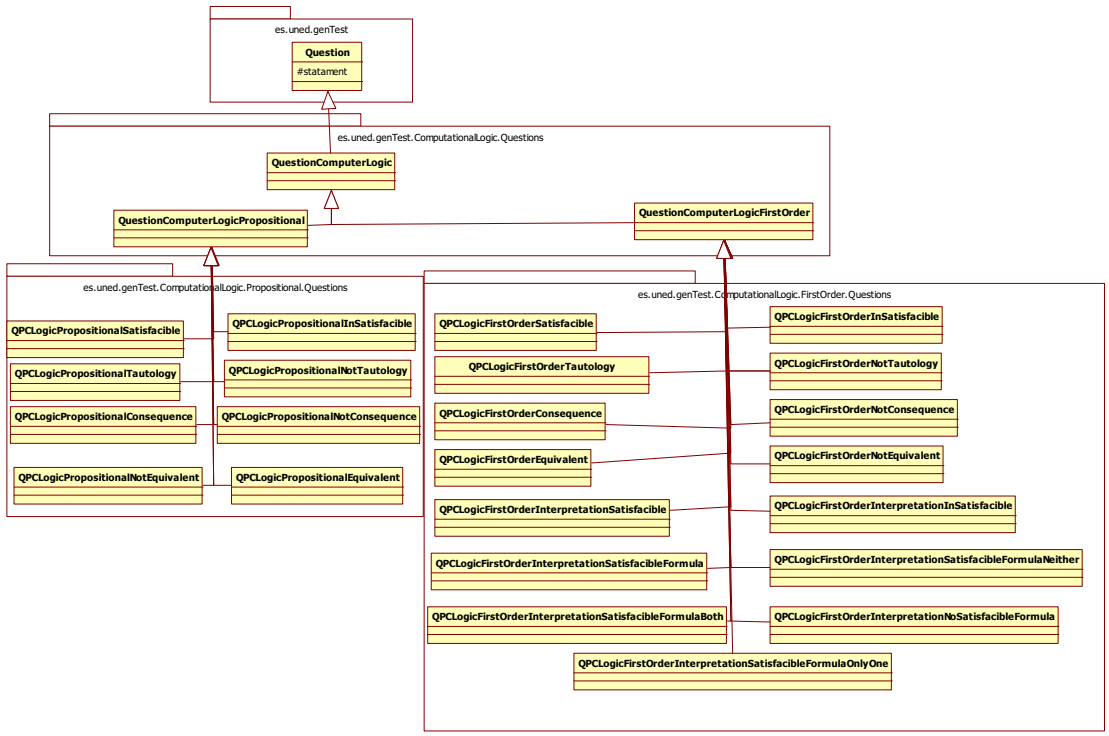


Figura 6.5: Diseño. Diagrama de clases de modelos de preguntas.

6.3.5.4. Diagrama de clases soluciones

Se expone el diagrama de clases que representa las soluciones a cada uno de los modelos de preguntas.

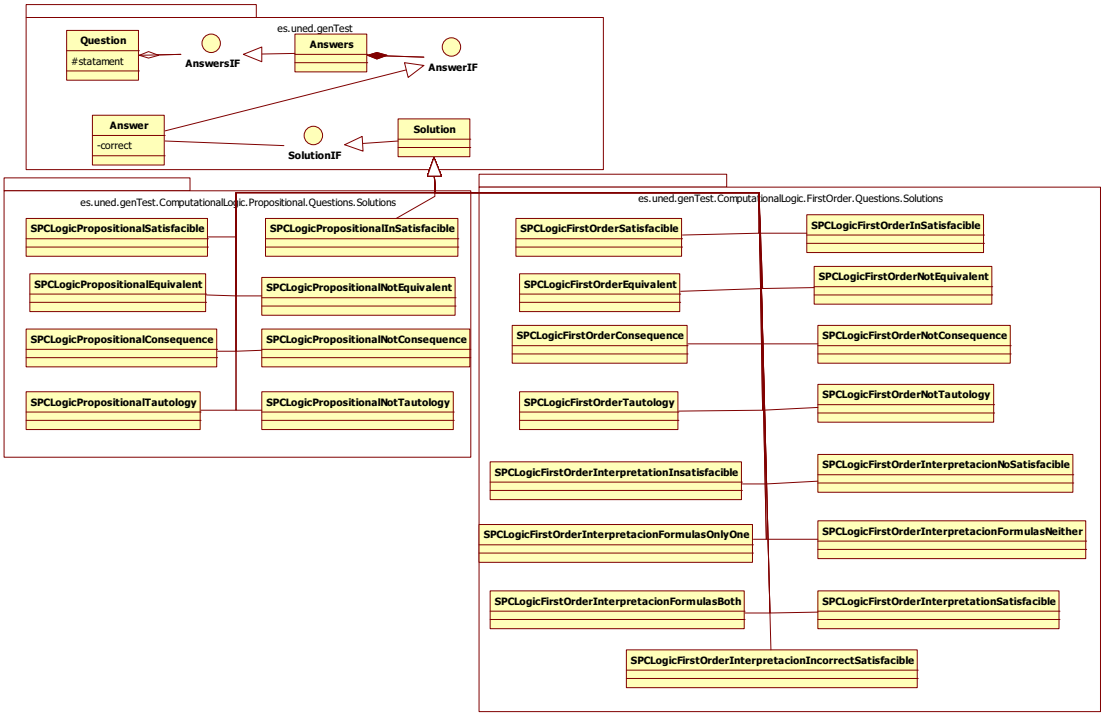


Figura 6.6: Diseño. Diagrama de clases soluciones.

6.3.5.5. Diagrama de clases conceptos teóricos y tipos de respuestas

Se expone un diagrama de clases que representa los distintos modelos de conceptos teóricos, así como los distintos formatos de respuestas.

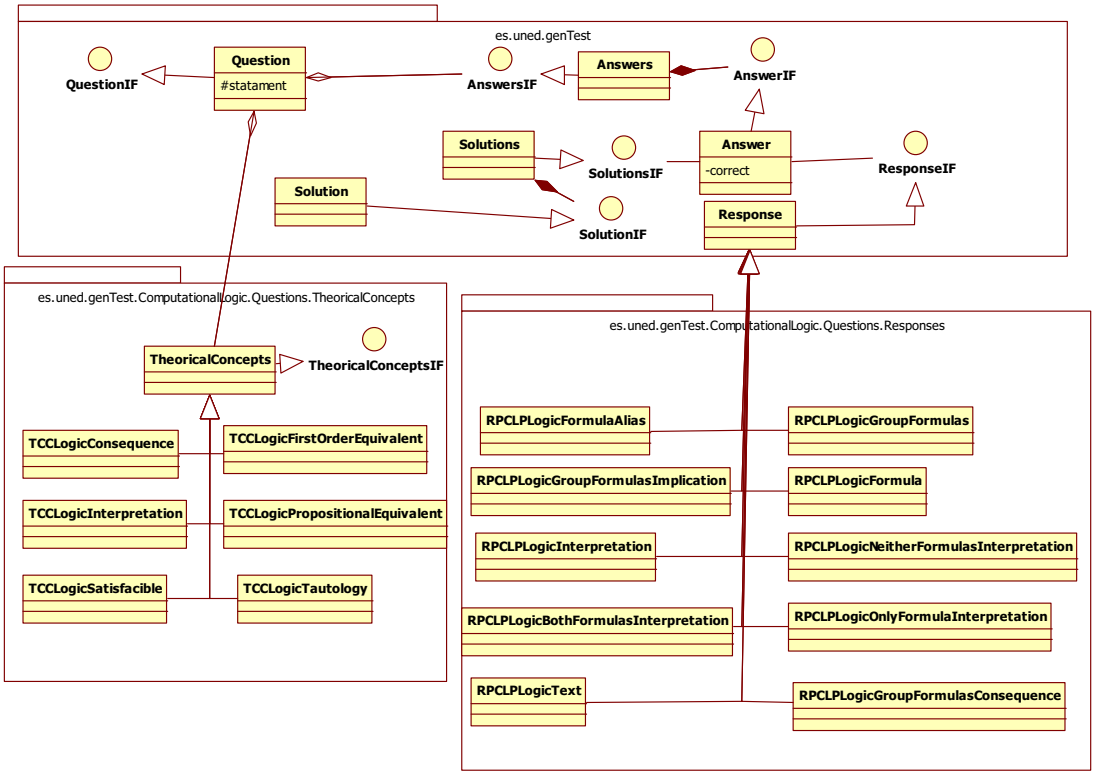


Figura 6.7: Diseño. Diagrama de clases conceptos teóricos y respuestas.

6.3.5.6. Descripción del paquete de ocultación de complejidad

Basado en el patrón de diseño *Facade* se ha diseñado una paquete denominado **view**, que contiene clases con la finalidad ocultar la complejidad de la definición de las instancias de los modelos de preguntas por parte del profesor desde el entorno de evaluación de conocimientos (SIETTE).

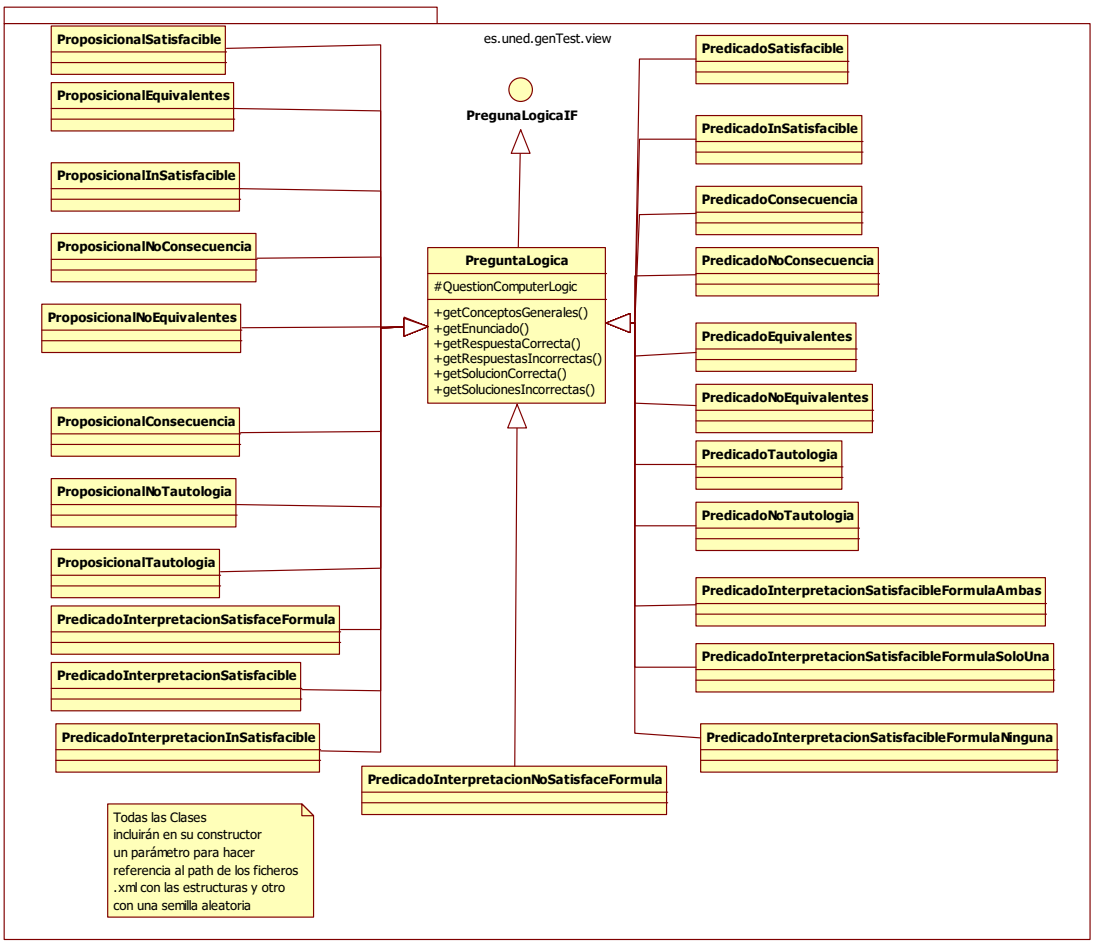


Figura 6.8: Diseño. Diagrama de clases view

6.4. Diseño de la realización de los casos de uso

A continuación se muestran dos diagramas de secuencia correspondientes a los casos de uso estudiados en el análisis que desde el punto de vista de diseño tienen especial interés por ser el núcleo de la aplicación: almacenar estructuras XML y la generación de preguntas.

6.4.1. Diagrama de secuencia de generación de preguntas

A continuación se muestra un diagrama de secuencias (véase figura 6.9) que representa la generación de preguntas, incluidas las soluciones y conceptos teóricos.

6.4.2. Diagrama de secuencia de almacenamiento de estructuras

Se muestra un diagrama de secuencias (véase figura 6.10) que representa la generación de estructuras (almacenadas en un ficheros XML) que se serán utilizados desde SIETTE en la creación de las preguntas en tiempo predecible, menor de un segundo y sin necesidad de tener instalado Prover9 ni Mace4.

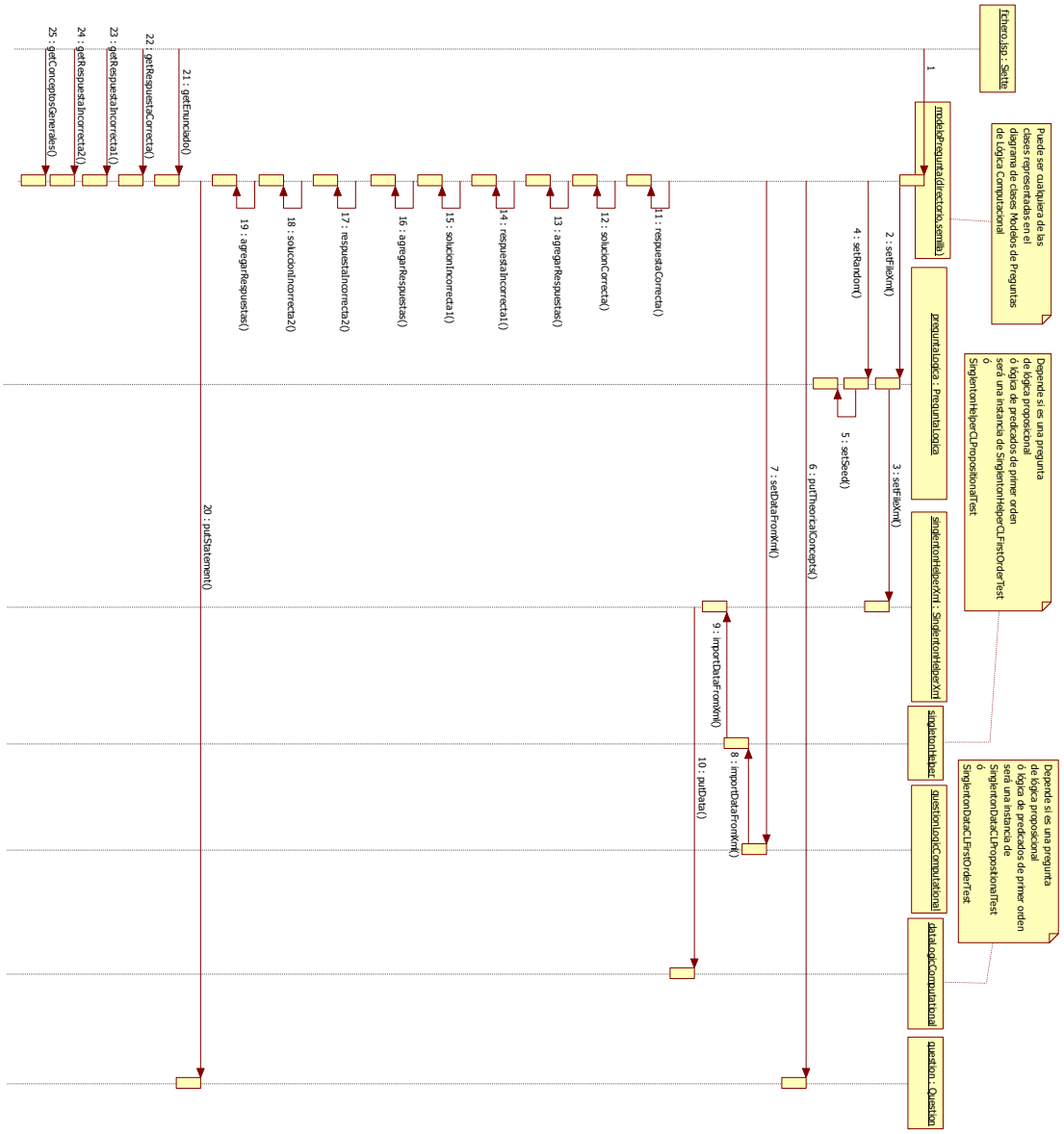


Figura 6.9: Diseño. Diagrama de secuencia generación de preguntas.

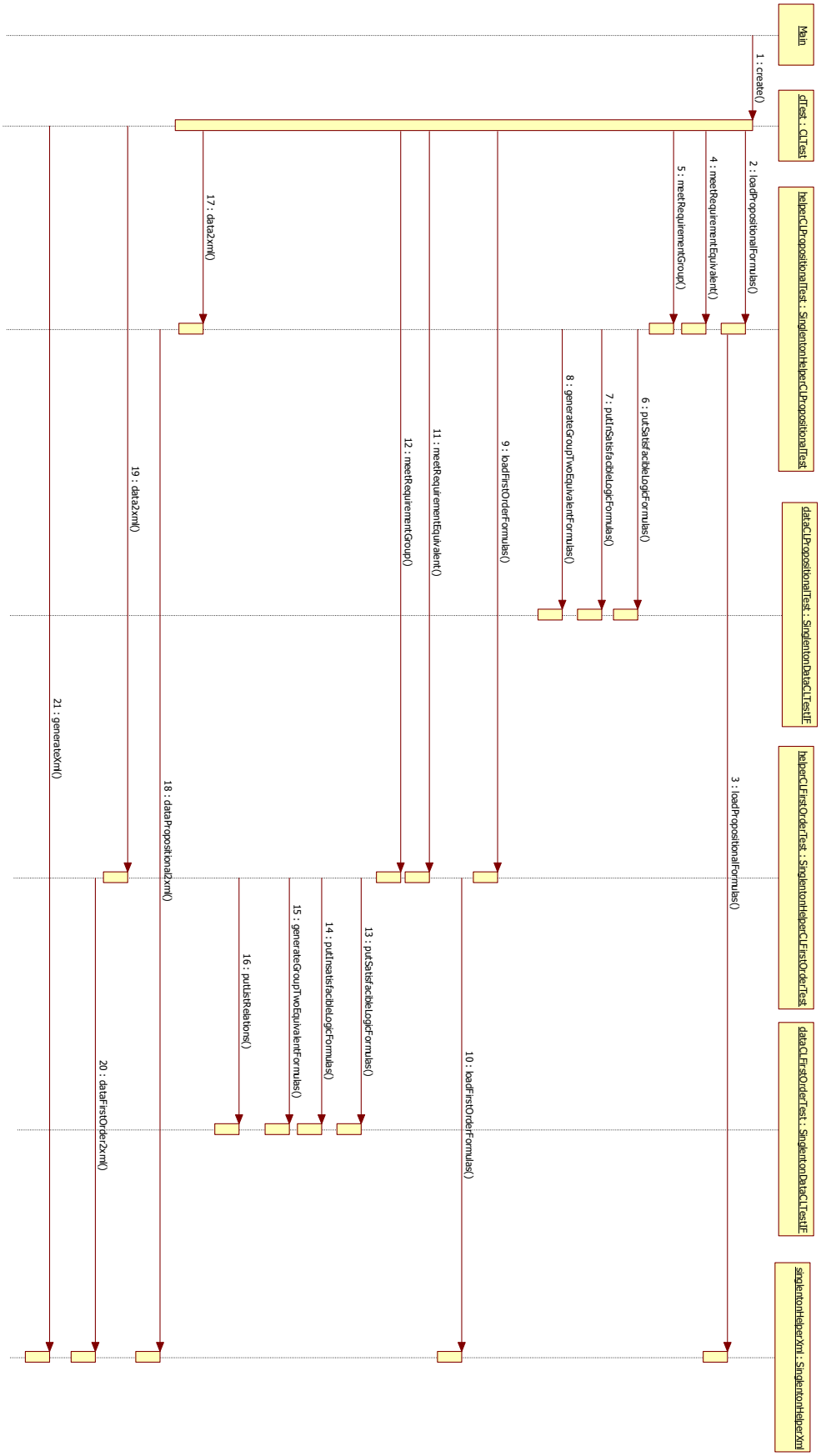


Figura 6.10: Diseño. Diagrama de secuencia almacenamiento estructuras XML.

6.5. Modelo de datos

La estructura de datos será gestionada por dos clases:

- **singletonDataCLPropositionalTest**.
- **singletonDataCLFirstOrderTest**.

Estas clases contendrán los atributos necesarios para generar los distintos modelos de preguntas respecto a lógica proposicional y lógica de predicados de primer orden.

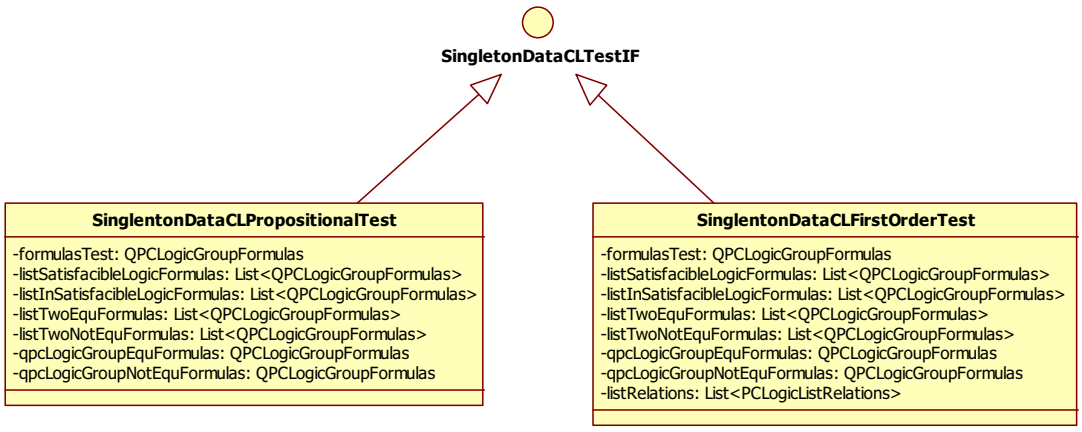


Figura 6.11: Diseño. Diagrama de clases modelo de datos.

La función de cada uno de los atributos, tanto en lógica proposicional, como en lógica de predicados de primer orden es la siguiente:

formulasTest. Almacenará tres ó cuatro fórmulas que reúnan las siguientes condiciones

- La primera de las fórmulas tendrá al menos dos fórmulas equivalentes.
- Tendrán al menos tres grupos de tres fórmulas insatisfacibles, incluyendo las fórmulas negadas.
- Tendrán al menos tres grupos de tres fórmulas satisfacibles, incluyendo las fórmulas negadas.

listSatisfacibleLogicFormulas. Almacenará grupos satisfacibles de tres fórmulas, obtenidas a partir de formulasTest, incluyendo sus fórmulas negadas, así como una interpretación satisfacible para cada uno de los grupos.

listInSatisfacibleLogicFormulas. Almacenará grupos insatisfacibles de tres fórmulas, obtenidas de formulasTest, incluyendo sus fórmulas negadas, así como una prueba de insatisfacibilidad para cada uno de los grupos.

qpcLogicGroupEquFormulas. Almacenará fórmulas equivalentes a la primera fórmula de formulasTest.

qpcLogicGroupNotEquFormulas. Almacenará fórmulas no equivalentes a la primera fórmula de formulasTest.

listTwoEquFormulas. Almacenará por cada pareja de fórmulas equivalentes una prueba de su equivalencia, es decir si tenemos dos fórmulas ψ y Φ equivalentes entre sí, se almacenará una prueba para $\psi \models \Phi$ y otra para $\Phi \models \psi$. ψ siempre será la primera fórmula de formulasTest.

listTwoNotEquFormulas. Almacenará por cada pareja de fórmulas no equivalentes una prueba de su no equivalencia, es decir si tenemos dos fórmulas ψ y Φ no equivalentes entre sí se almacenará una prueba para $\psi \models \Phi$ y otra para $\Phi \models \psi$, que demuestre su no equivalencia. ψ siempre será la primera fórmula de formulasTest.

Además para lógica de predicados de primer orden también se almacenarán en **listRelations** todas las interpretaciones posibles tanto para las fórmulas de formulaTest, como para sus negadas.

La información almacenada en las clases **singletonDataCLPropositionalTest** y **singletonDataCLFirstOrderTest**, se les dará persistencia almacenando sus atributos en un fichero XML, de forma que sea fácilmente exportable al entorno de evaluación de conocimientos (SIETTE), teniendo preprocesados y disponible los datos necesarios de un test, a partir del cuál poder generar gran cantidad de preguntas basadas en los modelos estudiados anteriormente (véase subsección 6.3.5), sin necesidad de tener instalado Prover 9 ni Mace 4 en dicho entorno, además asegurando tiempos predecibles y menores de un segundo en la generación de las preguntas.

Es importante destacar que por cada test se tendrá su propio fichero XML, podría existir otra posibilidad, un sólo fichero XML, con todos los datos preprocesados de todos los exámenes tipo test, pero se ha comprobado que desde un punto de vista de rendimiento esta opción es peor, ya que los tiempos de apertura de un fichero XML más grande son muchos mayores. Por otro lado, si se utiliza un fichero XML por cada test disponible es más modular, pudiendo el usuario fácilmente agregar o eliminar nuevos test al entorno de evaluación.

En el proceso de generación de una pregunta se importará a la clases descritas anteriormente (singleton**DataCLPropositionalTest** y singleton**DataCLFirstOrderTest**) desde un fichero XML elegido de forma aleatoria las estructuras anteriormente estudiadas, a continuación se seleccionará, también aleatoriamente, un conjunto de datos con los que se formularán la pregunta. Para mayor claridad veamos un ejemplo, supongamos que se quiere generar una pregunta del tipo:

El conjunto satisfacible es:

1. $\{X_1, X_2, \neg X_3\}$
2. $\{X_1, \neg X_2, X_3\}$
3. $\{X_1, X_2, X_3\}$

En este caso se cargaría un fichero XML de forma aleatoria, para posteriormente elegir también aleatoriamente un grupo de fórmulas satisfacibles almacenada en la estructura listSatisfacibleLogicFormulas (opción correcta) y otros dos grupos insatisfacibles de la estructura listInSatisfacible-

LogicFormulas (opciones incorrectas). Con estos datos se construirían la pregunta, respuestas y soluciones.

Puede consultar la estructura del fichero XML en formato DTD o XML Schema en el DVD adjunto a esta memoria o en el repositorio web del proyecto [Campillo Molina, 2012] , contenidos en el fichero examXml.rar.

6.6. Diagrama de despliegue

En la siguiente figura, se deja patente como se integrarán los distintos componentes respecto a la solución elegida.

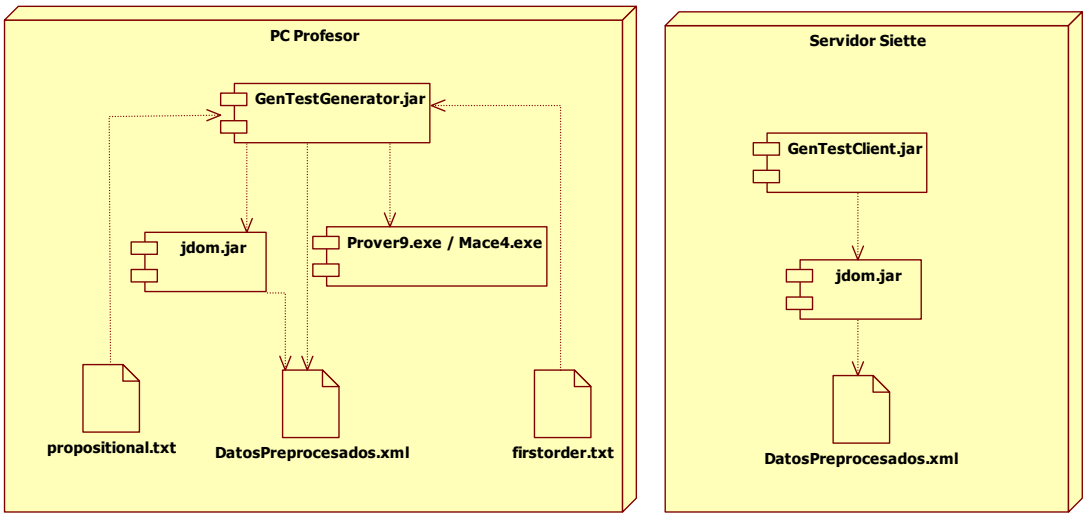


Figura 6.12: Estudio de Viabilidad. Diagrama de despliegue.

7. Construcción

Se han construido dos subsistemas:

- Módulo generador.
- Módulo evaluador.

7.1. Módulo generador

Se trata de un programa basado en líneas de comandos que se ejecutará en el ordenador del usuario (véase Apéndice B Manual de usuario) y tendrá dos cometidos:

- Como finalidad principal generar datos preprocesados en formato XML que serán desplegados en el entorno de evaluación SIETTE.
- Otra finalidad secundaria será la generación de exámenes tipo test en formato LaTeX.

Este módulo admite los modificadores *[-t [-f:file.xml]][-g[:n]][-p][-h[elp]]*, donde:

- **-t**. Genera un examen en formato LaTeX, depositándolo en el directorio .\LaTexTests.
- **-t -f:file.xml**. Genera un examen en formato LaTeX, tomando como referencia el fichero XML pasado como parámetro. El examen generado se deposita en el directorio .\LaTexTests. Por ejemplo *GenTest -t -f:ExamLogic0000.xml*.
- **-g**. Genera un fichero XML con datos preprocesados, que será desplegado por el usuario en el entorno de evaluación de conocimientos (SIETTE). El fichero generado se deposita en el directorio .\LogicTestsXML.
- **-g:n**. Genera n ficheros XML con datos preprocesados, que serán desplegados por el usuario en el entorno de evaluación de conocimientos (SIETTE). Los ficheros generados se depositan en el directorio .\LogicTestsXML. Por ejemplo de uso: *GenTest -g:3*, en este caso se generan tres ficheros XML con datos preprocesados.
- **-p**. Comprueba que todos los componentes requeridos están instalados correctamente, estos son:
  - Máquina Virtual de Java 1.5 ó superior [MaquinaVirtualJava, 2012].

- Prover9 y Mace4 [Prover9Mace4, 2009].

Para más información sobre la instalación de estos componentes véase Apéndice A Manual de instalación.

- **-h**. Muestra la ayuda.

7.2. Módulo evaluador

Se trata de una librería java que se desplegará en el entorno de evaluación SIETTE (véase Apéndice A Manual de instalación).

7.2.1. Particularidades de implementación en SIETTE

Como se ya se ha mencionado anteriormente el entorno de evaluación web será SIETTE, por ello a la hora de construir la librería que posteriormente será desplegada en dicho entorno, es importante saber que debe **cumplir** ciertas **características**, cuyo **conocimiento serían de utilidad a futuros proyectos** de esta índole y son fruto de conclusiones obtenidas a lo largo de análisis y diseño.

- Según indicaciones de los administradores de SIETTE, las librerías implementadas en Java, deben ser compiladas utilizando la versión 1.5. Por este motivo se ha creado una tarea Ant (se ha incluido el fichero 20111224AntGenTestClientUNED.rar, en el DVD adjunto a esta memoria o en el repositorio web del proyecto [Campillo Molina, 2012]). Destacar que el fichero build.xml, para conseguir dicho objetivo, en la tarea de compilación, se ha configurado la instrucción mostrada a continuación.

```
<javac srcdir="${fuente}" destdir="${destino}" compiler="javac1.5" source="1.5" target="1.5">
```

- Cada una de las preguntas se considerarán ítems generativos basados en Java, en realidad todos los componentes del ítem se combinará internamente e una única página JSP, que será instanciado a la hora de ejecutarse, por tanto se le dará un tratamiento similar al de una página JSP, por ello es importante, antes de desplegarlo en un servidor SIETTE, implementar una **página JSP**, que utilice todos los ítems posibles, de la forma más parecida al que se utilizaría en el entorno de evaluación. Esta acción tendrá doble finalidad, por un lado certifica que con casi toda probabilidad funcionará correctamente después del despliegue, además le servirá de guía a los administradores de SIETTE para posibles configuraciones y resolución de dudas futuras sobre el formato final. Como ejemplo se ha realizado una página JSP de ejemplo, con las características mencionadas (véase subsección 8.2.2.4).

- Para la gestión de **números aleatorios** SIETTE suministra la clase siette.util.Random, en realidad es una extensión de la clase java.util.Random (se ha incluido el fichero siette.util.jar, suministrada por el equipo de administración de SIETTE, en el DVD adjunto a esta memoria y en el repositorio web del proyecto [Campillo Molina, 2012]) . Esta clase se incluye por defecto, y no es necesaria configurarla en la sentencias import,s. No obstante la librería implementada en este proyecto necesita internamente generar nuevos números aleatorios, por ello la solución final adoptada, para compatibilizarla con la de SIETTE ha sido la siguiente:

  - En la definición del ítem se establece una semilla de número aleatorio que se pasa como parámetro en su constructor, un ejemplo podría ser (fíjese exclusivamente en la invocación al método **Random.nextLong()**) :

    ```
    PreguntaLogica preguntaLogica =
    new ProposicionalSatisfacible("/ac/LogicExams/*",Random.nextLong());
    ```

  - En el constructor de la pregunta se generará la semilla en función del parámetro (en el ejemplo random) actualizado en el punto anterior, para ello utilizaremos el siguiente código Java:

    ```
    (new siette.util.Random()).setSeed(random);
    ```

  - A partir de este momento siempre que se necesite en la librería un número aleatorio bastará con generarlo utilizando la clase siette.util.Random, por ejemplo:

    ```
    siette.util.Random.nextInt();
    ```

- El acceso a los ficheros preprocesados en formato XML, de donde se actualizan las estructuras necesarias para generar las preguntas, podría realizarse de forma interna en la librería implementada en este proyecto, pero el equipo de administración de SIETTE, propuso que sería mejor que la ubicación de los ficheros fuera un parámetro, dando mayor flexibilidad en el caso de cambio de ubicación, por ello la solución final adoptada ha sido la siguiente:

  - En primer lugar se creará un directorio, llamado ac/LogicExams, desde las opciones de administración de la asignatura Lógica Computacional (previamente dada de alta en SIETTE), más concretamente en el panel de *Gestión de Ficheros*, tal y como se muestra en la figura adjunta. A este directorio se transferirán todos los ficheros preprocesados que el equipo docente estime oportuno:

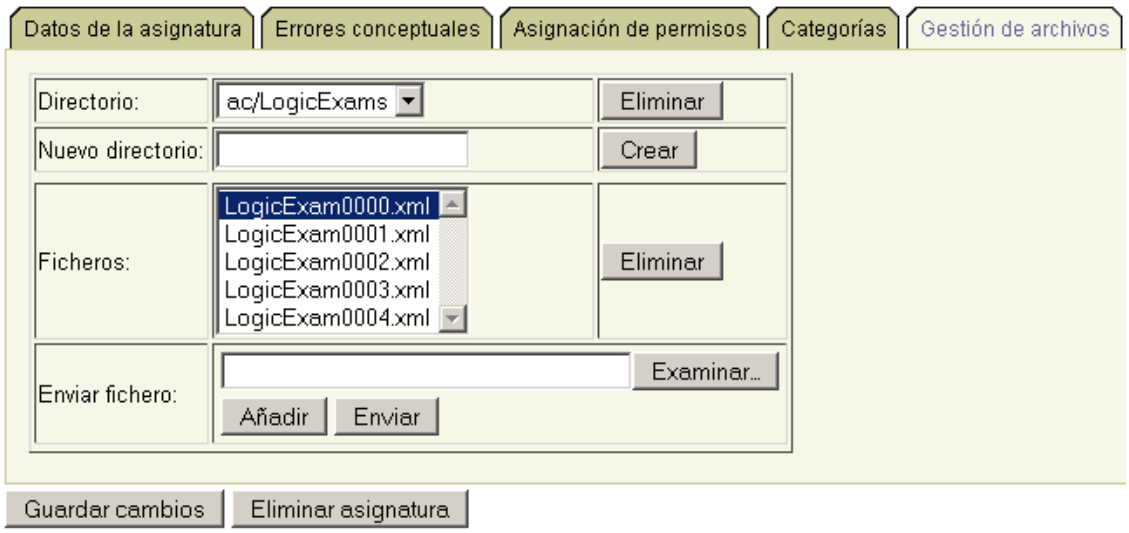


Figura 7.1: Particularidades SIETTE. Creación de directorio en una asignatura.

- En la definición del ítem se establece el path relativo donde son transferidos los ficheros preprocesados, que corresponde al directorio creado en el paso anterior. Para mayor flexibilidad podrá hacer referencia a todos los ficheros preprocesados, en este caso la librería elegirá de forma aleatoria un fichero (fíjese exclusivamente en el parámetro "**/ac/LogicExams/***") o se le puede indicar un fichero en concreto (fíjese exclusivamente en el parámetro "**/ac/LogicExams/LogicExam0000.xml**"):

```
PreguntaLogica preguntaLogica =
new ProposicionalSatisfacible( "/ac/LogicExams/*",Random.nextLong());
ó
PreguntaLogica preguntaLogica =
new ProposicionalSatisfacible(
"/ac/LogicExams/LogicExam0000.xml",Random.nextLong());
```

- Para localizar el directorio desde la librería se realizará referencia de forma absoluta. Se utilizará la constante SIETTE.PATH_ASIGNATURAS (incluida en la clase siette.SIETTE) que devuelve el path absoluto del contexto auxiliar. Esta variable toma un valor parecido a: "/usr/local/tomcat/webapps/siette.htdocs/", a lo que se añadirá el directorio "ac/LogicExams" creado anteriormente en la asignatura y descrito en pasos anteriores. Como información adicional, aunque no trascendente, la constante SIETTE.PATH_ASIGNATURAS se instancia al hacer la instalación de SIETTE a través de ABS_URL_APP_DEPLOY_DIR en web.xml. De esta forma el código final utilizado en el constructor de la clase que instancia la pregunta, donde locationFilesXml es el parámetro actualizado con la cadena “ac/LogicExams”, tendrá el siguiente formato:

```
siette.SIETTE.PATH_ASIGNATURAS+locationFilesXml
```

8. Resultados y pruebas

En este capítulo se describen los datos estadísticos que demuestran de forma cuantificable el cumplimiento de lo objetivos y requisitos de la librería implementada.

8.1. Cantidad de preguntas

En general la cantidad de preguntas que será capaz de generar el sistema en principio es ilimitada, no obstante se puede estimar por cada fichero XML preprocesado cuantas preguntas se pueden obtener, para una información más detallada puede consultar los resultados expuestos en la tabla cantidad de preguntas (véase Apéndice D.2).

8.1.1. Conjunto satisfacible de fórmulas

En las preguntas de tipo **"***El conjunto satisfacible es:...***"** (véase preguntas 1 y 9 del Apéndice C), el número de preguntas dependerá de cuantos grupos de tres fórmulas son satisfacibles y cuantos insatisfacibles, si tomamos como referencia tres fórmulas (cada una puede tener dos estados negada o no negada), todos los grupos de tres fórmulas capaces de ser generados en los que las fórmulas no se repitan son ocho (2^3). Lo habitual es que existan más grupos satisfacibles que insatisfacibles por ello vamos a tomar como referencia para los cálculos siguientes que tenemos cinco grupos de fórmulas satisfacibles y tres insatisfacibles.

En este tipo de preguntas se elige un grupo satisfacible y dos insatisfacibles, por tanto la cantidad de preguntas posibles sería el resultado de multiplicar el número de grupos satisfacibles (desde ahora *gs*) por los grupos insatisfacibles (desde ahora *gi*) tomados de dos en dos (combinaciones sin repetición, C^2_{gi}), por tanto $CantidadPreguntas = gs{*}C^2_{gi}$, $CantidadPreguntas = 5{*}C^2_3$, $CantidadPreguntas = 15$, esta cantidad es una estimación y puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.2. Conjunto insatisfacible de fórmulas

En las preguntas de tipo "*El conjunto insatisfacible es:...*" (véase preguntas 2 y 10 del Apéndice C), se puede hacer un razonamiento similar al anteriormente descrito para las preguntas del tipo "*El conjunto satisfacible es:...*", sólo que ahora en cada pregunta se elige un grupo insatisfacible y dos satisfacibles, por tanto $CantidadPreguntas = gi{*}C^2_{gs}$, $CantidadPreguntas = 3{*}C^2_5$,

$CantidadPreguntas = 3{*}10$, $CantidadPreguntas = 30$, esta cantidad puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.3. Conjunto de fórmulas que conforman tautología

Si estudiamos las preguntas de tipo "*Señale la tautología:...*" (véase preguntas 3 y 11 del Apéndice C), por cada pregunta se toma como referencia un grupo insatisfacible y dos satisfacibles, por tanto el razonamiento es similar que para las del tipo "*El conjunto insatisfacible es:...*": $CantidadPreguntas = gi{*}C_{gs}^{2}$, $CantidadPreguntas = 3{*}C_{5}^{2}$, $CantidadPreguntas = 3{*}10$, $CantidadPreguntas = 30$, esta cantidad puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.4. Conjunto de fórmulas que no conforman tautología

Si estudiamos las preguntas de tipo "*Señale cuál no es tautología:...*" (véase preguntas 4 y 12 del Apéndice C), por cada pregunta se toma como referencia un grupo satisfacible y dos insatisfacibles, por tanto el razonamiento es similar que para las del tipo "*El conjunto satisfacible es:...*": $CantidadPreguntas = gs{*}C_{gi}^{2}$, $CantidadPreguntas = 5{*}C_{3}^{2}$, $CantidadPreguntas = 5{*}3$, $CantidadPreguntas = 15$, esta cantidad puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.5. Consecuencia correcta

Si estudiamos las preguntas de tipo "*La consecuencia correcta es:...*" (véase preguntas 5 y 13 del Apéndice C), por cada pregunta se toma como referencia un grupo insatisfacible y dos satisfacibles, por tanto el razonamiento es similar que para las del tipo "*El conjunto insatisfacible es:...*" : $CantidadPreguntas = gi{*}C_{gs}^{2}$, $CantidadPreguntas = 3{*}C_{5}^{2}$, $CantidadPreguntas = 3{*}10$, $CantidadPreguntas = 30$, esta cantidad puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.6. Consecuencia no correcta

Si estudiamos las preguntas de tipo "*La consecuencia no correcta es*:..." (véase preguntas 6 y 14 del Apéndice C), por cada pregunta se toma como referencia un grupo satisfacible y dos insatisfacibles, por tanto el razonamiento es similar que para las del tipo "*El conjunto satisfacible es:...*": $CantidadPreguntas = gs{*}C_{gi}^{2}$, $CantidadPreguntas = 5{*}C_{3}^{2}$, $CantidadPreguntas = 5{*}3$, $CantidadPreguntas = 15$, esta cantidad puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.7. Fórmulas equivalentes

En este caso, las preguntas de tipo "*Indique la fórmula equivalente a ...*" (véase preguntas 7 y 15 del Apéndice C), se toma como referencia una fórmula equivalente a la primera fórmula del enunciado y otras dos no equivalentes a ésta. Al obtener los datos preprocesados, como mínimo

siempre habrá dos fórmulas equivalentes (desde ahora *fe*) y normalmente bastantes más fórmulas no equivalentes (desde ahora *fne*), supongamos cinco. Por tanto, $CantidadPreguntas = fe{*}C^2_{fne}$, $CantidadPreguntas = 2{*}C^2_5$, $CantidadPreguntas = 2{*}10$, $CantidadPreguntas = 20$.

8.1.8. Fórmulas no equivalentes

En este caso, las preguntas de tipo "*Indique la fórmula no equivalente a ...*" (véase preguntas 8 y 16 del Apéndice C), se toma como referencia una fórmula no equivalente a la primera fórmula del enunciado y otras dos equivalentes a ésta. Siguiendo el razonamiento del modelo "*Indique la fórmula equivalente a ...*": $CantidadPreguntas = fne{*}C^2_{fe}$, $CantidadPreguntas = 5{*}C^2_2$, $CantidadPreguntas = 5{*}1$, $CantidadPreguntas = 5$.

8.1.9. Interpretación satisfacible a un conjunto de fórmulas

Con este enunciado tenemos dos modelos de preguntas uno en la que la respuesta puede ser alguna interpretación y otro en el que el conjunto de formulas es insatisfacible.

8.1.9.1. Respuesta alguna interpretación

Las preguntas del tipo "*Sobre el Universo U = {0,1}. ¿Qué interpretación satisface...*", con respuesta alguna interpretación (véase pregunta 17 del Apéndice C), se basa en un grupo de fórmulas satisfacible, por tanto $CantidadPreguntas = gs$, $CantidadPreguntas = 5$, esta cantidad puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.9.2. Respuesta insatisfacible

Las preguntas del tipo "*Sobre el Universo U = {0,1}. ¿Qué interpretación satisface...*", con respuesta alguna interpretación (véase pregunta 18 del Apéndice C), se basa en un grupo de fórmulas insatisfacible, por tanto $CantidadPreguntas = gi$, $CantidadPreguntas = 3$, esta cantidad puede variar en función del número de grupos satisfacibles e insatisfacibles.

8.1.10. Una interpretación satisface a una ó dos fórmulas

Con el título *"La interpretación I_n satisface..."* tenemos cuatro modelos posibles, en el que la respuesta correcta puede algunas de las siguientes: una fórmula , "*Sólo una* f*órmul*a", "*a* fórmula1 *y* fórmula 2", "*ni a* fórmula 1 *ni a* fórmula 2". En estos tipos de preguntas es difícil de predecir cuantas preguntas distintas se pueden generar, ya que dependen en gran medida de las interpretaciones capaces de satisfacer a todas las fórmulas, por ello vamos a hacer una serie de suposiciones, pero insistiendo que debería estudiarse cada test de forma individual para ser más preciso.

8.1.10.1. Satisface a una fórmula

Las preguntas del tipo "*L*a interpretación I_n satisface...", con respuesta una fórmula (véase pregunta 20 del Apéndice C), se basa en encontrar una interpretación que satisfaga a dos fórmulas incluyendo en los cálculos sus formas negadas (desde ahora *nf*e) , si suponemos que existen

cuatro interpretaciones (desde ahora *ni*) capaces de satisfacer todas las posibles a las fórmulas del enunciado, si se elige una fórmula (respuesta correcta) que satisface a una interpretación propuesta en el enunciado , debemos descartar su forma negada (será un respuesta incorrecta), por tanto nos quedan otras cuatro candidatas a ser respuesta incorrecta, de esta forma:

$CantidadPreguntas = ni*((nfe*2) - 2)$, $CantidadPreguntas = 4*((3*2) - 2)$,

$CantidadPreguntas = 16$, esta cantidad podrá variar dependiendo de la cantidad de interpretaciones obtenidas.

8.1.10.2. Satisface sólo a una fórmula

Las preguntas del tipo "*L*a interpretación I_n satisface...", con respuesta "*Sólo una f*órmula" (véase pregunta 21 del Apéndice C), si suponemos que existen cuatro interpretaciones capaces de satisfacer todas las posibles a las fórmulas del enunciado y si se eligen dos fórmulas que no satisfacen a la interpretación propuesta en el enunciado (respuesta incorrecta), nos quedan otras cuatro candidatas a ser respuesta correcta, de esta forma $CantidadPreguntas = ni*((nfe*2) - 2)$, $CantidadPreguntas = 4*((3*2) - 2)$, $CantidadPreguntas = 16$, esta cantidad podrá variar dependiendo de la cantidad de interpretaciones obtenidas.

8.1.10.3. Satisface a dos fórmulas

En las preguntas del tipo "*L*a interpretación I_n satisface...", con respuesta "*a* fórmula1 *y* fórmula 2" (véase preguntas 22 del Apéndice C). Suponemos que existen cuatro interpretaciones capaces de satisfacer todas las posibles a las fórmulas del enunciado, si se eligen dos fórmulas que satisfacen a la interpretación propuesta en el enunciado (respuesta correcta), en este caso tendríamos combinaciones de seis elementos (todas las fórmulas del enunciado incluidas sus formas negadas) agrupadas de dos en dos (recordemos que en este caso la respuesta correcta incluye dos fórmulas), pero para ser más conservadores consideramos que sólo un cuarto de estos grupos satisfacen de forma simultánea a la interpretación propuesta en el enunciado, por tanto $CantidadPreguntas = ni*(C_6^2/4)$, $CantidadPreguntas = 4*(15/4)$, $CantidadPreguntas = 12$, esta cantidad podrá variar dependiendo de la cantidad de interpretaciones obtenidas.

8.1.10.4. No satisface a ninguna fórmula

En las preguntas del tipo "*L*a interpretación I_n satisface...", con respuesta "*ni a* fórmula1 *ni a* fórmula 2" (véase preguntas 23 del Apéndice C). Suponemos que existen cuatro interpretaciones capaces de satisfacer todas las posibles a las fórmulas del enunciado, si se eligen dos fórmulas que no satisfacen a la interpretación propuesta en el enunciado (respuesta correcta), en esta caso tendríamos combinaciones de seis elementos (todas las fórmulas del enunciado incluidas sus formas negadas) agrupadas de dos en dos (recordemos que en este caso la respuesta correcta incluye dos fórmulas), pero para ser más conservadores consideramos que sólo un cuarto de estos grupos satisfacen de forma simultánea a la interpretación propuesta en el enunciado, por tanto $CantidadPreguntas = ni*(C_6^2/4)$, $CantidadPreguntas = 4*(15/4)$,$CantidadPreguntas = 12$, esta cantidad podrá variar dependiendo de la cantidad de interpretaciones obtenidas.

8.1.11. Una interpretación no satisface a una fórmula

Las preguntas del tipo "*L*a interpretación I_n no satisface...", (véase preguntas 19 del Apéndice C), tiene un razonamiento similar a 8.1.10.1, se basa en encontrar una interpretación que no satisfaga a una fórmula (se tendrá en cuenta también la formas negadas) , si suponemos que existen cuatro interpretaciones capaces de satisfacer todas a todas fórmulas del enunciado, si se elige una fórmula que la satisface a interpretación propuesta en el enunciado (respuesta incorrecta), siendo su forma negada la respuesta correcta, por tanto nos quedan otras cuatro fórmulas candidatas a ser respuesta incorrecta, de esta forma $CantidadPreguntas = ni*((nfe*2) - 4)$, $CantidadPreguntas = 4*((3*2) - 2)$,$CantidadPreguntas = 16$, esta cantidad podrá variar dependiendo de la cantidad de interpretaciones obtenidas.

8.2. Pruebas realizadas

Se han realizado las pruebas oportunas para confirmar el correcto funcionamiento de acuerdo a los objetivos y requisitos.

Destacar dichas pruebas se han realizado en un ordenador personal con las siguientes características:

- Sistema Operativo Windows Server 2008 Standard 64 bits.
- Procesador Intel Core 2 Duo CPU E7400 2,80 GHz.
- Memoria RAM de 4 GB.
- Disco duro con tecnología Ultra ATA.

Para cada una de las pruebas detalladas, se han implementado clases de java, auxiliadas por la librería JUnit 4. Se puede acceder a todos los ficheros, tanto las clases java implementadas, como los ficheros con los resultados obtenidos, en el directorio *Pruebas* del DVD adjunto a esta memoria o en el repositorio web del proyecto [Campillo Molina, 2012], donde se ha depositado el fichero GenTestUNEDPruebas.rar.

8.2.1. Pruebas unitarias

Se han realizado las siguientes pruebas unitarias, con las que se demuestra el correcto comportamiento de los distintos módulos de forma independiente:

8.2.1.1. Ejemplo de uso de fundamentos de lógica computacional

Como recopilación de los conceptos de fundamentos de lógica computacional y las clases construidas al respecto se expone un ejemplo sencillo de como poder utilizar las clases *CLogicGroupFormulas* y *CLogicFormula*, puede consultar el código fuente del fichero EjemploUsoGrupoFormulas.java en el directorio ".\Pruebas\8.2.1 Pruebas Unitarias\8.2.1.1 Ejemplo Fundamentos de

lógica computacional" (DVD o GenTestUNEDPruebas.rar del repositorio), en este código se pretende demostrar que la fórmula $(p \wedge q) \wedge \neg(q \vee r)$ es insatisfacible (si se representase su tabla de verdad podríamos comprobar que todos los valores son falsos).

Se puede observar que la fórmula $(p \wedge q) \wedge \neg(q \vee r)$, se divide en dos subfórmulas $X_1 = p \wedge q$ y $X_2 = q \vee r$, siendo negada ésta última con el objetivo de probar el método *denyFormula*, quedando $\neg X_2 = \neg(q \vee r)$. Posteriormente se agregan a un grupo de fórmulas, sobre la que se realiza la pregunta si es insatisfacible mediante el método *isInsatisfacible*. Una vez ejecutado, tal y como se esperaba devuelve la cadena *"El grupo de fórmulas es insatisfacible"*.

8.2.1.2. Subsistema Generador. Fundamentos de lógica computacional

Mediante las clases java (ProofLogicFormula y ProofLogicGroupFormulas, auxiliadas de JUnit), implementada para este fin, se han realizado pruebas del correcto funcionamiento de los métodos que implementan los conceptos semánticos básicos de lógica computacional, tales como satisfacibilidad, insatisfacibilidad, tautología y equivalencia, obteniéndose resultados satisfactorios, como puede verse en las siguientes figuras:

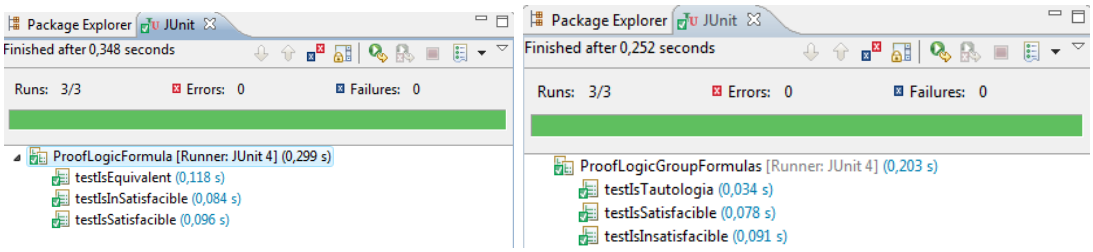


Figura 8.1: Pruebas Unitarias. Correcto funcionamiento conceptos semánticos.

Puede encontrar las clases ProofLogicFormula y ProofLogicGroupFormulas en el directorio ".\Pruebas\8.2.1 Pruebas Unitarias\8.2.1.2 Fundamentos de lógica computacional" (DVD o GenTestUNEDPruebas.rar del repositorio).

8.2.1.3. Subsistema Generador. Clases de cada modelo de pregunta

Mediante la clase java (ProofQuestionsLatex.java, auxiliada de JUnit), implementada para este fin, se han generado, en formato LaTeX, cada uno de los veintitrés modelos de preguntas, incluyendo enunciado, respuestas, conceptos generales y soluciones.

Para este fin en se han realizado tres iteraciones, e importando (aleatoriamente) la información necesaria de ficheros XML de datos preprocesados, obteniéndose como resultado tres ficheros textos (uno por cada iteración).

Los tiempos de cada iteración, expresados en milisegundos, han sido 169, 132 y 72.

Puede encontrar los ficheros con las veintitrés preguntas generadas en formato LaTeX, así como la clase ProofQuestionsLatex en el directorio ".\Pruebas\8.2.1 Pruebas Unitarias\8.2.1.3 Generador.Clases de cada modelo de pregunta" (DVD o GenTestUNEDPruebas.rar del repositorio).

8.2.1.4. Subsistema Evaluador. Clases de cada modelo de pregunta

Mediante la clase java (ProofQuestionsStatistics.java, auxiliada de JUnit), implementada para este fin, se han calculado la media, desviación típica, valor máximo y valor mínimo de los tiempos de generación de cada uno de los veintitrés modelos de preguntas, incluyendo enunciado, respuestas, conceptos generales y soluciones, en un intento de simular el comportamiento que tendría SIETTE. Para cada modelo se han generado 1.000 preguntas (en total 23.000). Para mayor fiabilidad de los datos estadísticos expuestos el mismo proceso se ha llevado a cabo tres veces consecutivas (generándose 69.000 preguntas en total), obteniéndose los datos que se exponen en el Apéndice D.1.

Como se puede observar en ningún caso se sobrepasa el requisito impuesto respecto al tiempo (ninguna pregunta puede superar el tiempo un segundo en su generación), en el peor de los casos ninguna ha superado 80 milisegundos , con una media total de 10 milisegundos.

Puede encontrar los ficheros con las 69.000 preguntas generadas, los ficheros en formato CSV (formato exportable a hojas de cálculo) con los datos estadísticos, así como la clase ProofQuestionsStatistics en el directorio ".\Pruebas\8.2.1 Pruebas Unitarias\8.2.1.4 Evaluador.Clases de cada modelo de pregunta" (DVD o GenTestUNEDPruebas.rar del repositorio).

8.2.2. Pruebas de integración

Con estas pruebas se demuestra el correcto funcionamiento después de integrar los distintos módulos:

8.2.2.1. Subsistema Generador. Generación de exámenes en formato LaTeX

Mediante la clase java (ProofTestsLatex, auxiliada por JUnit), implementada para este fin, se han generado tres exámenes, en formato LaTeX, de forma consecutiva, cada uno contiene los veintitrés modelos posibles de preguntas, incluyendo enunciado, respuestas, conceptos generales, soluciones y como datos comunes a las cuestiones cuatro fórmulas de cada tipo (proposicionales, predicado de primer orden).

Para este fin en se han realizado tres iteraciones, sin utilizar ningún fichero XML de datos preprocesados, obteniéndose como resultado tres ficheros en formato LaTeX, los cuáles han sido satisfactoriamente compilados y exportados a PDF.

Los tiempos de cada iteración, expresados en segundos, han sido 40, 42 y 44. Tras numerosas pruebas se ha constatado que en este caso los tiempos no son predecibles y dependen en gran medida de las fórmulas incluidas en los ficheros *propositional.txt* y *firstOrder.txt*.

Puede encontrar los ficheros con los exámenes tipo test generados en formato LaTeX, su exportación a PDF, así como la clase ProofTestsLatex en el directorio ".\Pruebas\8.2.2 Pruebas Integracion\8.2.2.1 Generacion de Test en Formato LaTeX" (DVD o GenTestUNEDPruebas.rar del repositorio).

8.2.2.2. Subsistema Generador. Generación ficheros preprocesados XML

Mediante la clase java ProofTestsXML (auxiliada por JUnit), implementada para este fin, se han generado tres ficheros de datos preprocesados XML, de forma consecutiva, listos para desplegar en SIETTE.

Para este fin en se han realizado tres iteraciones, obteniéndose como resultado tres ficheros en formato XML, los cuáles han sido satisfactoriamente probados mediante las opciones del programa generador *-t -f:file.xml.*

Los tiempos de cada iteración, expresados en segundos, han sido 23, 22 y 65. Tras numerosas pruebas se ha constatado que en este caso los tiempos no son predecibles y dependen en gran medida de las fórmulas incluidas en los ficheros *propositional.txt* y *firstOrder.txt.*

Puede encontrar los ficheros con los exámenes tipo test generados generados en formato XML, así como la clase ProofTestsXML en el directorio ".\Pruebas\8.2.2 Pruebas Integracion\8.2.2.2 Generacion ficheros preprocesados XML"(DVD o GenTestUNEDPruebas.rar del repositorio).

8.2.2.3. Subsistema Evaluador. Generación de exámenes en formato LaTeX

Mediante las clases java ProofTest y ProofsTestLatex (auxiliada por JUnit), implementadas para este fin, utilizando del módulo evaluador, es decir, el desplegado en SIETTE se ha generado un fichero en formato LaTeX a partir de un fichero de datos preprocesados en formato XML, con el objetivo de comprobar el correcto funcionamiento de todos los modelos de preguntas.

Puede encontrar los ficheros con los exámenes tipo test generados en formato LaTeX (proofXMLLaTex.tex), junto su exportación a formato PDF (proofXMLLaTex.pdf), el fichero de datos con datos preprocesados XML (LogicExam0000.xml), así como las clases utilizadas (ProofsTestLatex.java y ProofTest.java) en el directorio ".\Pruebas\8.2.2 Pruebas Integracion\8.2.2.3 Evaluador.Generacion de Test en formato LaTeX" (DVD o GenTestUNEDPruebas.rar del repositorio).

8.2.2.4. Subsistema Evaluador. Página JSP

Como ya se indicó anteriormente (véase subsección 7.2.1) cada una de las preguntas se considerarán ítems generativos basados en Java, por este motivo es conveniente antes de su despliegue en el servidor SIETTE comprobar que la librería se comporta de la forma esperada. Para reproducir de la forma más fiel posible el comportamiento que tendrá en SIETTE se debe crear un página JSP en la que se usen todas las clases y métodos accesibles por el usuario, tal y como lo haría SIETTE al utilizar los ítems generativos. Puede consultar el fichero de ejemplo testPreguntasIndependientes.jsp en el directorio ".Pruebas\8.2.2 Pruebas Integracion\8.2.2.4 Pagina JSP" (DVD o GenTestUNEDPruebas.rar del repositorio).

Para probar el correcto funcionamiento testPreguntasIndependientes.jsp se han realizado las siguientes configuraciones:

- Se ha instalado un servidor Apache/Tomcat.
- Se ha configurado la librería MathJax.
- En el constructor de las preguntas se ha modificado la línea de código

```
siette.SIETTE.PATH_ASIGNATURAS + locationFilesXml
```

por

```
locationFilesXml
```

(se debe recordar restaurar el primer código antes de desplegar en SIETTE).

- Implementar una aplicación Web que cumpla con los siguientes requisitos:
  - Tener creado el directorio exams en WEB-INF, donde se copiarán los datos preprocesados XML.
  - Tener creado el directorio lib en WEB-INF, donde se copiarán la librería implementada GenTestUNED.jar, jdom.jar y siette.util.jar (lo puede encontrar en el DVD adjunto a esta memoria o en el repositorio web del proyecto [Campillo Molina, 2012]).
  - Haber copiado el fichero testPreguntasIndependientes.jsp al directorio WEB-INF.

8.2.3. Pruebas de sistema

Con estas pruebas se demostrará que se cumplen los requisitos y funcionalidades propuestas.

8.2.3.1. Subsistema Generador.

1. **Parámetro -t**. Se ha generado de forma correcta un test aleatorio en formato LaTex con datos comunes (incluye cuatro fórmulas proposicionales y cuatro fórmulas de predicados de primer orden), así como todas sus interpretaciones posibles. Puede consultarse tanto el fichero LaTex generado, como su formato en PDF en el directorio ".\Pruebas\8.2.3 Prueba Sistema\8.2.3.1 Subsistema Generador\1. -t" (DVD o GenTestUNEDPruebas.rar del repositorio).

2. **Parámetro -t -f:ExamenLogica.xml**. Se ha generado de forma correcta un test, a partir de un fichero .XML con datos preprocesados, en formato LaTeX con datos comunes (incluye tres fórmulas proposicionales y tres fórmulas de predicados de primer orden, así como todas sus interpretaciones posibles. Puede consultarse tanto el fichero LaTeX generado, como su formato en PDF en el directorio ".\Pruebas\8.2.3 Prueba Sistema\8.2.3.1 Subsistema Generador\2. -t -f ExamenLogica.xml" (DVD o GenTestUNEDPruebas.rar del repositorio).

3. **Parámetro -g**. Se ha generado de forma correcta un fichero con datos preprocesados. Puede consultarse el fichero XML generado en el directorio ".\Pruebas\8.2.3 Prueba Sistema\8.2.3.1 Subsistema Generador\3. -g" (DVD o GenTestUNEDPruebas.rar del repositorio).

4. **Parámetro -g:3**. Se han generado de forma correcta y consecutiva tres ficheros XML con datos preprocesados. Puede consultarse los ficheros XML generados en el directorio ".\Pruebas\8.2.3 Prueba Sistema\8.2.3.1 Subsistema Generador\3. -g 3" (DVD o GenTestUNEDPruebas.rar del repositorio).

5. **Parámetro -h**. Muestra textos de ayuda.

6. **Ninguno de los anteriores**. Distintos mensajes de error y textos de ayuda.

8.2.3.2. Subsistema Evaluador

Según se detalla en el manual de usuario (véase Apéndice B.2), se han configurado en SIETTE **una asignatura** (Álgebra Computacional), así como los **veintitrés modelos de preguntas**, para después configurar tres modelos de exámenes tipo test: Lógica Proposicional, Lógica de Predicados de Primer Orden y Álgebra Computacional.

8.3. Estudio de tiempos y resultados

Del estudio y pruebas realizadas en este capítulo se desprenden los siguientes resultados:

- La **cantidad de preguntas** capaz de generar el sistema, en principio **es ilimitada**, mientras más fórmulas sean utilizadas mayor cantidad de exámenes tipo test distintos se podrán generar. Se ha realizado una estimación de la cantidad de cuestiones que se pueden obtener por cada tipo de pregunta en función de un fichero XML con datos preprocesados. Cabe destacar que se pueden desplegar, sin límite, ficheros preprocesados en SIETTE.
- Los **tiempos de generación de preguntas** por parte del módulo evaluador a partir de datos preprocesados XML presuntamente será como máximo de 80 milisegundos/pregunta, con una media de **10 milisegundos/pregunta**, siendo **predecibles**, cumpliendo el requisito de 1 segundo/pregunta como máximo.
- Los **tiempos de generación de los ficheros XML con datos preprocesados** (posteriormente exportados a SIETTE), por parte del módulo generador se realiza en un tiempo medio de **40 segundos/ficheroXML** (aproximadamente), además **no** es **predecible**, dependiendo en gran medida de las fórmulas utilizadas. Esto **no** es **preocupante** debido a que es un **proceso fuera de línea**.
- Los **tiempos de generación de exámenes en formato LaTeX** por parte del módulo generador se realiza en un tiempo medio de **45 segundos/ficheroXML** (aproximadamente), además **no** es **predecible**, dependiendo en gran medida de las fórmulas utilizadas. Esto **no** es **preocupante** debido a que es un **proceso fuera de línea**.
- En general, se han implementado clases diseñadas, auxiliadas por JUnit, con la finalidad de probar cada una de las partes (pruebas unitarias), la integración de éstas (pruebas de integración) y el funcionamiento correcto tanto del módulo generador, como del módulo evaluador (pruebas de sistema), siendo en todos los casos satisfactorias.

9. Conclusiones y líneas futuras

9.1. Conclusiones respecto al proyecto

En resumen, el auge de la docencia a través de Internet implica proporcionar a los equipos docentes de herramientas, cada vez más demandadas, capaces de motivar e incrementar la interactividad por parte de los alumnos, por tanto considero que el presente proyecto es una interesante aportación en este campo.

La generación automática de preguntas respecto a los temas seleccionados (lógica proposicional y lógica de predicados de primer orden) han planteado desafíos importantes, tales como la notación matemática, la búsqueda, uso y adaptación de programas y librerías que pudieran reutilizarse dentro del campo del álgebra, la generación de las preguntas, así como las soluciones y explicaciones que sirvieran de fuente de estudio a los alumnos, las restricciones de tiempo independientemente de la complejidad de las fórmulas y por último su instalación en un entorno de evaluación de conocimientos en producción (en particular SIETTE) en la que el profesor pudiera de forma sencilla y flexible gestionar nuevos exámenes tipo test. La solución satisfactoria a todas estas cuestiones y como se han implementado hace pensar que el presente proyecto pudiera ser una buena referencia para la generación de preguntas de forma automática en otras áreas del saber.

También se han proporcionado dos patrones, preguntas tipo test en general y conceptos básicos de lógica proposicional y lógica de predicados de primer orden, de forma que puedan ser reutilizadas en futuros proyectos relacionados con estos u otros temas.

Desde el comienzo del proyecto se estimó que los productos finales deberían ser instalados en sistemas en producción y no quedarse en un plano teórico, por ello durante la realización se ha utilizado gran cantidad de los conocimientos, habilidades y técnicas estudiadas a lo largo de la carrera, de esta forma, se han seguido paradigmas, metodologías, ciclos de vida y utilizado herramientas, consiguiendo desde un enfoque ingenieril los objetivos propuestos. En este tema, cabe mención especial las herramientas utilizadas, siendo todas de software libre, lo que da valor añadido al proyecto, pudiendo servir como guía a otros ingenieros a la hora de elegir herramientas que le ayuden en sus labores diarias.

El entorno de evaluación de conocimientos seleccionado, SIETTE, ha cumplido con todas las expectativas, siendo sorprendente el enfoque que le da a los ítems generativos, capaz de utilizar librerías implementadas en Java, ocultando en gran medida la complejidad al profesor, aunque el programador de dichas librerías debe poner especial cuidado en simplificar al máximo el uso de

éstas. Destacar que durante el despliegue y el uso extensivo de SIETTE se detectó una incidencia al mostrar las ayudas de las preguntas, no era capaz de traducirlas a notación matemática, una vez comunicado a los administradores fue subsanado en el plazo de dos semanas, gracias a lo cuál estará disponible para futuros exámenes tipo test.

En general, pienso que ha sido un proyecto muy completo y bien enfocado, en el que se han utilizado de forma directa o indirecta gran cantidad de lo conocimientos y habilidades aprendidos durante la carrera, así como otros nuevos que sin duda me serán de gran utilidad en el futuro profesional como Ingeniero en Informática.

9.2. Conocimientos adquiridos

Son muchos los conocimientos adquiridos y otros tantos que han facilitado el afianzamiento de los adquiridos durante la carrera; cabe destacar los siguientes:

- He adquirido conocimientos respecto a todos los temas y herramientas relacionados con el formato LaTeX, así como su aplicación en notación matemática tanto en formato PDF como en Internet.
- He afianzado los conocimientos de álgebra computacional, aprendidos durante la carrera, así como su aplicación en el uso de herramientas relacionadas con este campo, como por ejemplo Prove9 y Mace4.
- A pesar de que antes de comenzar el proyecto, tenía un buen nivel de conocimientos relacionados con el paradigma Orientado a Objetos y el lenguaje de programación Java, he afianzado conceptos respecto a estos temas, más concretamente respecto a patrones y su aplicación práctica.
- He utilizado por primera vez la librería de distribución gratuita JDOM, para la gestión de ficheros XML mediante Java.
- Me ha parecido de gran utilidad el repositorio Bitbucket, como herramientas de control de versiones.
- He aprendido a gestionar exámenes tipo test el entorno de evaluación de conocimientos SIETTE, así como conceptos relacionados con ítems generativos.
- Desde un punto de vista de la gestión de proyectos, he aprendido a utilizar herramientas que hasta la fecha sólo sabía de su existencia, como por ejemplo StarUML, poniendo en práctica los algoritmos propuestos por COCOMO II, para la estimación de costes y esfuerzo, que a su vez me han servido para realizar un Diagrama de Gantt, capaz de distribuir cronológicamente las tareas a realizar por un equipo capaz de analizar, diseñar, construir e implementar el producto final.
- La propia generación de documentación me ha llevado a utilizar herramientas hasta ahora desconocidas tales como Lyx y JabRef.

- El trabajo colaborativo, en los que han participado al tutor del proyecto y los administradores-desarrolladores de SIETTE, ha sido muy adecuada, constatando que la separación geográfica actualmente no es un inconveniente para el desarrollo de proyectos, independientemente de su complejidad, además de no causar ningún desembolso económico. Las herramientas utilizadas para este fin han sido correo electrónico, Skype, Dropbox y Bitbucket.

9.3. Mejoras

Como en cualquier otro proyecto informático, a pesar, de haber conseguido un producto final, capaz de ser utilizado por los usuarios finales, es decir, no se ha quedado en un plano teórico, siempre se puede mejorar. Las mejoras más importantes que se podrían abordar en un futuro inmediato son:

- Sería conveniente la construcción de una interfaz gráfica que complementase a los comandos del módulo generador. En el producto entregado éste módulo realiza todas las funcionalidades mediante línea de comandos, pero si se dispusiera de una interfaz gráfica lo haría más usable por parte del equipo docente.

- Se han propuesto veintitrés modelos de preguntas de lógica proposicional y lógica de predicados de primer orden deducidas a partir de exámenes propuestos en años anteriores en las asignaturas " *Lógica y Estructuras Discretas*“ y “ *Lógica Computacional*”, incluidas en las titulaciones de la UNED “ *Grado en Ingeniería Informática* “, “ *Grado en Ingeniería en Tecnologías de la Información”* e “ *Ingeniero en Informática*”, por su puesto se podrían implementar otros modelos que se consideren oportunos por parte del equipo docente.

- De forma adicional, en el módulo generador además de obtener datos preprocesados, da la posibilidad de obtener exámenes tipo test en formato LaTeX mostrándose las fórmulas e interpretaciones como datos comunes a todas las preguntas, sin embargo en las ítems generativos propuestos, cada pregunta tiene sus propios datos. Sería interesante estudiar la posibilidad de la implantación en SIETTE de ítems generativos con datos comunes, al igual que se muestran en el formato LaTeX.

- Aunque las preguntas se generan de forma aleatoria, parten de fórmulas proposicionales y de predicados de primer orden que el profesor debe actualizar si lo considera oportuno, sería muy interesante dar la posibilidad de generar dichas fórmulas también de forma aleatoria.

- En el uso de las fórmulas proposicionales y de predicados de primer orden, se ha utilizado la sintaxis propuesta por Prover9, por ello sería conveniente el uso de analizadores léxicos y sintácticos, de tal forma, que aquellas fórmulas que no cumpliesen los requisitos ni siquiera fueran cargadas en la aplicación.

- Cuando se generan los datos preprocesados, se necesita un conjunto de fórmulas que cumplan con un conjunto de requisitos capaz de responder a las necesidades de los modelos de preguntas, se comienza buscando aquellas que tienen al menos otras dos equivalentes, se ha

observado que para el sistema este proceso es costoso, por ello podría buscarse algún otro algoritmo que aliviase esta carga, no obstante, destacar que en ningún caso redunda en los tiempos de respuesta que el sistema ofrece al alumno ya que en SIETTE se utilizarán los datos preprocesados.

- La internacionalización de las preguntas, es decir poder generar la misma pregunta, en distintos idiomas. Este proceso aunque laborioso no sería demasiado difícil de implementar, obteniendo un resultado bastante interesante, ya que al ser SIETTE un aplicación web, no tiene restricciones geográficas, aunque el propio test impone restricciones idiomáticas, que podrían ser subsanada teniendo en cuenta este requisito no funcional.

9.4. Futuros trabajos

El tema abordado, así como la plataforma utilizada SIETTE, incita a pensar en trabajos futuros.

- Como ya se ha comentado en numerosas ocasiones, se han abordado dos temas de lógica computacional (lógica proposicional y lógica de predicados de primer orden), pero existen otros, que por si mismo podrían ser objeto de un proyecto cada uno, como por ejemplo: programación lógica, verificación de programas secuenciales, lógica modal y más en concreto lógica modal temporal, aunque contarían con la ventaja de poder utilizar lo ya implementado como base para afrontar nuevos retos.

- Los datos preprocesados se ha decidido almacenarlos en ficheros XML, pero podría optimizarse si se utilizasen gestores de bases de datos, aunque conllevaría coordinación con los administradores de SIETTE.

- Si los algoritmos generadores de preguntas y respuestas estuvieran lo suficientemente optimizados capaces de obtener las preguntas y respuestas en menos de un segundo como máximo, podrían eliminarse el módulo generador, así como los datos preprocesados, de hecho en una primera versión se evalúo esta posibilidad, pero los tiempos de respuesta dependiendo de las fórmulas utilizadas eran en numerosos casos superior a un segundo, además conllevaba la instalación en el servidor SIETTE de Prover9/Mace4, lo que hacía un despliegue más dificultoso.

- En el estudio de viabilidad se optó por Prover 9 / Mace4 como software reutilizable capaz de gestionar los conceptos de lógica proposicional y lógica de predicados de primer orden, pero también se estudiaron otras librerías como por ejemplo Orbital [Platzer, 2011], que se debería tener en cuenta en trabajos futuros por suministrar algoritmos para lógica, álgebra, algoritmos matemáticos y prueba de teoremas entre otras posibilidades; sería de interés probar su comportamiento, incluso la búsqueda de otras librerías que pudieran reutilizarse para la creación de nuevos en ítems generativos.

- En las demostraciones de insatisfacibilidad se ha utilizado métodos de resolución, no cabe duda, que otra opción sería el uso de tablas semánticas, el poder utilizarlas enriquecería

en gran medida la información suministrada al alumno, pero conllevaría su correspondiente construcción que en sí mismo podría considerarse un proyecto.

- Sería interesante trabajar en demostraciones que pudieran enriquecer aún más los conocimientos de los alumnos, un ejemplo sería la capacidad de generar tablas de verdad en el caso de lógica de proposicional.
- Finalmente, probar la arquitectura y conceptos presentados en este proyecto en la creación de nuevos ítems generativos relacionados con otras asignaturas, temas o áreas del saber.

Bibliografía

UNED ADMS. Proceso unificado de rational, 2012. URL http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node60.html.

Apache Ant. Apache ant 1.8.2, 2012. URL http://ant.apache.org/.

Bitbucket. Bitbucket, 2012. URL https://bitbucket.org/.

Joaquín Campillo Molina. Repositorio proyecto ficheros, 2012. URL https://bitbucket.org/jcampillom/gentestclient/downloads.

DescargaProver9Mace4. Descarga prover9 mace4, 2012. URL http://www.cs.unm.edu/~mccune/prover9/gui/v05.html.

Dropbox. Dropbox 1.2, 2012. URL http://www.dropbox.com.

Eclipse.org. Eclipse 3.5 (galileo), 2012. URL http://www.eclipse.org/downloads/.

EEES. Espacio europeo de educación superior, 2012. URL http://www.eees.es/.

EHEA. European higher education area, 2012. URL http://www.ehea.info/.

USC Center Software Engineering. Cocomo ii, 2012. URL http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html#downloads.

Evernote. Evernote 3.1, 2012. URL http://www.evernote.com.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, 2005.

GanttProject. Ganttproject 2.0.10, 2012. URL http://www.ganttproject.biz/download.

Joaquín Gracia. Patrontes de diseño, 2005. URL http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

ItemGenerativo, 2011. URL http://jupiter.lcc.uma.es/siette.wiki.es/index.php/%C3%8Dtems_generativos.

A. Manjarrés y F. J. Díez J. L. Fernández Vindel. Lógica computacional, 2003 - revisado en 2007. URL http://www.ia.uned.es/asignaturas/logica4/.

JabRef. Jabref 2.7, 2012. URL http://jabref.sourceforge.net/.

JavaForge. Mercurialeclipse, 2012. URL http://javaforge.com/project/HGE.

JUnit. Junit, 2012. URL http://www.junit.org/.

LaTeX. Latex, 2011. URL http://www.latex-project.org/.

Lyx. Lyx 2.0, 2012. URL http://www.lyx.org/Download.

MaquinaVirtualJava. Máquina virtual java, 2012. URL http://www.java.com/es/download/manual.jsp.

MathJax. Mathjax, 2012. URL http://www.mathjax.org/.

MathLM. Mathlm, 2012. URL http://www.w3.org/TR/MathML2/.

MetricaV3. Metrica v.3.0, 2012. URL http://administracionelectronica.gob.es/?_nfpb=true&_pageLabel=P60085901274201580632&langPae=es.

MiKTeX. Miktex, 2012. URL http://www.miktex.org/.

Oracle. Jdk 1.6, 2012. URL http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Andre Platzer, 2011. URL http://symbolaris.com/orbital/.

Prover9Mace4. Prover9mace4, 2009. URL http://www.Prover9.org.

Prover9Sintaxis. Prover9sintaxisformulas, 2009. URL http://www.cs.unm.edu/~mccune/mace4/manual/2009-11A/.

SIETTE, 2011. URL http://www.siette.org.

StarUML. Staruml 5.0, 2012. URL http://staruml.sourceforge.net/en/.

TeXnicCenter. Texniccenter, 2012. URL http://www.texniccenter.org/.

Apache Tomcat. Apache tomcat 6.0, 2012. URL http://tomcat.apache.org/download-60.cgi.

UbuntuProver9Mace4. Descarga prover9 mace4 ubuntu, 2012. URL http://packages.ubuntu.com/en/lucid/math/prover9.

UML. Uml, 2012. URL http://www.uml.org/.

UNED, 2012. URL http://portal.uned.es/portal/page?_pageid=93,1&_dad=portal&_schema=PORTAL.

UNEDAragon. Depósito de exámenes, 2012. URL http://www.calatayud.unedaragon.org/examenes/examenes_auth.asp.

Apéndices

A. Manual de instalación

A.1. Módulo generador

Este módulo se ejecutará en el ordenador personal del usuario encargado de generar los exámenes en formato LaTeX ó los ficheros con datos preprocesados en formato XML. Debido a las librerías utilizadas el software proporcionado es multiplataforma.

Destacar que la aplicación suministrada necesita tener instalada y configurado previamente una **máquina virtual de Java** y **Prover/Mace4** independientemente de la plataforma utilizada (Windows, Linux u otras). Por tanto la instalación conlleva los siguientes pasos:

1. Instalación de Máquina Virtual de Java 1.5 (JRE 1.5) ó superior

Es posible que ya tenga instalada una Máquina Virtual de Java adecuada, para asegurarse basta con ejecutar desde el shell de comandos: *java -version*, si obtiene como respuesta la versión de Java instalada y es igual o superior a Java 1.5, ya tiene configurado este paso, en caso de no ser así debe instalar dicho software, para más información véase MaquinaVirtualJava.

2. Instalación de Prover9/Mace4

Para la instalación de Prover9/Mace4 basta con descargarse dicho software de DescargaProver9Mace4, donde podrá encontrar versiones para Windows, Linux y Macintosh. Se han probado con éxito la instalación del paquete *Prover9-Mace4-v05-setup.exe*. También es posible descargar dicho software desde repositorios de software específicos para el sistema operativo utilizado, por ejemplo se ha descargado y probado con éxito, para la distribución de Linux Ubuntu los paquetes l*ibladr4_0.0.200902a-2_i386.deb* y *prover9_0.0.200902a-2_i386.deb*, descargados de UbuntuProver9Mace4.

3. Modificación de la variable de entorno PATH

Para facilitar que Prover9/Mace4 pueda ejecutarse sin depender del directorio en el que se haya instalado conviene configurar la variable de entorno PATH, agregando a dicha variable el path donde se ubican los ejecutables. Por ejemplo, en el caso de Windows, Prover9/Mace4 se habrá instalado en *C:\Program Files\Prover9-Mace4\bin-win32*. Una vez comprobado dicho path modifique la variable de entorno. Como prueba debe poder ejecutar desde la shell de comandos *prover9* y *mace4*

4. Descompresión del fichero GentTestUNED.rar

Basta con descomprimir el fichero *GenTestUNED.rar* en el directorio que se desee, lo puede encontrar en el DVD adjunto a esta memoria o en el repositorio web del proyecto Campillo Molina.

5. Comprobación de la instalación

Una vez realizados los cuatro pasos anteriores puede realizar dos pruebas para comprobar que la instalación ha sido correcta:

Ejecute desde el directorio donde descomprimió el fichero *GenTestUNED.rar* el siguiente comando *GenTest -h*, esta acción debe mostrarle la ayuda de la aplicación.

Si el paso anterior ha sido satisfactorio ejecute *GenTest -p*, en realidad esta opción comprueba que una sencilla fórmula proposicional es correcta, si está acción es satisfactoria significa que tanto la Máquina Virtual de Java, como Prover9/Mace4 han sido configurado correctamente.

6. Editor de LaTeX

Debido a que es posible generar exámenes en formato LaTeX, conviene instalar algún software capaz de gestionar ficheros de este tipo. Por ejemplo, se ha probado de forma satisfactoria TeXnicCenter, para más información véase TeXnicCenter

A.2. Módulo evaluador

En este caso no requiere ninguna acción por parte del usuario encargado de generar los exámenes, corresponde a los administradores del entorno de evaluación (SIETTE) su despliegue.

B. Manual de usuario

B.1. Módulo generador

Este módulo se ejecutará en el ordenador personal del profesor pudiendo realizar las siguientes acciones:

- Generar ficheros con datos preprocesados que posteriormente serán exportados al entorno de evaluación de conocimientos SIETTE.
- Generar exámenes tipo test en formato LaTeX.
- Probar la correcta instalación de componentes.

Además se podrán configurar distintas opciones, así como agregar nuevas fórmulas proposicionales y de primer orden según estime el equipo docente.

B.1.1. Opciones de configuración

Una vez instalado el módulo generador en el ordenador personal del profesor y antes de ejecutar la aplicación se debe tener en cuenta la posibilidad de configurar distintas opciones, así como agregar o eliminar fórmulas proposicionales y de primer orden, que la aplicación tendrá en cuenta a la hora de realizar los cálculos.

B.1.1.1. Gestión de fórmulas

En el directorio "*Formulas*" (ubicado en el directorio raíz de instalación) existen dos ficheros: *propositional.txt* y *firstOrder.txt* que contienen las fórmulas proposicionales y de primer orden a partir de las cuáles la aplicación realiza los cálculos. Para dar mayor flexibilidad se permite, mediante cualquier editor de textos, que el usuario agregue, elimine o comente aquellas que estime oportunas, teniendo en cuenta las siguientes premisas:

- En el fichero *propositional.txt* se gestionarán las fórmulas proposicionales, mientras que en *firstOrder.txt* las de primer orden.
- Sólo podrá haber una fórmula por línea.
- Se pueden comentar las fórmulas anteponiendo la cadena //. Las fórmulas comentadas no se tratarán durante la ejecución.

Operador y Cuantificadores	**Representación**	**Ejemplo Prover9**
Negación (¬)	-	-p
Disyunción (∨)	|	p | q
Conjunción (∧)	&	p & q
Condicional (→)	->	p -> q
Bicondicional (↔)	<->	p <-> q
Existe (∃)	exists	exists x exists y P(x,y)
Para todo (∀)	all	all x all y P(x,y)

Tabla B.1: Manual de usuario. Sintaxis fórmulas.

- La sintaxis utilizada será la propuesta por Prover9 (véase sección B.1.1.2).

B.1.1.2. Sintaxis fórmulas

La sintaxis de las fórmulas almacenadas en los ficheros *propositional.txt* y *firstOrder.txt* es la propuesta por Prover9.

En general, existe una regla para distinguir variables de constantes. Por defecto una variable empezará por una letra minúscula comprendida entre la u y la z, considerándose constantes el resto de letras. Por ejemplo en la fórmula P(a,x), el término a es una constante y x es una variable.

Los operadores y cuantificadores utilizados son los siguientes:

Con carácter general, tanto las relaciones como las funciones utilizadas en las fórmulas de predicados de primer orden se incluirán entre paréntesis y separados por comas sus constantes y variables, por ejemplo $\forall x \forall y Pxy$ se representaría en Prover9 como *all x all y P(x,y).*

Se considerará la siguiente precedencia de operadores: Primero las negaciones, luego conjunciones y disyunciones, y después condicionales y bicondicionales. No obstante, en caso de duda se aconseja el uso de paréntesis.

B.1.1.3. Gestión de parámetros

En el directorio "*properties*" (ubicado en el directorio raíz de instalación) existe un fichero denominado *configuration.properties*, en el que se podrán configurar distintos parámetros que posteriormente se mostrarán en los exámenes tipo test generados en formato LaTeX.

Tiene especial relevancia la propiedad **desordenarFormulas**, que podrá tener dos valores posibles (true o false), si se actualizase a true, después de ser cargadas las fórmulas (véase sección B.1.1.1) durante la ejecución de la aplicación se procederá a desordenarlas para obtener una mayor aleatoriedad, acción que no ocurrirá en caso de estar a false.

Existen otras dos propiedades **tiempoProver9** y **tiempoMace4**, que servirán para ajustar el tiempo máximo en segundos en el que Prover9 y Mace4 devuelva una respuesta.

B.1.2. Ejecución del módulo

Para la ejecución de este módulo podrá utilizar algunos de los ficheros (ubicados en el directorio raíz de instalación) *genTest.bat* (Windows) o *genTest.sh* (Linux), su sintaxis será la siguiente:

GenTest [-t [-f:file.xml]][-g[:n]][-p][-h[elp]]

B.1.2.1. Generación de ficheros con datos preprocesados

La principal finalidad de este módulo es la obtención de ficheros con datos preprocesados en formato XML, que posteriormente serán exportados al entorno de evaluación de conocimientos SIETTE (véase sección B.2.1). Para este fin se dispone de dos modificadores:

- **GenTest -g**: Generará un fichero XML con datos preprocesados, configurado para desplegar en SIETTE.
- **GenTest -g:n**: Generará n ficheros XML con datos preprocesados, configurado para desplegar en SIETTE. Por ejemplo si se ejecuta *GenTest -g:3*, se generarán tres ficheros XML con datos preprocesados.

En ambos casos los ficheros generados se depositarán en el directorio *LogicTestsXML* (ubicado en el directorio raíz de instalación).

B.1.2.2. Generación de exámenes tipo test en formato LaTeX

Como finalidad opcional de este módulo se podrán generar exámenes tipo test en formato LaTeX. Para este fin se dispone de dos modificadores:

- **GenTest -t**: Generará un examen en formato LaTeX, incluyendo en los datos comunes cuatro fórmulas proposicionales y cuatro de primer orden.
- **GenTest -t -f:file.xml**: Generará un examen en formato LaTeX, tomando como referencia el fichero.xml (este fichero corresponderá a un fichero XML con datos preprocesados generado previamente). El examen resultante contendrá en sus datos comunes tres fórmulas proposicionales y tres fórmulas de primer orden. Por ejemplo, si se ejecuta *GenTest -t -f:LogicExam0000.xml* (donde *LogicExam0000.xml* es un fichero con datos preprocesados generado previamente) creará un fichero con un examen tipo test en formato LaTeX.

En ambos casos los ficheros generados se depositarán en el directorio *LaTexTests* (ubicado en el directorio raíz de instalación).

B.1.2.3. Otras opciones

Además de los modificadores descritos anteriormente existen otros dos a tener encuentra:

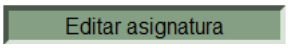
- **GenTest -p**: Permitirá comprobar que todos los componentes necesarios, tales como Prover9 y Mace4 están instalados correctamente.
- **GenTest -h**: Mostrará una ayuda con una descripción de uso de los distintos modificadores.

B.2. Módulo evaluador

Antes de utilizar las instrucciones mostradas a continuación se supone que el profesor tiene una cuenta en SIETTE con derechos de administración sobre la asignatura *Lógica Computacional*.

B.2.1. Configuración de la asignatura

En primer lugar deberá configurar los parámetros de la asignatura:

- El profesor accederá a SIETTE, acreditándose con su usuario y contraseña.
- Se accederá a la opción Editar asignatura. 

- Clic sobre la asignatura Álgebra Computacional . 

- A continuación transferir todos los ficheros XML con datos preprocesados, creados con el módulo generador (véase A.1), mediante el panel Gestión de Archivos, una vez seleccionado el directorio ac/LogicExam.

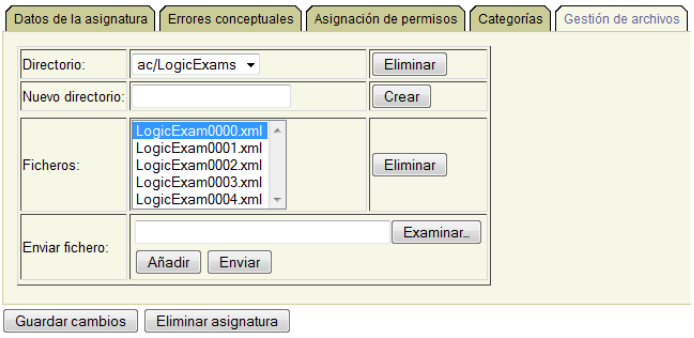


Figura B.1: Manual Usuario SIETTE. Gestión de archivos preprocesados.

- Asegúrese que en el panel Datos de la Asignatura la opción Utilizar fórmulas MathJax está a Sí.

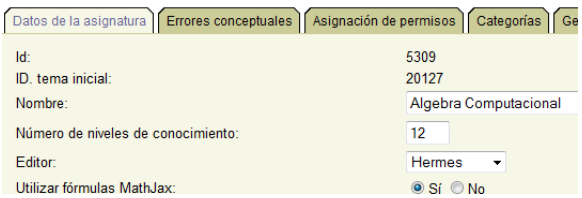


Figura B.2: Manual Usuario SIETTE. Configuración de MathJax.

- Aunque se han preconfigurado dos temas (Lógica de proposiciones y Lógica de predicados de primer orden)

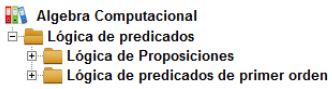


Figura B.3: Manual Usuario SIETTE. Temas preconfigurados.

puede agregar los temas que considere oportunos con la opción Nuevo > Tema.

Figura B.4: Manual Usuario SIETTE. Nuevo tema.

B.2.2. Creación de preguntas

Para crear preguntas siga los siguientes pasos:

- Si necesita crear una pregunta sin tener ninguna otra como referencia, seleccione el tema de la pregunta.

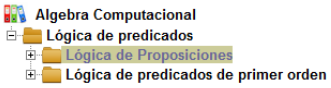


Figura B.5: Manual Usuario SIETTE. Selección tema.

- Seleccione la opción Nuevo > Pregunta.

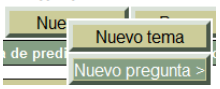


Figura B.6: Manual Usuario SIETTE. Nueva pregunta.

- Seleccione el tipo Opción Múltiple, Respuesta Única

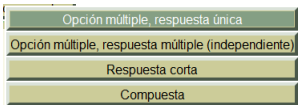


Figura B.7: Manual Usuario SIETTE. Pregunta opción múltiple.

- En la pregunta que está configurando escriba el enunciado el siguiente código JSP:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<%@ page import="es.uned.genTest.view.*" %>
<% PreguntaLogica preguntaLogica = new ProposicionalSatisfacible( "/ac
    /LogicExams/*",Random.nextLong());
 String enunciado = preguntaLogica.getEnunciado();
 String respuestaCorrecta = preguntaLogica.getRespuestaCorrecta();
 String respuestaIncorrecta1 = preguntaLogica.getRespuestasIncorrectas()
    .get(0);
 String respuestaIncorrecta2 = preguntaLogica.getRespuestasIncorrectas()
    .get(1);
 String ayudaRespuestaCorrecta = preguntaLogica.getSolucionCorrecta();
 String ayudaRespuestaIncorrecta1 = preguntaLogica.
    getSolucionesInCorrectas().get(0);
 String ayudaRespuestaIncorrecta2 = preguntaLogica.
    getSolucionesInCorrectas().get(1);
 String conceptosGenerales = preguntaLogica.getConceptosGenerales(); %>
<%= enunciado %>
```

- A continuación agregue las respuestas tal y como se muestra en la figura.

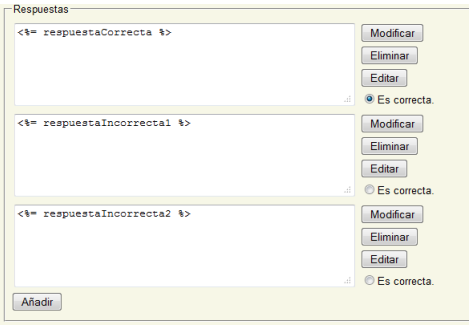


Figura B.8: Manual Usuario SIETTE. Agregar respuestas.

- Para agregar el refuerzo a la respuesta correcta, para ello pulse el botón Modificar (ubicado a la derecha de *<%= respuestaCorrecta %>*) y rellene la caja de Refuerzo con el siguiente texto.

```
<%= ayudaRespuestaCorrecta %><br><br>
 Pero le interesar&aacute; saber que:<br><br>
<%= ayudaRespuestaIncorrecta1 %><br><br>
 y adem&aacute;s: <br><br>
<%= ayudaRespuestaIncorrecta2 %>
```

Por tanto, el refuerzo quedará configurado tal y como se muestra en la figura:

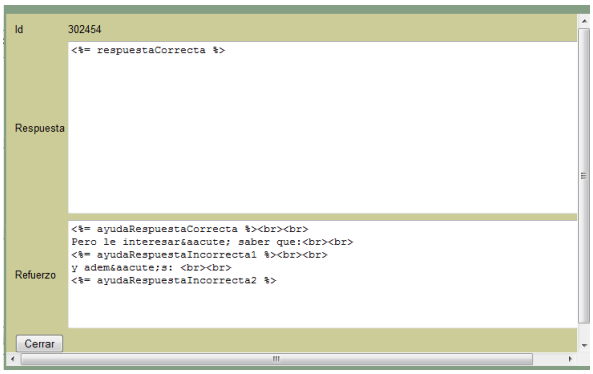


Figura B.9: Manual Usuario SIETTE. Refuerzo respuesta correcta.

A continuación pulse el botón Cerrar.

- Para agregar el refuerzo a las respuestas incorrectas, pulse el botón Modificar (ubicado a la derecha de *<%= respuestaIncorrecta1 %>* y *<%= respuestaIncorrecta2 %>* respectivamente) y rellene la caja de Refuerzo con el siguiente texto (en el caso de la respuesta incorrecta 2 sustituya ayudaRespuestaIncorrecta1 por ayudaRespuestaIncorrecta2):

```
<%= ayudaRespuestaIncorrecta1 %><br><br>
 Sin embargo puede constatar que:<br><br>
<%= ayudaRespuestaCorrecta %><br><br>
```

Por tanto, el refuerzo quedará configurado tal y como se muestra en la figura (recuerde realizar la misma acción para la respuesta incorrecta 2):

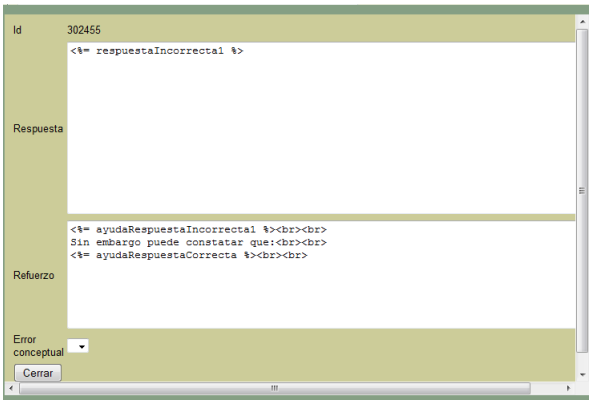


Figura B.10: Manual Usuario SIETTE. Refuerzo respuestas incorrectas.

A continuación pulse el botón Cerrar.

- Configure la opción de refuerzo.

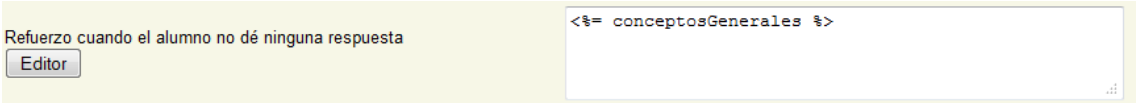


Figura B.11: Manual Usuario SIETTE. Refuerzo conceptos generales.

- Y las ayuda.

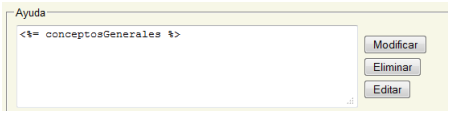


Figura B.12: Manual Usuario SIETTE. Ayuda conceptos generales.

- Finalmente Guarde Cambios.

- En el panel *Previsualizar*, puede comprobar como lo verá el alumno.

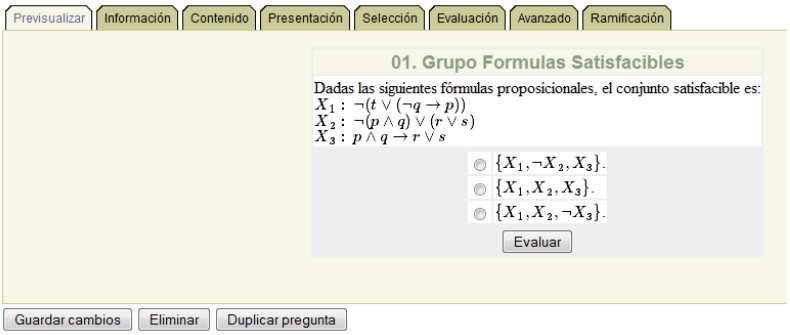


Figura B.13: Manual Usuario SIETTE. Previsualización de preguntas.

- A partir de esta pregunta generar otras preguntas es sumamente sencillo, lo único que deberá hacer es seleccionar el botón Duplicar Pregunta

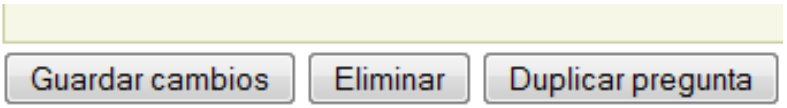


Figura B.14: Manual Usuario SIETTE. Duplicar pregunta.

y sustituir en el enunciado el modelo de pregunta (tiene veintitrés modelos disponibles), es decir en *PreguntaLogica preguntaLogica = new* ***ProposicionalSatisfacible****("/ac/LogicExams/*",Random.nextLong());* sustituir ProposicionalSatisfacible por otro de los mostrados a en la tabla adjunta (se detalla cada modelo con una pregunta de ejemplo que podrá consultar en Ápendice C):

Módelo de Pregunta	**Número de Pregunta Ápendice C**
ProposicionalSatisfacible	1
ProposicionalInSatisfacible	2
ProposicionalTautologia	3
ProposicionalNoTautologia	4
ProposicionalConsecuencia	5
ProposicionalNoConsecuencia	6
ProposicionalEquivalentes	7
ProposicionalNoEquivalentes	8
PredicadoSatisfacible	9
PredicadoInSatisfacible	10
PredicadoTautologia	11
PredicadoNoTautologia	12
PredicadoConsecuencia	13
PredicadoNoConsecuencia	14
PredicadoEquivalentes	15
PredicadoNoEquivalentes	16
PredicadoInterpretacionInSatisfacible	17
PredicadoInterpretacionSatisfacible	18
PredicadoInterpretacionNoSatisfaceFormula	19
PredicadoInterpretacionSatisfaceFormula	20
PredicadoInterpretacionSatisfacibleFormulaSoloUna	21
PredicadoInterpretacionSatisfacibleFormulaAmbas	22
PredicadoInterpretacionSatisfacibleFormulaNinguna	23

Tabla B.2: Manual de usuario. Modelos de preguntas disponibles.

.

- La cadena *"/ac/LogicExams/*"* indica que se elija al azar un fichero de datos preprocesado, pero si el profesor lo estima oportuno puede seleccionar uno en concreto, por ejemplo para forzar que se utilice el fichero *LogicExam0000.xml* se escribiría

```
PreguntaLogica preguntaLogica = new ProposicionalSatisfacible("/ac/
    LogicExams/LogicExam0000.xml",Random.nextLong());
```

B.2.3. Creación de exámenes tipo test

A continuación, basándose en las preguntas creadas en el punto anterior, el profesor podrá configurar los exámenes tipo test que crea oportunos, para ello siga los siguientes pasos:

- Seleccione botón Tests.

Figura B.15: Manual Usuario SIETTE. Botón Tests.

- Seleccione botón Nuevo Test.

Figura B.16: Manual Usuario SIETTE. Botón Nuevo Test.

- Cumplimente el panel de Información tal y como se muestra en la figura:

Figura B.17: Manual Usuario SIETTE. Panel de Información del test.

En la **Descripción** es aconsejable escribir el siguiente texto aclaratorio en formato HTML:

> *<B><B>¡AVISO!</B></B>

*
> *Tenga en cuenta que en cada pregunta se le presentará un botón de ayuda <B>Ayuda 1 (x 1.0)</B>, al ser pulsada se le mostrarán conceptos generales teóricos respecto a la pregunta en curso.

*
> *Al finalizar el test le mostrará una solución detallada tanto de las respuestas correctas como incorrectas.*

En el **número de preguntas,** el **criterio de selección,** así como **los temas** podrán variar según las preferencias del profesor**.**

C. Ejemplo de test generado de forma automática

1. El conjunto satisfacible es:
   a) $\{X_1, \neg X_2, \neg X_3\}$
   b) $\{\neg X_1, \neg X_2, X_3\}$
   c) $\{\neg X_1, X_2, \neg X_3\}$
2. El conjunto insatisfacible es:
   a) $\{X_1, \neg X_2, \neg X_3\}$
   b) $\{\neg X_1, X_2, \neg X_3\}$
   c) $\{\neg X_1, \neg X_2, \neg X_3\}$
3. Señale la tautología:
   a) $\{\neg X_1 \wedge \neg X_2 \rightarrow X_3\}$
   b) $\{X_1 \wedge \neg X_2 \rightarrow X_3\}$
   c) $\{\neg X_1 \wedge X_2 \rightarrow X_3\}$
4. Señale cual no es tautología:
   a) $\{\neg X_1 \wedge \neg X_2 \rightarrow \neg X_3\}$
   b) $\{\neg X_1 \wedge X_2 \rightarrow X_3\}$
   c) $\{X_1 \wedge \neg X_2 \rightarrow X_3\}$

5. La consecuencia correcta es:

   a) $\{\neg X_1, X_2 \models X_3\}$

   b) $\{\neg X_1, \neg X_2 \models X_3\}$

   c) $\{X_1, \neg X_2 \models X_3\}$

6. La consecuencia no correcta es:

   a) $\{\neg X_1, \neg X_2 \models \neg X_3\}$

   b) $\{\neg X_1, X_2 \models X_3\}$

   c) $\{X_1, \neg X_2 \models X_3\}$

7. Indique la fórmula equivalente a X_1.

   a) $p \vee q \rightarrow r \wedge s$

   b) $\neg\left(\neg t \rightarrow (q \vee p)\right)$

   c) $p \wedge q \rightarrow r \vee s$

8. Indique la formula no equivalente a X_1.

   a) $\neg\left(\neg t \rightarrow (q \vee p)\right)$

   b) $\neg t \wedge \neg\left(q \vee p\right)$

   c) $p \wedge q \rightarrow r \vee s$

9. El conjunto satisfacible es:

   a) $\{Y_1, \neg Y_2, Y_3\}$

   b) $\{Y_1, Y_2, Y_3\}$

   c) $\{Y_1, Y_2, \neg Y_3\}$

10. El conjunto insatisfacible es:

    a) $\{Y_1, \neg Y_2, \neg Y_3\}$

    b) $\{Y_1, Y_2, \neg Y_3\}$

    c) $\{Y_1, Y_2, Y_3\}$

11. Señale la tautología:

    a) $\{Y_1 \wedge \neg Y_2 \rightarrow Y_3\}$

    b) $\{Y_1 \wedge Y_2 \rightarrow Y_3\}$

    c) $\{Y_1 \wedge Y_2 \rightarrow \neg Y_3\}$

12. Señale cual no es tautología:

    a) $\{Y_1 \wedge \neg Y_2 \rightarrow \neg Y_3\}$

    b) $\{Y_1 \wedge Y_2 \rightarrow \neg Y_3\}$

    c) $\{Y_1 \wedge Y_2 \rightarrow Y_3\}$

13. La consecuencia correcta es:

    a) $\{Y_1, Y_2 \models \neg Y_3\}$

    b) $\{Y_1, \neg Y_2 \models Y_3\}$

    c) $\{Y_1, Y_2 \models Y_3\}$

14. La consecuencia no correcta es:

    a) $\{Y_1, Y_2 \models Y_3\}$

    b) $\{Y_1, \neg Y_2 \models \neg Y_3\}$

    c) $\{Y_1, Y_2 \models \neg Y_3\}$

15. Indique la fórmula equivalente a Y_1.

    a) $\forall x \forall y \left(Qxy \rightarrow \exists z \left(Qxz \wedge Qzy\right)\right)$

    b) $\exists x \forall y Sxy$

    c) $\neg \left(\exists x \exists y \neg Sxy\right)$

16. Indique la fórmula no equivalente a Y_1.

    a) $\exists x \forall y Sxy$

    b) $\exists x \neg \left(\exists y \neg Sxy\right)$

    c) $\neg \left(\exists x \exists y \neg Sxy\right)$

17. Sobre el Universo $U = \{0, 1\}$. ¿Qué interpretación satisface $Y_1 \wedge Y_2 \wedge \neg Y_3$?.

    a) $S = \emptyset, Q = \emptyset$

    b) $S = \{(0, 0), (0, 1)\}, Q = \emptyset$

    c) es insatisfacible

18. Sobre el Universo $U = \{0, 1\}$. ¿Qué interpretación satisface $Y_1 \wedge Y_2 \wedge Y_3$?.

    a) $Q = \emptyset, S = \emptyset$

    b) $Q = \emptyset, S = \{(0, 0), (0, 1)\}$

    c) es insatisfacible

19. La interpretacion $I1$ no satisface:

    a) Y_3

    b) $\neg Y_2$

    c) Y_2

20. La interpretación $I3$ satisface:

    a) Y_2

    b) $\neg Y_3$

    c) Y_3

21. La interpretación $I1$ satisface:

    a) Solo Y_2

    b) a Y_2 y $\neg Y_1$

    c) ni a Y_2 ni a $\neg Y_1$

22. La interpretación $I2$ satisface:

    a) Solo $\neg Y_1$

    b) a $\neg Y_1$ y Y_2

    c) ni a $\neg Y_1$ ni a Y_2

23. La interpretación $I1$ satisface:

    a) Solo $\neg Y_1$

    b) a $\neg Y_1$ y $\neg Y_2$

    c) ni a $\neg Y_1$ ni a $\neg Y_2$

Datos

Lógica Proposicional.
$X_1 : \neg (t \vee (\neg q \to p))$
$X_2 : p \wedge q \to r \vee s$
$X_3 : p \vee q \to r \wedge s$

Lógica de Predicados.
$Y_1 : \neg (\forall x \exists y \neg Sxy)$
$Y_2 : \forall x \forall y (Qxy \to \exists z (Qxz \wedge Qzy))$
$Y_3 : \forall x \forall y (\exists z (Qxz \wedge Qzy) \vee \neg Qxy)$

Interpretaciones.
I_1^Y: dominio $U = \{0, 1\}$, con $S = \{(0, 0), (0, 1)\}$, $Q = \emptyset$
I_2^Y: dominio $U = \{0, 1\}$, con $S = \emptyset$, $Q = \emptyset$
I_3^Y: dominio $U = \{0, 1\}$, con $Q = \{(0, 1)\}$, $S = \{(0, 0), (0, 1)\}$

D. Resultados

D.1. Tabla estadísticas de tiempos

Los datos expuestos corresponden a la generación de 1.000 preguntas de cada modelo (en total 23.000 preguntas). Todos los valores se expresan en milisegundos para mayor información véase 8.2.1.4. Se acompaña entre paréntesis el número de pregunta, incluidas en el ejemplo de examen del Apéndice C.

Estadística de tiempos (milisegundos) generación de preguntas proposicionales

Tipo Pregunta	Media	Desviación Típica	Máximo	Mínimo
Satisfacible (1)	10,1	7,06	38	6
InSatisfacible (2)	9,64	7,14	42	5
Consecuencia (5)	9,45	6,95	39	5
No Consecuencia (6)	10,31	7,11	42	6
Tautología (3)	10,8	9,33	76	5
No Tautología (4)	11,55	9,49	85	6
Equivalente (7)	10,2	7,29	44	5
No Equivalente (8)	11,29	7,32	48	6

Tabla D.1: Resultados. Tiempos generación preguntas proposicionales.

Estadística de tiempos (milisegundos) generación de preguntas de preguntas de predicados de primer orden

Tipo Pregunta	**Media**	**Desv. Típica**	**Máximo**	**Mínimo**
Satisfacible (9)	11,91	9,61	77	5
Insatisfacible (10)	9,53	6,9	32	5
Consecuencia (13)	9,59	6,8	33	5
No Consecuencia (14)	10,69	7,11	34	5
Tautología (11)	11,81	10,73	85	5
No Tautología (12)	12,12	9,29	79	5
Equivalente (15)	10,89	7,55	37	5
No Equivalente (16)	12,58	7,92	41	6
Interpretación Insatisfacible (17)	9,63	7,03	44	5
Interpretación Satisfacible (18)	9,53	7,21	39	4
Interpretación Satisface Fórmula (20)	9,26	7,17	38	4
Interpretación No Satisface Fórmula (19)	9,47	7,26	40	4
Interpretación Satisface Fórmula Ambas (22)	9,6	7,42	48	4
Interpretación Satisface Fórmula Ninguna (23)	9,31	7,1	33	4
Interpretación Satisface Fórmula Sólo Una (21)	9,43	7,25	37	5

Tabla D.2: Resultados. Tiempos generación preguntas primer orden.

D.2. Tabla cantidad de preguntas

Los datos expuestos corresponden a una tabla resumen de la cantidad de cuestiones que se pueden obtener por cada tipo de pregunta en función de un fichero con datos preprocesados XML, para mayor información del razonamiento seguido en su obtención véase 8.1. Debe tenerse en cuenta que son datos estimados, por tanto variarán dependiendo del fichero tomado como referencia. Se acompaña entre paréntesis el número de pregunta, incluidas en el ejemplo de examen del Apéndice C.

Resumen cantidad de posibles preguntas a partir de un fichero preprocesado. Proposicional.

Tipo Pregunta	**Cantidad de Preguntas**
Satisfacible (1)	15
InSatisfacible (2)	30
Consecuencia (7)	30
No Consecuencia (8)	15
Tautología (3)	30
No Tautología (4)	15
Equivalente (5)	20
No Equivalente (6)	5

Tabla D.3: Resultados. Cantidad de preguntas proposicionales.

Resumen cantidad de posibles preguntas a partir de un fichero preprocesado. Predicados de primer orden

Tipo Pregunta	**Cantidad de Preguntas**
Satisfacible (9)	15
Insatisfacible (10)	30
Consecuencia (15)	30
No Consecuencia (16)	15
Tautología (11)	30
No Tautología (12)	15
Equivalente (13)	20
No Equivalente (14)	5
Interpretación Insatisfacible (17)	5
Interpretación Satisfacible (18)	3
Interpretación Satisface Fórmula (20)	16
Interpretación No Satisface Fórmula (19)	16
Interpretación Satisface Fórmula Ambas (22)	12
Interpretación Satisface Fórmula Ninguna (23)	12
Interpretación Satisface Fórmula Sólo Una (21)	16

Tabla D.4: Resultados. Cantidad de preguntas primer orden.