UNIVERSIDAD DE OVIEDO

MÁSTER UNIVERSITARIO EN INGENIERÍA WEB

TRABAJO FIN DE MÁSTER

"S.C.O.T.-XML SISTEMA DE COMPARACIÓN DE OPCIONES PARA EL TRATAMIENTO DE XML"

SIMÓN CARLEOS ARTIME

Directores: B. Cristina Pelayo García-Bustelo
Jordán Pascual Espada

Oviedo, julio de 2014

Resumen

S.C.O.T.-XML es un sistema para la comparación de diferentes tecnologías para el tratamiento de XML que hace uso del desarrollo orientado a modelos.

Este sistema surge de la dificultad de comparar las diferentes tecnologías que permiten trabajar sobre XML. Para comparar su rendimiento se debe de implementar el código y ejecutar pruebas midiendo los tiempos de ejecución, lo que puede resultar pesado y un derroche de tiempo si se tienen que comparar muchas consultas diferentes.

S.C.O.T.-XML busca facilitar estar pruebas automatizándolas, haciendo uso de un lenguaje específico de dominio para expresar consultas sobre un documento XML y de la creación automática de código para obtener las implementaciones de los diferentes test.

Palabras Clave

Web, XML, servicio, modelo, ANTLR.

Abstract

SCOT-XML is a system for comparing different XML processing technology that uses model-driven development.

This system comes up from the difficulty of comparing the different technologies to work with XML. To compare their performances you need to implement the code and run tests measuring the execution times, which can be tedious and a waste of time if you have to compare many different queries.

SCOT-XML seeks to facilitate this testing by automating it, using a domain-specific language to express queries over an XML document and automatic code creation to obtain the different implementations of the tests.

Keywords

Web, XML, service, model, ANTLR.

Índice General

1MEMORIA DEL PROYECTO....11

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto....11
1.2 Resumen de Todos los Aspectos....11
1.3 Posibles ámbitos de aplicación....13

2INTRODUCCIÓN....14

2.1 Justificación del Proyecto....14
2.2 Objetivos del Proyecto....14
2.3 Estudio de la Situación Actual....15

3ASPECTOS TEÓRICOS....18

3.1 XML....18
3.2 C#....18
3.2.1 LINQ....18
3.2.2 XmlDocument....19
3.2.3 XmlReader....19
3.3 JAX-RS....19
3.4 Ingeniería orientada a modelos....19
3.5 Lenguaje específico de dominio....20

4PLANIFICACIÓN DEL PROYECTO Y RESUMEN DE PRESUPUESTOS....21

4.1 Planificación....21
4.2 Resumen del Presupuesto....23

5ANÁLISIS....24

5.1 Definición del Sistema....24
5.1.1 Determinación del Alcance del Sistema....24
5.2 Requisitos del Sistema....25
5.2.1 Obtención de los Requisitos del Sistema....25
5.2.2 Identificación de Actores del Sistema....28
5.2.3 Especificación de Casos de Uso....29
5.3 Identificación de los Subsistemas en la Fase de Análisis....31
5.3.1 Descripción de los Subsistemas....31
5.3.2 Descripción de los Interfaces entre Subsistemas....31
5.4 Diagrama de Clases Preliminar del Análisis....32
5.4.1 Diagrama de Clases....32
5.4.2 Descripción de las Clases....32
5.5 Análisis de Casos de Uso y Escenarios....37
5.5.1 Caso de uso 1: Solicitud de testeo....37
5.5.2 Caso de uso 2: Consulta de estado....38
5.5.3 Caso de uso 3: Obtención de resultados....38
5.5.4 Caso de uso 4: Cancelación de solicitud....39
5.5.5 Caso de uso 5: Ejecución de solicitud....39
5.5.6 Caso de uso 6: Cancelación de ejecución....41
5.5.7 Caso de uso 7: Ejecución de trabajo....42
5.5.8 Caso de uso 8: Eliminación de trabajos pasados de tiempo....43
5.6 Análisis de Interfaces de Usuario....43

5.6.1 Descripción de la Interfaz....43
5.6.2 Descripción del Comportamiento de la Interfaz....45
5.7 Especificación del Plan de Pruebas....45

6DISEÑO DEL SISTEMA....47

6.1 Arquitectura del Sistema....47
6.1.1 Diagramas de Paquetes....47
6.1.2 Diagrama de Componentes....48
6.1.3 Diagrama de Despliegue....49
6.2 Diseño de Clases....50
6.2.1 Diagrama de Clases....50
6.3 Diagramas de Interacción y Estados....51
6.3.1 Caso de Uso 1: Solicitud de testeo....51
6.3.2 Caso de Uso 2: Consulta de estado....52
6.3.3 Caso de Uso 3: Obtención de resultados....53
6.3.4 Caso de Uso 4: Cancelación de testeo....54
6.3.5 Caso de Uso 5: Ejecución de testeo....55
6.3.6 Caso de Uso 6: Cancelación de ejecución....56
6.3.7 Caso de uso 7: Ejecución de trabajo....57
6.3.8 Caso de uso 8: Eliminación de trabajos pasados de tiempo....58
6.3.9 Diagrama de estados de los trabajos....59
6.4 Diagramas de Actividades....61
6.4.1 Caso de uso 7: Ejecución de trabajo....61
6.5 Diseño de la Interfaz....62
6.5.1 Interfaz gráfica Web....62
6.5.2 Interfaz de servicios Web....63
6.6 Diseño del lenguaje específico del dominio y modelo....65
6.7 Diseño de la transformación del modelo a código....69
6.7.1 XmlDocument....69
6.7.2 LINQ....69
6.7.3 XmlReader....70
6.8 Especificación Técnica del Plan de Pruebas....70
6.8.1 Pruebas de Integración y del Sistema....71
6.8.2 Pruebas de Usabilidad....74

7IMPLEMENTACIÓN DEL SISTEMA....76

7.1 Lenguajes de Programación....76
7.1.1 Java....76
7.1.2 ECMAScript....76
7.1.3 C#....77
7.2 Herramientas y Programas Usados para el Desarrollo....77
7.2.1 Eclipse....77
7.2.2 Dia....78
7.2.3 GIMP....78
7.2.4 Web Sequence Diagrams....79
7.2.5 Microsoft Project....79
7.2.6 JQuery....79
7.2.7 Google Code Prettify....80
7.2.8 Google Charts....80
7.2.9 Maven....80
7.2.10 ANTLR....81
7.2.11 ANTLR IDE 4 para Eclipse....81
7.2.12 Tomcat....81

7.2.13 Mono....82
7.2.14 Spring....82
7.2.15 Jersey....82
7.3 Creación del Sistema....82
7.3.1 Problemas Encontrados....82
7.3.2 Descripción Detallada de las Clases....83

8DESARROLLO DE LAS PRUEBAS....87

8.1 Pruebas de Integración y del Sistema....87
8.2 Pruebas de Usabilidad y Accesibilidad....87
8.2.1 Pruebas de Usabilidad....87

9MANUALES DEL SISTEMA....91

9.1 Manual de Instalación y ejecución....91
9.2 Manual de Usuario....98
9.3 Manual del Programador....102
9.3.1 Cómo agregar implementaciones de test para nuevas tecnologías....102
9.3.2 Cómo crear un cliente para la API Web del sistema....105

10CONCLUSIONES Y AMPLIACIONES....107

10.1 Conclusiones....107
10.2 Ampliaciones....108

11PRESUPUESTO....109

11.1 Presupuesto del cliente....109
11.2 Presupuesto de costes....109

12REFERENCIAS BIBLIOGRÁFICAS....112

12.1 Referencias....112

13APÉNDICES....114

13.1 Glosario y Diccionario de Datos....114
13.2 Contenido Entregado en el CD-ROM....115
13.2.1 Contenidos....115
13.2.2 Código Ejecutable e Instalación....115
13.2.3 Ficheros de Configuración....115

Índice de Figuras

FIGURA 2.1: FORMULARIO DE INTRODUCCIÓN DE CÓDIGO DE JSPERF.COM 16
FIGURA 2.2: RESULTADOS DE UNA COMPARACIÓN DE RENDIMIENTO DE TECNOLOGÍAS XML PUBLICADA EN INTERNET 17
FIGURA 4.1: DIAGRAMA DE GANTT 22
FIGURA 5.1: DIAGRAMA DE CASOS DE USO 29
FIGURA 5.2: DIAGRAMA DE CLASES PRELIMINAR 32
FIGURA 5.3: BOCETO DE LA INTERFAZ 44
FIGURA 6.1: DIAGRAMA DE PAQUETES 47
FIGURA 6.2: DIAGRAMA DE COMPONENTES 48
FIGURA 6.3: DIAGRAMA DE DESPLIEGUE 49
FIGURA 6.4: DIAGRAMA DE CLASES 50
FIGURA 6.5: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO SOLICITUD DE TESTEO 51
FIGURA 6.6: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO CONSULTA DE ESTADO 52
FIGURA 6.7: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO OBTENCIÓN DE RESULTADOS 53
FIGURA 6.8: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO CANCELACIÓN DE TESTEO 54
FIGURA 6.9: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO EJECUCIÓN DE TESTEO 55
FIGURA 6.10: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO CANCELACIÓN DE EJECUCIÓN 56
FIGURA 6.11: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO EJECUCIÓN DE TRABAJO 57
FIGURA 6.12: DIAGRAMA DE SECUENCIA PARA EL CASO DE USO ELIMINACIÓN DE TRABAJOS PASADOS DE TIEMPO 58
FIGURA 6.13: DIAGRAMA DE ESTADOS DE LOS TRABAJOS 59
FIGURA 6.14: DIAGRAMA DE ACTIVIDADES DEL CASO DE USO EJECUCIÓN DE TRABAJO 61
FIGURA 6.15: ASPECTO FINAL QUE TENDRÁ LA INTERFAZ 63
FIGURA 6.16: ESQUEMA DEL MODELO DEL DSL 68
FIGURA 7.1: PANTALLA DE ECLIPSE 77
FIGURA 7.2: PANTALLA DE DIA 78
FIGURA 7.3: PANTALLA DE GIMP 78

FIGURA 7.4: PÁGINA INICIAL DE WEB SEQUENCE DIAGRAMS.....79
FIGURA 7.5: CÓDIGO COLOREADO CON GOOGLE CODE PRETTIFY.....80
FIGURA 7.6: GRÁFICO DE BARRAS GENERADO CON GOOGLE CHARTS.....80
FIGURA 9.1: PANTALLA DE SELECCIÓN PARA INSTALAR XTEXT.....92
FIGURA 9.2: PANTALLA DE SELECCIÓN PARA INSTALAR ANTLR 4 IDE.....93
FIGURA 9.3: PANTALLA DE PROPIEDADES DE ANTLR4 IDE.....94
FIGURA 9.4: PESTAÑA SERVERS DE ECLIPSE.....95
FIGURA 9.5: PANTALLA DE CREACIÓN DE SERVIDOR.....96
FIGURA 9.6: PANTALLA DE CREACIÓN DE SERVIDOR II.....97
FIGURA 9.6: INICIO DEL SERVIDOR DESDE ECLIPSE.....97
FIGURA 9.7: FORMULARIO DE LA INTERFAZ WEB.....98
FIGURA 9.8: PANEL DE MENSAJES DE LA INTERFAZ WEB.....98
FIGURA 9.9: PANTALLA DE UN RESULTADO MOSTRADO EN LA INTERFAZ WEB.....99
FIGURA 9.10: PANTALLA DEL GRÁFICO DE RESULTADOS DE LA INTERFAZ WEB.....99

1 Memoria del Proyecto

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

XML es un lenguaje ampliamente utilizado en la informática en general y en la Web en particular. Lógicamente esto hace que a su alrededor surjan multitud de herramientas y tecnologías para trabajar con él, y a la hora de incluir el tratamiento de XML en un sistema surge la necesidad de escoger la tecnología que se utilizará.

El problema es que es difícil saber a priori que herramienta ofrecerá el mejor rendimiento para la situación específica en que se utilizará. La mejor forma de comprobarlo es probando cada tecnología, pero para ello hay que implementar el código, lo que en sistemas con muchas consultas resulta pesado y lleva demasiado tiempo.

Así surge la idea de S.C.O.T.-XML, un sistema que ayude a automatizar este trabajo de implementación y testeo de las tecnologías que tratan XML.

Este proyecto pretende crear un sistema que, recibiendo un comando en un lenguaje específico que exprese una operación de búsqueda sobre un archivo XML, genere código que ejecute esa operación en cada una de las tecnologías a probar y mida sus tiempos de ejecución.

La aplicación tendrá una API en forma de servicios Web. También se pretende crear una interfaz Web que permita a los usuarios utilizar el sistema directamente.

No es objetivos de este proyecto hacer un sistema para exponer directamente en redes públicas como Internet, por lo que no se tendrán en cuenta las exigencias de seguridad frente a ataques que serían necesarias en este escenario. Si se desea exponer en redes públicas se deberán implementar las medidas necesarias, ya sea con una fachada, con un filtro o con alguna otra tecnología.

1.2 Resumen de Todos los Aspectos

El contenido de este documento se estructura de la siguiente manera:

- En la memoria del proyecto (capítulo 1) se hace un resumen del proyecto de manera que sirva de introducción para personas que desconozcan la naturaleza del proyecto o sus objetivos.

- En la introducción (capítulo 2) se describe la motivación para el desarrollo del proyecto, explicándose de forma no técnica en qué consiste, qué necesidades pretende cubrir y cuáles son sus principales objetivos.
- En los aspectos teóricos (capítulo 3) se describen brevemente los conceptos, herramientas y tecnologías existentes relevantes para el proyecto.
- En la planificación del proyecto y resumen del presupuesto (capítulo 4) se muestra una estimación inicial del coste que supondría el proyecto y del plazo de ejecución.
- En el análisis (capítulo 5) se analizan las funcionalidades que debe cubrir el sistema desarrollado, así como los requisitos que debe cumplir. También se define cómo debería ser la interfaz de usuario y qué pruebas se deberían realizar para comprobar su correcto funcionamiento.
- En el diseño del sistema (capítulo 6) se expone la arquitectura del sistema y su comportamiento, diseñada en base a los requerimientos del capítulo anterior. También se muestra la que será la interfaz de usuario final y las pruebas definitivas.
- En la implementación del sistema (capítulo 7) se muestran las tecnologías y herramientas utilizadas durante la fase de implementación, así como los problemas encontrados y el esquema de clases final.
- En el desarrollo de pruebas (capítulo 8) se muestran los resultados de las pruebas realizadas sobre el sistema.
- En los manuales del sistema (capítulo 9) se recogen las guías que describen cómo instalar, ejecutar y utilizar el sistema, así como una guía para programadores que quieran ampliar, modificar o simplemente entender mejor la aplicación.
- En las conclusiones y ampliaciones (capítulo 10) se da una visión personal de lo logrado con el proyecto, y se proponen posibles ampliaciones para mejorarlo y ampliar su funcionalidad.
- En el presupuesto (capítulo 11) se muestran los presupuestos de cliente y de costes del proyecto.
- En las referencias bibliográficas (capítulo 12) se muestran las principales fuentes de información para el proyecto.
- En los apéndices (capítulo 13) se presenta el contenido del CD-ROM del proyecto y los ficheros de configuración para la aplicación.

1.3 Posibles ámbitos de aplicación

S.C.O.T.-XML tendría varios ámbitos en los que podría resultar de utilidad.

Podría ser utilizado en proyectos en los que es importante la eficiencia al tratar XML, ya sea debido a que se utilizan documentos muy pesados o porque hay un elevado número de consultas, para conocer qué tecnología da el mejor resultado evitando así la necesidad de implementar a mano todas las pruebas.

También puede ser utilizado en el ámbito de la docencia, donde puede ayudar a mostrar cómo se utilizan las tecnologías implementadas y sus eficiencias en distintos escenarios.

Puesto que el núcleo de S.C.O.T.-XML será principalmente un generador de código, también puede usarse como tal, ya sea directamente por los usuarios a través de su interfaz Web copiando y pegando el código, o como subsistema de un sistema de generación más grande a través de sus servicios.

2 Introducción

2.1 Justificación del Proyecto

Cuando una tecnología tiene éxito surgen a su alrededor multitud de herramientas y tecnologías derivadas que pretenden facilitar el trabajo con ella. Esta variedad da a los desarrolladores la libertad de poder escoger entre varias opciones para cubrir sus necesidades, pero también les pone en la situación de tener que escoger la tecnología que hará mejor el trabajo.

Una de las características que más se suelen tener en cuenta a la hora de comparar tecnologías es la del rendimiento, tomando como tal el tiempo de ejecución. Como cada tecnología puede mostrar diferentes rendimientos dependiendo del escenario la forma más fiable de compararlas es testear su desempeño en el escenario concreto que nos interese, pero esto requiere de la implementación de ese escenario en cada una de las tecnologías, lo que lleva demasiado trabajo y tiempo.

Una solución a este problema es el uso del desarrollo orientado a modelos. Se crea un modelo que permita definir esos escenarios, y procedimientos que permitan crear automáticamente implementaciones de esos escenarios en cada una de las tecnologías a probar. De esta manera, introduciendo un modelo obtenemos directamente el código a testear.

Una tecnología muy extendida en la informática, especialmente en la Web, es XML, el conocido lenguaje de marcas para expresar información jerárquica de manera textual, siendo comprensible para computadoras y humanos.

Como es lógico, este éxito hace que haya muchas tecnologías para trabajar con él, surgiendo el problema de cuál elegir. Es de esta problemática de la que surge la idea de este proyecto, un sistema que permita automatizar el testeo del rendimiento de operaciones sobre XML.

2.2 Objetivos del Proyecto

Los objetivos del proyecto son los siguientes:

- Diseñar un lenguaje específico de dominio para expresar las operaciones sobre XML. Concretamente nos centraremos en las consultas de búsqueda de nodos.
- Diseñar el proceso de transformación de ese lenguaje a código para realizar los test de rendimiento. Esta tarea se tendrá que realizar para cada una de las tecnologías que se quieran poder testear. En este

proyecto trataremos de crear implementaciones para las tecnologías que ofrece C#: LINQ, XmlDocument y XmlReader.

- Diseñar e implementar un sistema que use los dos elementos anteriores para automatizar el testeo de operaciones expresadas en el lenguaje especifico de dominio sobre documentos XML. Este sistema tendrá que ser fácilmente utilizable por otros sistemas, por lo que ofrecerá servicios Web a modo de API.
- Diseñar e implementar una interfaz Web que permita a los usuarios trabajar directamente con este sistema.

No son objetivos de este proyecto centrarse en la seguridad del sistema, y serán los sistemas que a su vez utilicen a este los que tendrán que preocuparse de este aspecto. La API en forma de servicios no estará diseñada para ser expuesta directamente en Internet.

2.3 Estudio de la Situación Actual

Actualmente no existe, que el autor haya encontrado, ningún sistema como S.C.O.T.-XML.

Existen algunos sistemas para ayudar a automatizar test de rendimiento, pero siendo los usuarios los que deben crear estos test. Tal es el ejemplo de jsperf.com, que permite que los usuarios creen test para ECMAScript con ayuda de formularios y los compartan con otras personas. Estos sistemas ayudan a la creación y publicación de test, pero no automatizan el proceso de creación del código de pruebas.

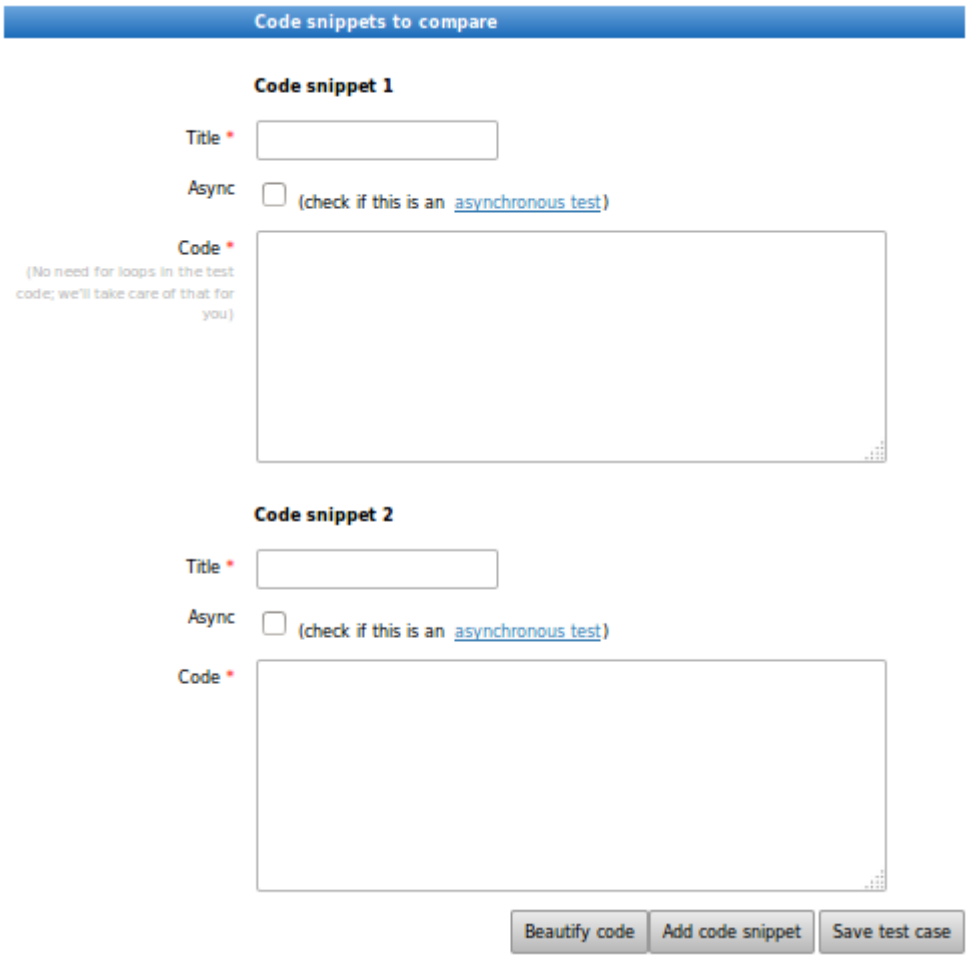
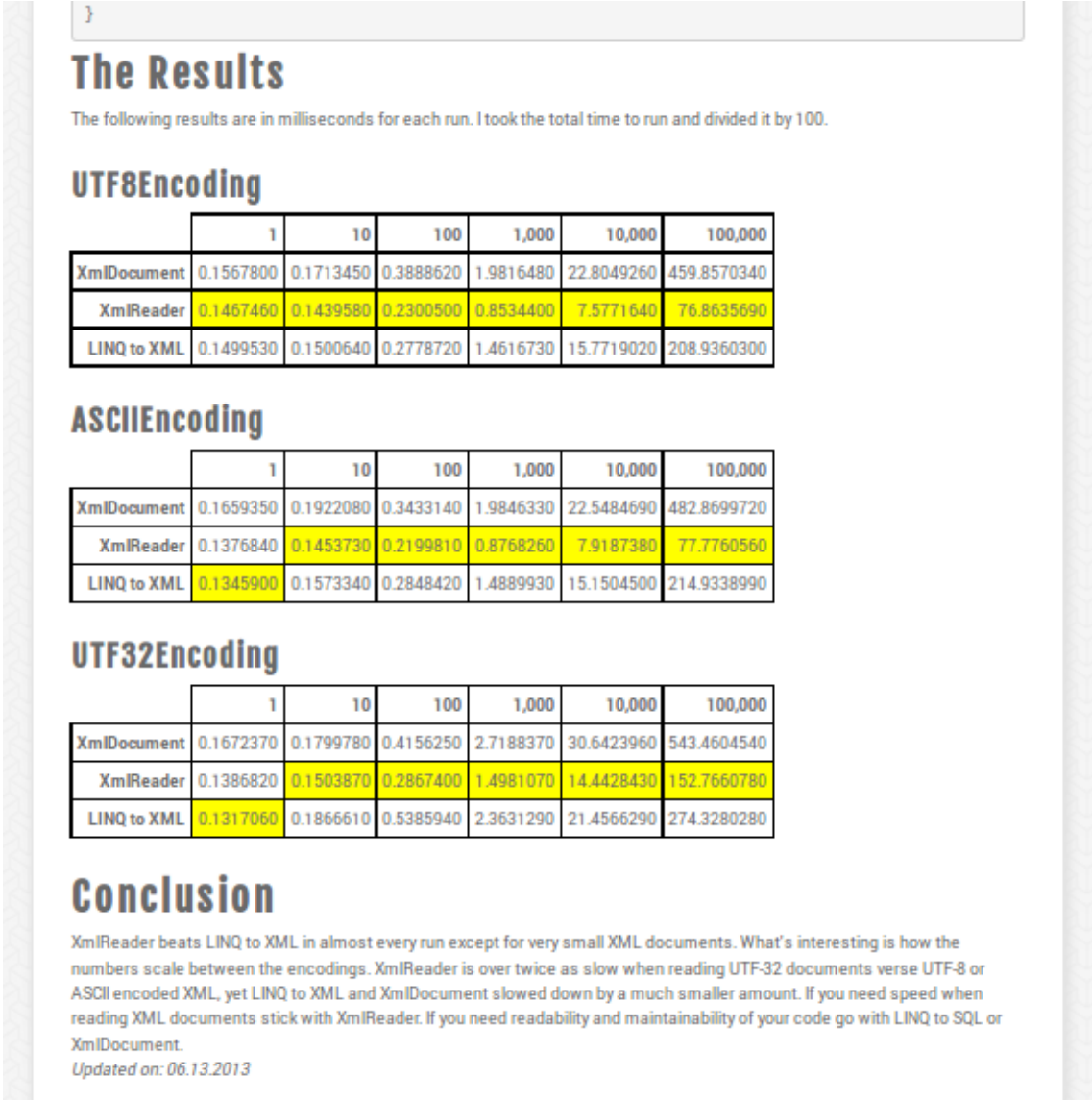


Figura 2.1: Formulario de introducción de código de jsperf.com

También existen multitud de estudios de rendimiento de las tecnologías que implementaremos, pero que proporcionan únicamente el resultado de las pruebas realizadas, pruebas concretas que no necesariamente son extrapolables a otros ámbitos o situaciones. Estos estudios ofrecen una muestra general del rendimiento de los sistemas, pero no ofrecen la precisión y confianza de una prueba concreta de la operación a implementar.

Un ejemplo de estos estudios es el que se encuentra en la dirección https://www.altamiracorp.com/blog/employee-posts/performance-linq-to-sql-vs, y del que se muestra una captura de pantalla a continuación.



}

The Results

The following results are in milliseconds for each run. I took the total time to run and divided it by 100.

UTF8Encoding

	1	10	100	1,000	10,000	100,000
XmlDocument	0.1567800	0.1713450	0.3888620	1.9816480	22.8049260	459.8570340
XmlReader	0.1467460	0.1439580	0.2300500	0.8534400	7.5771640	76.8635690
LINQ to XML	0.1499530	0.1500640	0.2778720	1.4616730	15.7719020	208.9360300

ASCIIEncoding

	1	10	100	1,000	10,000	100,000
XmlDocument	0.1659350	0.1922080	0.3433140	1.9846330	22.5484690	482.8699720
XmlReader	0.1376840	0.1453730	0.2199810	0.8768260	7.9187380	77.7760560
LINQ to XML	0.1345900	0.1573340	0.2848420	1.4889930	15.1504500	214.9338990

UTF32Encoding

	1	10	100	1,000	10,000	100,000
XmlDocument	0.1672370	0.1799780	0.4156250	2.7188370	30.6423960	543.4604540
XmlReader	0.1386820	0.1503870	0.2867400	1.4981070	14.4428430	152.7660780
LINQ to XML	0.1317060	0.1866610	0.5385940	2.3631290	21.4566290	274.3280280

Conclusion

XmlReader beats LINQ to XML in almost every run except for very small XML documents. What's interesting is how the numbers scale between the encodings. XmlReader is over twice as slow when reading UTF-32 documents verse UTF-8 or ASCII encoded XML, yet LINQ to XML and XmlDocument slowed down by a much smaller amount. If you need speed when reading XML documents stick with XmlReader. If you need readability and maintainability of your code go with LINQ to SQL or XmlDocument.

Updated on: 06.13.2013

Figura 2.2: Resultados de una comparación de rendimiento de tecnologías XML publicada en Internet

Por último, también hay muchos artículos académicos en la red sobre la creación de test a partir de modelos, pero no se encontró ninguna herramienta o sistema que permita automatizar este proceso, menos aún para el ámbito concreto del trabajo con XML.

Así, no se pudo encontrar ningún sistema o herramienta que nos brinde las características que se buscan en este proyecto.

3 Aspectos Teóricos

3.1 XML

Este proyecto gira en torno a esta tecnología. XML es un lenguaje de marcas desarrollado por la World Wide Web Consortium (W3C) para representar como texto información estructurada de manera que sea legible para las personas e interpretable por las computadoras.

XML ofrece una manera fácil de representar y transmitir información estructurada.

XML es un estándar abierto, lo que permite su libre utilización por individuos y organizaciones y que sea uno de los estándares más extendidos y utilizados en el ámbito de la Web.

3.2 C#

C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft como parte del framework .NET. Su sintaxis y semántica se basa en la de C++.

C# es uno de los lenguajes diseñados para la infraestructura de lenguaje común (especificación de un entorno de ejecución de aplicaciones, inicialmente creado por Microsoft).

Las implementaciones en código del modelo de nuestro sistema se harán en C#. Se escogió este lenguaje por tener varias tecnologías para el tratamiento de XML, que nos permitirán probar nuestro sistema.

3.2.1 LINQ

LINQ es una de las tecnologías que ofrece la plataforma .NET para trabajar con XML (y otras fuentes de datos).

LINQ convierte el documento XML a una colección de objetos XElement, que puede ser consultada usando el motor local de ejecución de LINQ.

LINQ ofrece una forma fácil de trabajar con fuentes de datos como XML haciendo uso de un lenguaje de consulta similar a los usados en bases de datos.

3.2.2 XmlDocument

XmlDocument es otra tecnología que ofrece la plataforma .NET para trabajar con XML.

XmlDocument carga el documento XML en memoria como un conjunto de objetos, y ofrece diversos métodos para consultar y manipular los nodos.

3.2.3 XmlReader

XmlReader es otra de las tecnologías que ofrece la plataforma .NET para trabajar con XML.

XmlReader es la más básica de estas tecnologías, proporcionando un acceso de bajo nivel al documento. No posee almacenamiento en caché y sólo permite recorrer el documento hacia delante.

3.3 JAX-RS

JAX-RS es una API del lenguaje de programación Java que permite la creación de servicios Web RESTful a partir clases planas Java (POJOs) marcadas con anotaciones específicas.

De esta manera, basta con marcar los métodos que se quieran exponer como servicios Web indicando el path y método HTTP en que escuchará cada uno de ellos, y la implementación de JAX-RS creará el servicio Web automáticamente.

La implementación de JAX-RS se usará en el proyecto es Jersey.

JAX-RS se utilizará en este proyecto para la creación de la API Web del sistema a desarrollar, ya que permite de manera fácil la creación y modificación de servicios Web.

3.4 Ingeniería orientada a modelos

La ingeniería orientada a modelos es una metodología de desarrollo de software basada en la creación y explotación de modelos de dominio, en lugar de centrarse en algoritmos.

En esta metodología el software se crea mediante la conversión automática del modelo a código, por lo que para poder utilizar esta metodología hay que diseñar tanto el modelo como el proceso de conversión.

El aspecto que resulta interesante de esta metodología para este proyecto es que si se crean distintas transformaciones del modelo a código se pueden obtener distintas implementaciones a partir del mismo modelo.

En este proyecto se usará esta metodología para crear los test de las diferentes tecnologías que pretendemos comparar a partir del modelo. De esta manera y partiendo de una entrada en el lenguaje específico de dominio, se podrán obtener implementaciones en cada una de las tecnologías a probar.

3.5 Lenguaje específico de dominio

Un lenguaje específico de dominio es un lenguaje de programación dedicado a dar cobertura a una situación o dominio en particular, en contraposición a los lenguajes de propósito general (como C# o Java) que están diseñados para dar cobertura a multitud de posibles dominios.

El beneficio de usar un lenguaje específico de dominio es que ofrece un mayor nivel de abstracción, encapsulando complejidades que no es necesario tener en cuenta en el dominio en cuestión, y mejorando así la productividad y la facilidad de uso.

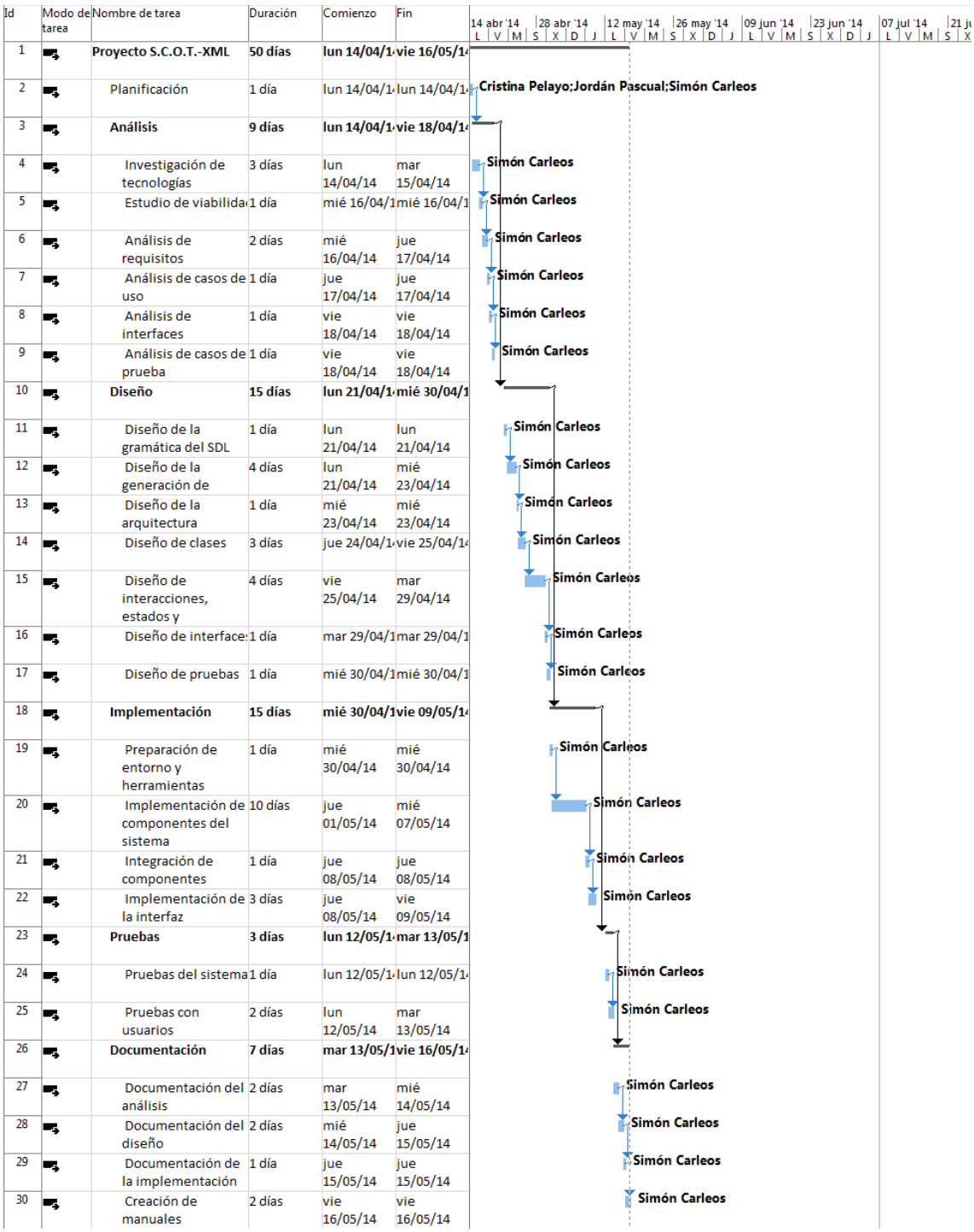
En este proyecto se usa un lenguaje específico de dominio para expresar las consultas que se quieren testear.

4 Planificación del Proyecto y Resumen de Presupuestos

4.1 Planificación

Para la planificación se fijó una jornada laboral de cuatro horas diarias (es decir, media jornada).

El diagrama de Gantt de la planificación es el que sigue:



Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		**Proyecto S.C.O.T.-XML**	**50 días**	**lun 14/04/14**	**vie 16/05/14**
2		Planificación	1 día	lun 14/04/14	lun 14/04/14
3		**Análisis**	**9 días**	**lun 14/04/14**	**vie 18/04/14**
4		Investigación de tecnologías	3 días	lun 14/04/14	mar 15/04/14
5		Estudio de viabilidad	1 día	mié 16/04/14	mié 16/04/14
6		Análisis de requisitos	2 días	mié 16/04/14	jue 17/04/14
7		Análisis de casos de uso	1 día	jue 17/04/14	jue 17/04/14
8		Análisis de interfaces	1 día	vie 18/04/14	vie 18/04/14
9		Análisis de casos de prueba	1 día	vie 18/04/14	vie 18/04/14
10		**Diseño**	**15 días**	**lun 21/04/14**	**mié 30/04/14**
11		Diseño de la gramática del SDL	1 día	lun 21/04/14	lun 21/04/14
12		Diseño de la generación de	4 días	lun 21/04/14	mié 23/04/14
13		Diseño de la arquitectura	1 día	mié 23/04/14	mié 23/04/14
14		Diseño de clases	3 días	jue 24/04/14	vie 25/04/14
15		Diseño de interacciones, estados y	4 días	vie 25/04/14	mar 29/04/14
16		Diseño de interfaces	1 día	mar 29/04/14	mar 29/04/14
17		Diseño de pruebas	1 día	mié 30/04/14	mié 30/04/14
18		**Implementación**	**15 días**	**mié 30/04/14**	**vie 09/05/14**
19		Preparación de entorno y herramientas	1 día	mié 30/04/14	mié 30/04/14
20		Implementación de componentes del sistema	10 días	jue 01/05/14	mié 07/05/14
21		Integración de componentes	1 día	jue 08/05/14	jue 08/05/14
22		Implementación de la interfaz	3 días	jue 08/05/14	vie 09/05/14
23		**Pruebas**	**3 días**	**lun 12/05/14**	**mar 13/05/14**
24		Pruebas del sistema	1 día	lun 12/05/14	lun 12/05/14
25		Pruebas con usuarios	2 días	lun 12/05/14	mar 13/05/14
26		**Documentación**	**7 días**	**mar 13/05/14**	**vie 16/05/14**
27		Documentación del análisis	2 días	mar 13/05/14	mié 14/05/14
28		Documentación del diseño	2 días	mié 14/05/14	jue 15/05/14
29		Documentación de la implementación	1 día	jue 15/05/14	jue 15/05/14
30		Creación de manuales	2 días	vie 16/05/14	vie 16/05/14

Figura 4.1: Diagrama de Gantt

El número de tareas es de 30. Las tareas que más duran son las de diseño e implementación, con 15 días cada una. La tarea que más dura entre las que son hoja (no tienen subtareas) es la implementación de componentes del sistema con 10 días. Hay varias tareas con la duración más corta, que es de 1 día.

Está planeado que el proyecto comience el lunes 14 de abril del 2014 y termine el 14 de junio de 2014.

4.2 Resumen del Presupuesto

A continuación se resume el presupuesto de costes del proyecto. El presupuesto del cliente (mostrado en el apartado de presupuesto) es muy simple y no merece la pena resumirlo.

Nombre	Descripción	Unidades	Coste unitario (€)	Coste total (€)
Amortización				16,68
Oficina				430
Personal				9280
Subtotal				9726,68
Subtotal+10%				10699,35
Total (IVA 21%)				12946,21

5 Análisis

5.1 Definición del Sistema

5.1.1 Determinación del Alcance del Sistema

Para llevar a cabo los objetivos de este proyecto se deberán cubrir los siguientes puntos.

- Diseñar y definir mediante una gramática el lenguaje específico de dominio que usaremos en nuestra aplicación y obtener los analizadores para ese lenguaje mediante el uso de ANTLR.
- Diseñar e implementar un sistema capaz de generar y ejecutar los test de tecnologías XML usando generación automática de código basada en modelos.
- Diseñar e implementar una API Web de servicios RESTful para el sistema mencionado.
- Diseñar e implementar una interfaz gráfica Web que permita utilizar la funcionalidad del sistema.

5.2 Requisitos del Sistema

5.2.1 Obtención de los Requisitos del Sistema

5.2.1.1 Requisitos funcionales

Lenguaje específico

Código	Nombre Requisito	Descripción del Requisito
RF01	Lenguaje específico	Se deberá diseñar un lenguaje específico de modelo para expresar las consultas a testear sobre el documento XML.
RF02	Relaciones entre nodos	El lenguaje específico deberá ser capaz de expresar búsquedas de nodos en un documento XML en función de sus relaciones (padre, hijo, descendiente).
RF03	Encadenamiento de relaciones	El lenguaje específico deberá ser capaz de expresar relaciones encadenadas (p.e. el nodo 'x' hijo de 'y' que es ancestro de 'z').
RF04	Filtrado por atributos	El lenguaje específico deberá ser capaz de expresar el filtrado de nodos en base a los valores de sus atributos.
RF05	Combinación lógica de condiciones	Los filtros por atributo deberán poder combinarse usando los operadores lógicos 'and' y 'or'.
RF06	Filtrado por cadenas de texto	Se diseñarán filtros para la igualdad y desigualdad de los atributos con una cadena dada.
RF07	Filtrado por valores numéricos	Se diseñarán filtros para atributos numéricos, en función de si son iguales, diferentes, mayores, menores, mayores o iguales, o menores o iguales que un valor numérico dado.
RF08	Unión de consultas	Se permitirá hacer unión de consultas.
RF09	Múltiples consultas	Se permitirá expresar varias consultas en un solo comando.

API Web

Código	Nombre Requisito	Descripción del Requisito
RF10	API Web	El sistema debe tener una API en forma de servicios Web RESTful.
RF11	Solicitud de testeo	La API debe tener un método que permita solicitar al sistema que lleve a cabo un testeo de tecnologías. Recibirá como parámetros la operación a testear expresada en el lenguaje específico de modelo, el número de repeticiones de los test y un documento XML sobre el que realizar los test. Devolverá el identificador del trabajo creado o un error.
RF12	Consulta de estado	La API debe tener un método que permita consultar el estado en que se encuentra una solicitud. Recibirá como parámetro el identificador del trabajo recibido en la solicitud. Devolverá el estado del trabajo solicitado.
RF13	Obtención de resultados	La API debe tener un método que permita obtener los resultados de los test. Recibirá como parámetro el identificador del trabajo recibido en la solicitud. Los resultados incluirán el código generado, los nodos encontrados y el tiempo de ejecución, separados por tecnologías.
RF14	Cancelación de solicitud	La API debe tener un método que permita cancelar una solicitud para ahorrarle al servidor el consumo de recursos en un trabajo que ya no se necesita. Recibirá como parámetro el identificador del trabajo recibido en la solicitud.

Tecnologías

Código	Nombre Requisito	Descripción del Requisito
RF15	Implementaciones de tecnologías	Se deberá tratar de obtener implementaciones para las tecnologías por defecto de C#: LINQ, XmlDocument y XmlReader.
RF16	Facilidad para añadir tecnologías	El sistema deberá tener un diseño que permita agregar implementaciones para otras tecnologías sin demasiada dificultad.

Funcionamiento

Código	Nombre Requisito	Descripción del Requisito
RF17	Cola de ejecución	El sistema deberá contar con una cola de ejecución para las solicitudes, de manera que se ejecuten de una en una, evitando sobrecarga y variaciones de carga que afecten al resultado de los test.
RF18	Liberación de recursos	El sistema deberá de eliminar los archivos en disco (documentos XML, código y ejecutables) cuando se haya acabado de trabajar con ellos. También se deberán eliminar los trabajos que lleven demasiado tiempo sin ser consultados (trabajos abandonados por el cliente).

Interfaz

Código	Nombre Requisito	Descripción del Requisito
RF19	Interfaz Web	Para poder trabajar con el sistema directamente se implementará una interfaz Web que permita hacer peticiones a través de un formulario y mostrar los resultados por pantalla.
RF20	Formulario de entrada	La interfaz dispondrá de un formulario para la entrada de los datos necesarios para la solicitud: la consulta a testear en el lenguaje específico, el número de repeticiones del test y el archivo XML sobre el que realizar el test.
RF21	Cuadro de mensajes	La interfaz dispondrá de un cuadro de mensajes que informe al usuario de los eventos relevantes.
RF22	Consulta continua	Una vez que el servidor haya aceptado una solicitud y devuelto un identificador de trabajo, la interfaz consultará por el estado de este cada pocos segundos hasta que esté finalizado.
RF23	Muestra de resultados	Una vez finalizado el trabajo la interfaz obtendrá los resultados y los mostrará por pantalla. Los resultados a mostrar por cada consulta y tecnología son: entrada que generó el resultado, código generado a partir de esta entrada, nodos encontrados con la consulta y tiempo de ejecución de la consulta X veces (siendo X el valor indicado por el usuario en el formulario).

5.2.1.2 Requisitos de usuario

Código	Nombre Requisito	Descripción del Requisito
RU01	Conocimiento de XML	Para poder usar la aplicación los usuarios deberán tener ciertos conocimientos básicos sobre XML, como qué son los nodos o elementos, tipos de relaciones entre ellos (padre, hijo, descendiente, etc.) y qué son los atributos.

5.2.1.3 Requisitos tecnológicos

Código	Nombre Requisito	Descripción del Requisito
RT01	Funcionamiento en UNIX	El sistema deberá funcionar en sistemas UNIX, puesto que el ordenador de desarrollo y pruebas tiene instalado Ubuntu 12.04. No obstante se deberá de tratar que sea fácilmente modificable para funcionar bajo sistemas Windows.

5.2.2 Identificación de Actores del Sistema

5.2.2.1 Cliente

Este actor representa a los programas o sistemas externos que realizan peticiones a los servicios de la API Web de S.C.O.T.-XML.

5.2.2.2 Usuario de la interfaz Web

Este actor representa a las personas usuarias de la interfaz Web. Nótese que la interfaz Web es a su vez un actor cliente.

5.2.3 Especificación de Casos de Uso

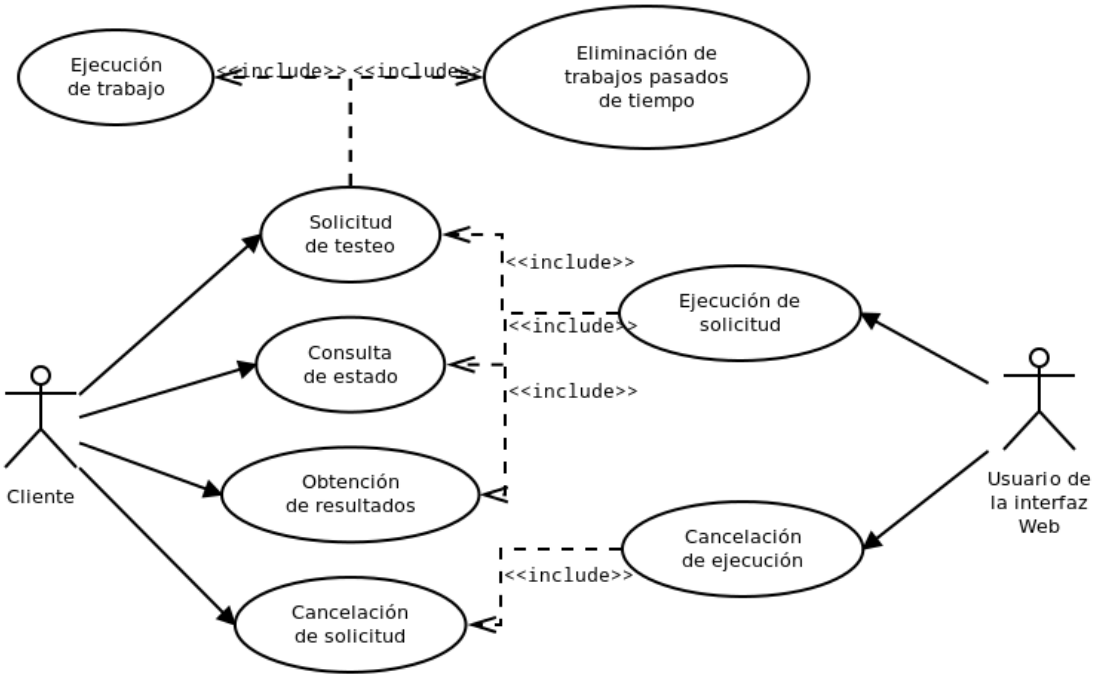


Figura 5.1: Diagrama de casos de uso

Nombre del Caso de Uso
Solicitud de testeo
Descripción
El cliente solicita al sistema que realice un testeo de una consulta sobre XML para lo que le proporciona la consulta expresada en el lenguaje específico de dominio, un número que indique el número de veces que se repetirá cada test y un documento XML sobre el que se llevarán a cabo las pruebas. El sistema crea y encola un trabajo asociado a un identificador y se retorna este identificador.

Nombre del Caso de Uso
Consulta de estado
Descripción
El cliente consulta al sistema cuál es el estado en que se encuentra un trabajo, del que le proporciona el identificador. El sistema le retorna el estado del trabajo.

Nombre del Caso de Uso
Obtención de resultados
Descripción
El cliente solicita al sistema que le proporcione los resultados de un trabajo

finalizado, del que le proporciona el identificador. El sistema le retorna los resultados, divididos primero por tecnología y luego por test. Cada resultado contiene la entrada que generó el test, el código generado, el número de nodos encontrado en el test y el tiempo de ejecución.

Nombre del Caso de Uso
Cancelación de solicitud
Descripción
El cliente informa al sistema de que puede cancelar un trabajo, del que le proporciona el identificador.

Nombre del Caso de Uso
Ejecución de solicitud
Descripción
El usuario solicita al sistema que lleve a cabo un testeo, para lo que le proporciona, vía formulario, la consulta expresada en el lenguaje específico de dominio, el número de repeticiones de cada test y un documento XML sobre el que se llevarán a cabo las pruebas. La interfaz realiza de forma automática las tareas de consultar el estado del trabajo periódicamente y de obtener los resultados una vez finalizado.

Nombre del Caso de Uso
Cancelación de ejecución
Descripción
El usuario cancela una ejecución en curso.

Nombre del Caso de Uso
Ejecución de trabajo
Descripción
Cuando llega el turno de ejecución del trabajo el sistema lo recupera de la cola de ejecución y crea, compila y ejecuta los test a partir de la información del trabajo, obteniendo en el proceso los resultados de los test.

Nombre del Caso de Uso
Eliminación de trabajos pasados de tiempo
Descripción
Los trabajos que exceden determinado tiempo sin ser consultados por el cliente son considerados abandonados y eliminados liberando sus recursos.

5.3 Identificación de los Subsistemas en la Fase de Análisis

5.3.1 Descripción de los Subsistemas

- **API del sistema:** Subsistema que contiene los servicios Web presentados a los clientes. Es el subsistema encargado de procesar las peticiones de los clientes y devolverles las respuestas.
- **Modelo:** Subsistema encargado de obtener el modelo a partir del comando textual proporcionado por el cliente.
- **Trabajos:** Subsistema encargado de crear y gestionar los trabajos solicitados por los clientes.
- **Implementaciones:** Subsistema encargado de obtener el código de cada implementación concreta de una tecnología a partir del modelo, compilarlo, ejecutarlo y obtener los resultados de los test.
- **Interfaz Web:** Subsistema encargado de proporcionar a los usuarios una interfaz para interactuar con el sistema.

5.3.2 Descripción de los Interfaces entre Subsistemas

La interfaz Web se comunica con la API del sistema a través de llamadas HTTP a sus servicios RESTful, lo que puede hacer de manera local o remota.

Todos los demás subsistemas se comunican de forma local a través de sus interfaces.

5.4 Diagrama de Clases Preliminar del Análisis

5.4.1 Diagrama de Clases

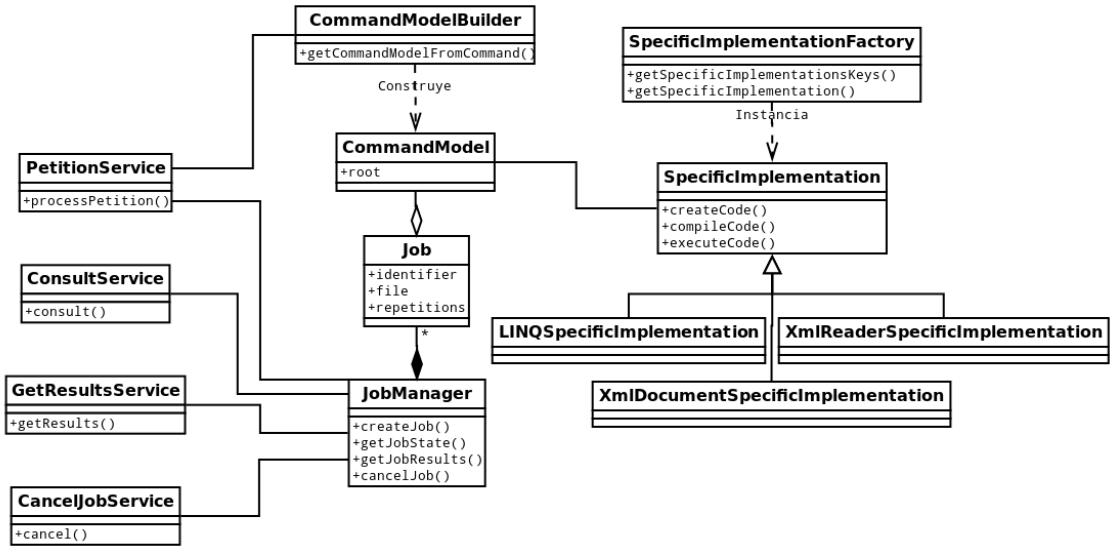


Figura 5.2: Diagrama de clases preliminar

5.4.2 Descripción de las Clases

5.4.2.1 API del sistema

Nombre de la Clase
PetitionService
Descripción
Servicio para la solicitud de testeos por parte de los clientes. Clase con etiquetas para ser usada con JAX-RS.
Responsabilidades
Llevar a cabo los procedimientos necesarios para procesar las solicitudes de los clientes.
Atributos Propuestos
Ninguno.
Métodos Propuestos
processPetition: Recibe y valida las entradas del cliente, solicita al CommandModelBuilder que cree el modelo, solicita al JobManager que cree un nuevo trabajo y retorna al cliente el identificador del nuevo trabajo.

Nombre de la Clase
ConsultService
Descripción
Servicio para la consulta del estado de un trabajo por parte de los clientes. Clase con etiquetas para ser usada con JAX-RS.
Responsabilidades
Obtener el estado del trabajo.
Atributos Propuestos
Ninguno.
Métodos Propuestos
consult: Recibe un identificador de trabajo y obtiene del JobManager el estado de este, que retorna al cliente.

Nombre de la Clase
GetJobResultsService
Descripción
Servicio para la obtención de resultados por parte de los clientes. Clase con etiquetas para ser usada con JAX-RS.
Responsabilidades
Obtener los resultados del trabajo.
Atributos Propuestos
Ninguno.
Métodos Propuestos
getResults: Recibe un identificador de trabajo y obtiene del JobManager los resultados obtenidos de su ejecución.

Nombre de la Clase
CancelJobService
Descripción
Servicio para la cancelación de un trabajo por parte de los clientes. Clase con etiquetas para ser usada con JAX-RS.
Responsabilidades
Cancelar el trabajo.
Atributos Propuestos
Ninguno.
Métodos Propuestos
cancel: Recibe un identificador de trabajo y solicita al JobManager que lo cancele.

5.4.2.2 Modelo

Nombre de la Clase
CommandModel
Descripción
Clase que contiene el modelo de la solicitud del cliente. El modelo serán clases creadas por ANTLR a partir de la gramática.
Responsabilidades
Simplemente contener el modelo.
Atributos Propuestos
root: la raíz del modelo.
Métodos Propuestos
Ninguno.

Nombre de la Clase
CommandModelBuilder
Descripción
Clase que crea el modelo. Internamente utiliza clases generadas por ANTLR.
Responsabilidades
Obtener el modelo a partir de la entrada textual proporcionada por el cliente en el lenguaje específico de dominio.
Atributos Propuestos
Ninguno.
Métodos Propuestos
getCommandModelFromCommand: Obtiene y devuelve el modelo a partir de la cadena de texto de entrada.

5.4.2.3 Trabajos

Nombre de la Clase
Job
Descripción
Clase que contiene la información de un trabajo de un cliente.
Responsabilidades
Simplemente contener la información.
Atributos Propuestos
identifier: el identificador del trabajo. **file:** el archivo XML proporcionado por el cliente. **repetitions:** el número de repeticiones para los test.
Métodos Propuestos
Ninguno.

Nombre de la Clase
JobManager
Descripción
Clase que gestiona todo lo relacionado con los trabajos.
Responsabilidades
Crear los trabajos con identificadores únicos. Mantener una cola de ejecución de trabajos. Ejecutar los trabajos. Proporcionar información de los trabajos a otras clases. Controlar la concurrencia.
Atributos Propuestos
Ninguno.
Métodos Propuestos
createJob: Recibe un modelo, un archivo XML y un número de repeticiones y crea un trabajo con identificador único, lo agrega a la cola de ejecución y retorna el identificador. **getJobState:** Recibe un identificador de trabajo y retorna su estado. El estado puede ser “no existente”. **getJobResults:** Recibe un identificador de trabajo y retorna sus resultados. **cancelJob:** Recibe un identificador de trabajo y lo cancela.

5.4.2.4 Implementaciones

Nombre de la Clase
SpecificImplementation
Descripción
Clase que ofrece una transformación del modelo a una implementación específica de una tecnología.
Responsabilidades
Crear el código a partir del modelo. Compilar el código. Ejecutar el código y obtener los resultados.
Atributos Propuestos
Ninguno.
Métodos Propuestos
createCode: Crea el código. **compileCode:** Compila el código. Para ello usa llamadas al sistema. **executeCode:** Ejecuta el código y obtiene los resultados. La ejecución se hace con llamadas al sistema.

Nombre de la Clase
SpecificImplementationFactory
Descripción
Factoría de implementaciones específicas.
Responsabilidades
Crear las implementaciones específicas.
Atributos Propuestos
Ninguno.
Métodos Propuestos
getSpecificImplementationsKeys: Retorna los nombres de las implementaciones específicas de que dispone. **getSpecificImplementation:** Recibe un nombre de implementación específica y retorna una nueva instancia.

5.5 Análisis de Casos de Uso y Escenarios

5.5.1 Caso de uso 1: Solicitud de testeo

Solicitud de testeo	
Precondiciones	Ninguna.
Poscondiciones	Debe existir un nuevo trabajo en el sistema.
Actores	Iniciado y terminado por el cliente.
Descripción	El cliente: 1. Envía una solicitud al servicio de solicitudes, enviando un mensaje multiparte con la consulta en el lenguaje específico, un fichero XML y un entero con el número de repeticiones. El sistema: 2. Comprueba que no falten entradas. 3. Crea el modelo a partir de la consulta en el lenguaje específico, validando al mismo tiempo la consulta. 4. Crea un trabajo con el modelo, el fichero y el número de repeticiones. 5. Asocia un identificador único al trabajo. 6. Encola el trabajo en la cola de ejecución. 7. Retorna el identificador del trabajo al cliente.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1**: Algún dato de entrada falta o es incorrecto. o Retornar un error.
Excepciones	• **Error de entrada-salida**: No se puede guardar el fichero. o Retornar un error.
Notas	Al crear el trabajo este escenario posibilita el inicio de los casos de uso **Ejecución de trabajo** (si el trabajo no se cancela antes de que llegue su turno) y **Eliminación de trabajos pasados de tiempo** (si el trabajo está demasiado tiempo sin ser consultado por el cliente).

5.5.2 Caso de uso 2: Consulta de estado

Consulta de estado	
Precondiciones	Ninguna.
Poscondiciones	La marca de tiempo del trabajo se actualiza
Actores	Iniciado y terminado por el cliente.
Descripción	El cliente: 1. Envía una solicitud al servicio de consulta pasándole un identificador de trabajo. El sistema: 2. Obtiene el trabajo. 3. Obtiene su estado. 4. Actualiza su marca de tiempo al momento actual. 5. Lo retorna al cliente.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1**: No existe ningún trabajo asociado al identificador. o Retornar el estado “no existente”.
Excepciones	Ninguna.
Notas	El estado incluye el estado propiamente dicho y, si este es “encolado”, un número que indica la posición en cola.

5.5.3 Caso de uso 3: Obtención de resultados

Obtención de resultados	
Precondiciones	El trabajo asociado al identificador pasado debe existir y haber finalizado.
Poscondiciones	Ninguna.
Actores	Iniciado y terminado por el cliente.
Descripción	El cliente: 1. Envía una solicitud al servicio de resultados, pasándole un identificador de trabajo. El sistema: 2. Obtiene el trabajo. 3. Obtiene los resultados del trabajo. 4. Los retorna al cliente.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1**: No existe ningún trabajo asociado al identificador o el trabajo asociado no finalizó. o Retornar null.
Excepciones	Ninguna.
Notas	-

5.5.4 Caso de uso 4: Cancelación de solicitud

Cancelación de solicitud	
Precondiciones	Ninguna.
Poscondiciones	El trabajo deja de existir en el sistema.
Actores	Iniciado y terminado por el cliente.
Descripción	El cliente: 1. Envía una solicitud al servicio de cancelación, pasándole un identificador de trabajo. El sistema: 2. Obtiene el trabajo. 3. Libera los recursos del trabajo (elimina los ficheros asociados a él). 4. Elimina el trabajo.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1**: El trabajo se está ejecutando. o Marcar el estado del trabajo como cancelado para liberar sus recursos y eliminarlo cuando esté libre.
Excepciones	Ninguna.
Notas	-

5.5.5 Caso de uso 5: Ejecución de solicitud

Ejecución de solicitud	
Precondiciones	No hay una ejecución de solicitud en curso.
Poscondiciones	Los resultados de la ejecución se muestran en la interfaz.
Actores	Iniciado y terminado por el usuario de la interfaz Web.

Descripción	El usuario: 1. Introduce los datos en el formulario: la consulta a realizar sobre el XML expresada en el lenguaje específico del dominio, el número de repeticiones para los test y el archivo XML sobre el que se realizarán. 2. Pulsa el botón aceptar. La interfaz: 3. Realiza algunas validaciones sobre las entradas. 4. Envía al servidor la solicitud enviando los datos del formulario en un mensaje multiparte, iniciando el escenario **Solicitud de testeo**. 5. Al obtener la respuesta muestra un mensaje en el cuadro de mensajes informando de la correcta creación del trabajo y desactiva el botón de aceptación y activa el de cancelación. 6. Inicia entonces una consulta periódica del estado del trabajo (con su identificador como parámetro), iniciando el escenario **Consulta de estado** de manera repetitiva. Se informa al usuario del estado mediante el cuadro de mensajes. 7. Cuando el estado es “finalizado” llama al servicio de obtención de resultados con el identificador del trabajo como parámetro, iniciando el escenario **Obtención de resultados**. 8. Una vez obtenidos los resultados los muestra por pantalla formateados y agrega un gráfico de barras. 9. Desactiva el botón de cancelación y activa el de aceptación.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1**: Se produce algún error durante los procesos de validación o de ejecución. o Mostrar los errores en el cuadro de mensajes. o Desactivar el botón de cancelación y activar el de aceptación. • **Escenario alternativo 2:** No se puede conectar con el servidor en la consulta de estado. o Mostrar el error en el cuadro de mensajes. o Seguir intentando la conexión. • **Escenario alternativo 3:** No se puede conectar con el servidor en alguna de las otras llamadas. o Mostrar los errores en el cuadro de mensajes. o Desactivar el botón de cancelación y activar el de aceptación.
Excepciones	Ninguna.
Notas	-

5.5.6 Caso de uso 6: Cancelación de ejecución

Cancelación de ejecución	
Precondiciones	Hay una ejecución de solicitud en curso.
Poscondiciones	La ejecución se detiene.
Actores	Iniciado y terminado por el usuario de la interfaz Web.
Descripción	El usuario: 1. Pulsa el botón de cancelar. La interfaz: 2. Cancela el temporizador de consulta de estado. 3. Envía al servidor la solicitud de cancelación con el identificador del trabajo en ejecución, iniciando el escenario **Cancelación de solicitud**. 4. Desactiva el botón de cancelación y activa el de aceptación.
Variaciones (escenarios secundarios)	Ninguno.
Excepciones	• **Error de conexión con el servidor:** no se puede conectar con el servidor. ◦ Se sigue con el escenario saltándose el paso 3.
Notas	-

5.5.7 Caso de uso 7: Ejecución de trabajo

Ejecución de trabajo	
Precondiciones	Hay algún trabajo en la cola de ejecución.
Poscondiciones	El trabajo contiene los resultados de los test. Los recursos del trabajo son liberados.
Actores	Iniciado y terminado por el sistema.
Descripción	El sistema: 1. Obtiene el trabajo de la cola de ejecución. 2. Marca su estado como “en ejecución”. 3. Renueva el número de cola de los trabajos encolados, de haberlos. 4. Obtiene la lista de nombres de las tecnologías disponibles para testear y por cada una: 1. Obtiene la implementación de la tecnología. 2. La configura con los datos del trabajo. 3. Crea el archivo de código de test en base al modelo del trabajo. 4. Compila el código. 5. Ejecuta el código. 6. Obtiene los resultados y los almacena. 7. Libera los recursos de los ficheros de código y ejecutable. 5. Agrega los resultados obtenidos al trabajo. 6. Libera los recursos del archivo XML del trabajo. 7. Marca el estado del trabajo como finalizado.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** No hay trabajos encolados. ◦ Se pausa la ejecución hasta recibir señal de que los hay. • **Escenario alternativo 2:** Alguno de los procesos de creación de código, compilación o ejecución falla. ◦ Seguir con la ejecución de la siguiente tecnología. ◦ Si al final ninguna tecnología arrojó resultados marcar el estado del trabajo como “error”.
Excepciones	Ninguna.
Notas	-

5.5.8 Caso de uso 8: Eliminación de trabajos pasados de tiempo

Eliminación de trabajos pasados de tiempo	
Precondiciones	Existe algún trabajo en el sistema.
Poscondiciones	Los trabajos pasados de tiempo son eliminados y sus recursos liberados.
Actores	Iniciado y terminado por el sistema.
Descripción	El sistema: 1. Calcula la marca de tiempo mínima que los trabajos deben tener, en base al momento actual y al tiempo de sesión configurado. Aquellos trabajos con una marca menor se consideran pasados de tiempo. 2. Recorre el conjunto de trabajos del sistema, comparando su marca de tiempo con la mínima calculada, y eliminando aquellos pasados de tiempo. 3. Temporiza la próxima
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** No hay trabajos en el sistema. ◦ Se pausa la ejecución hasta que se reciba señal de que hay trabajos. • **Escenario alternativo 2:** Un trabajo pasado de tiempo se está ejecutando y no es posible liberar sus recursos en el momento. ◦ Marcarlo como cancelado para que sea borrado cuando sea posible.
Excepciones	Ninguna.
Notas	-

5.6 Análisis de Interfaces de Usuario

5.6.1 Descripción de la Interfaz

La interfaz constará de una única página, sin contar con la página de ayuda que será simplemente una explicación textual del modo de uso.

La página realizará las llamadas al servidor usando AJAX y se actualizará dinámicamente.

La interfaz tendrá la siguiente estructura:

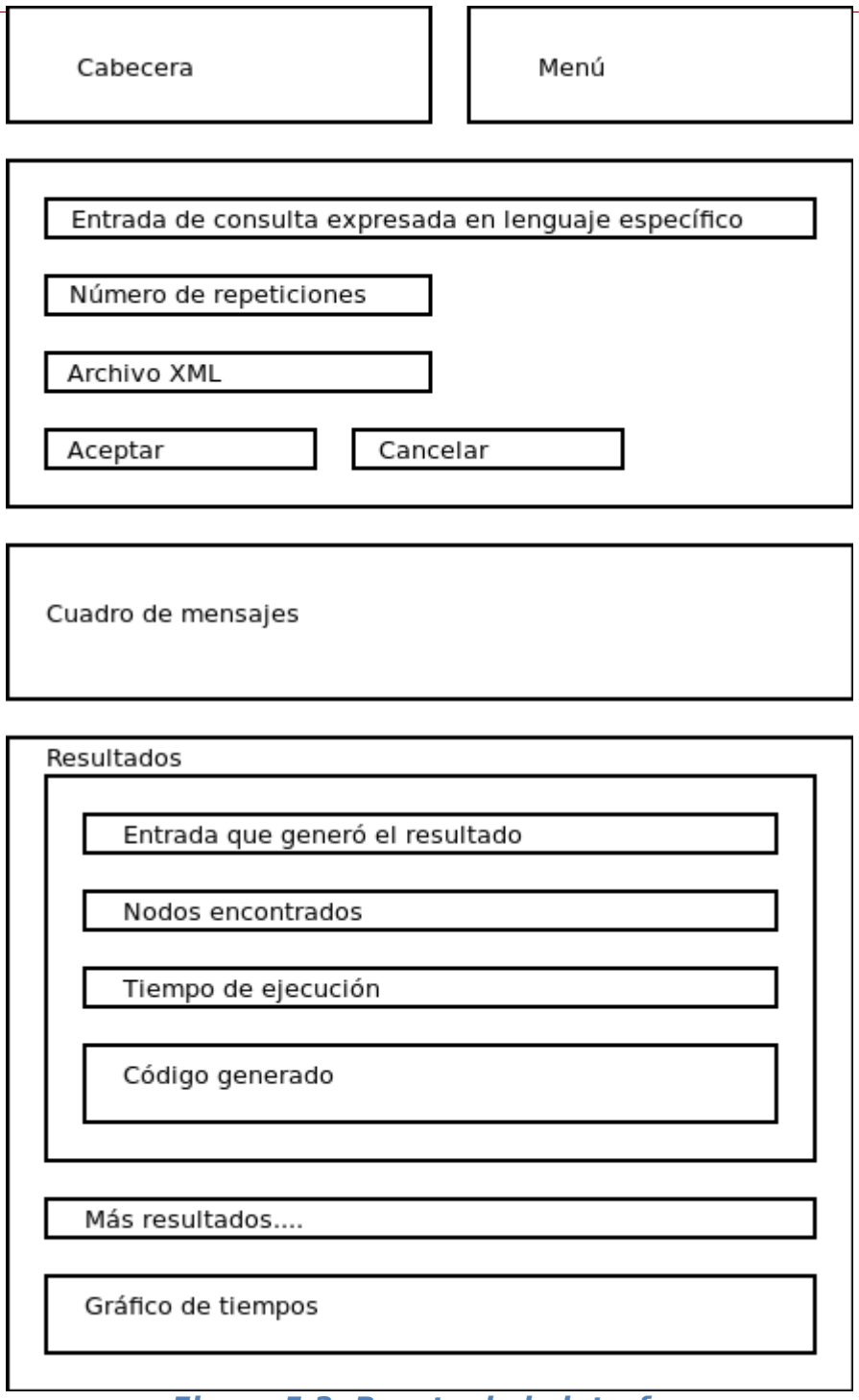


Figura 5.3: Boceto de la interfaz

En la parte superior se encontrarán la cabecera y el menú de la página, que contará sólo con enlaces a la página principal y a la página de ayuda.

Debajo se encontrará el formulario de solicitud, con campos para introducir la consulta a realizar sobre el XML, expresada en el lenguaje específico de modelo, el número de repeticiones que se quiere que hagan los test y el archivo XML.

Por debajo de este se encuentra el cuadro de mensajes, en donde se imprimirán mensajes informando de los eventos relevantes.

Por último, una vez que se obtengan resultados se hará visible el cuadro de resultados, que mostrará para cada resultado la entrada que lo generó, el número de nodos que encontró con esa consulta en el documento XML pasado, el tiempo que le llevó en milisegundos (nótese que será el tiempo de todas las repeticiones) y el código generado.

Al final del todo se mostrará un gráfico de barras con los tiempos de ejecución.

La interfaz estará internacionalizada en castellano e inglés, a excepción de los mensajes de error en el modelo que genera ANTLR y están sólo en inglés y de la página de ayuda que está sólo en castellano.

5.6.2 Descripción del Comportamiento de la Interfaz

La parte más importante de la interfaz serán los scripts que controlarán todo su comportamiento. Al cargar la página los scripts se inicializarán, creando los elementos necesarios en la página.

Cuando el usuario acepte su solicitud los datos del formulario se validarán en el cliente antes de ser enviados vía AJAX al servicio de peticiones. Si son correctos y el servidor retorna un identificador de trabajo, la interfaz comienza automáticamente a consultar al servidor sobre el estado del trabajo cada pocos segundos, informando al usuario mediante el cuadro de mensajes. Cualquier error se informará al usuario por el cuadro de mensajes.

Cuando el servidor retorne el estado “finalizado” se le solicitarán los resultados, y una vez obtenidos se mostrarán como se indicó en el apartado anterior.

5.7 Especificación del Plan de Pruebas

Caso de Uso 1: Solicitud de testeo	
Prueba	**Resultado Esperado**
Petición en la que falten datos o con datos erróneos.	El sistema retorna un error.
Petición correcta.	El sistema retorna un identificador de trabajo.

Caso de Uso 2: Consulta de estado	
Prueba	**Resultado Esperado**
Petición con identificador de trabajo existente.	El sistema retorna un estado que no sea “no existente”.
Petición con identificador de trabajo incorrecto.	El sistema retorna el estado “no existente”.

Caso de Uso 3: Obtención de resultados	
Prueba	**Resultado Esperado**
Petición con identificador de trabajo incorrecto.	El sistema retorna null.
Petición con identificador de trabajo existente pero no finalizado.	El sistema retorna null.
Petición con identificador de trabajo existente y finalizado.	Es sistema retorna los resultados.

Caso de Uso 4: Cancelación de solicitud	
Prueba	**Resultado Esperado**
Petición con identificador de trabajo incorrecto.	No debe ocurrir nada.
Petición con identificador de trabajo existente.	El sistema cancela y elimina el trabajo.

Caso de Uso 5: Ejecución de testeo	
Prueba	**Resultado Esperado**
Petición en la que falten datos o con datos erróneos.	Se muestra un mensaje de error en el panel de mensajes.
Petición correcta.	La petición se procesa y se obtienen los resultados.

Caso de Uso 6: Cancelación de ejecución	
Prueba	**Resultado Esperado**
Cancelar	El trabajo se cancela.

6 Diseño del Sistema

6.1 Arquitectura del Sistema

6.1.1 Diagramas de Paquetes

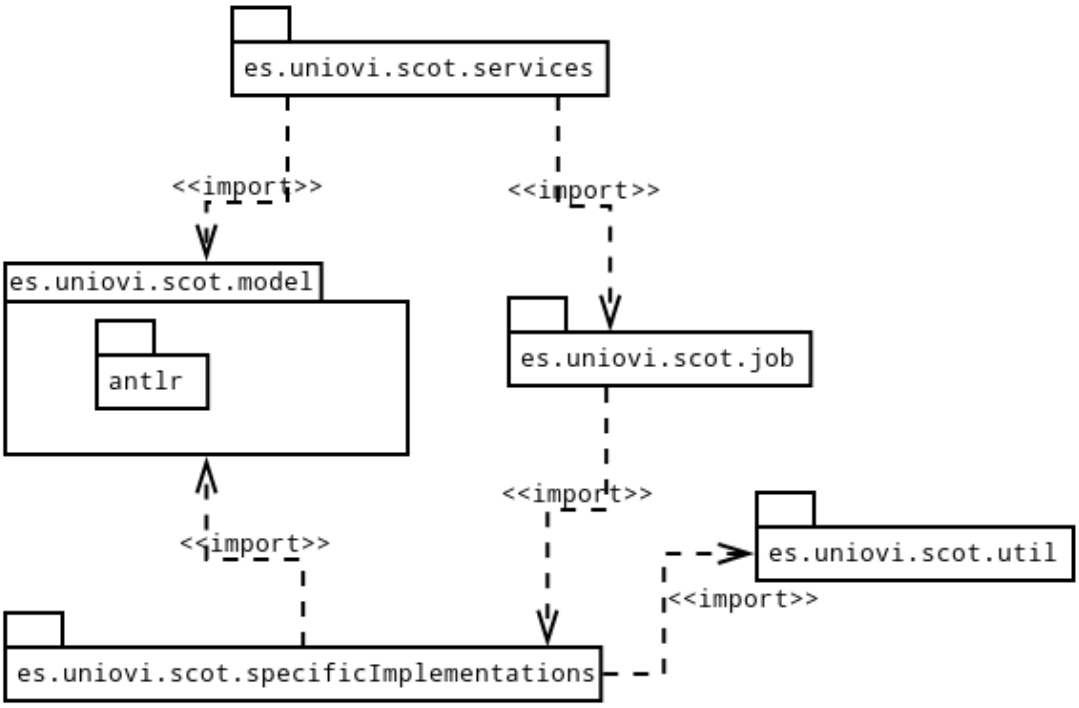


Figura 6.1: Diagrama de paquetes

6.1.1.1 es.uniovi.scot.services

Paquete que contiene la API Web del sistema. Son clases planas de Java (POJOs) marcadas con etiquetas para ser usadas con JAX-RS.

6.1.1.2 es.uniovi.scot.model

Contiene las clases que representan el modelo usado en el sistema y aquellas clases encargadas de obtenerlo.

6.1.1.3 es.uniovi.scot.model.antlr

Contiene las clases generadas por ANTLR a partir de la gramática, usadas para procesar las entradas en el lenguaje específico de modelo.

6.1.1.4 es.uniovi.scot.job

Contiene las clases que representan los trabajos creados a partir de las solicitudes de los clientes y aquellas clases encargadas de gestionarlos y procesarlos.

6.1.1.5 es.uniovi.scot.specificImplementations

Contiene las clases encargadas de transformar el modelo a código, compilarlo, ejecutarlo y obtener los resultados de los test.

6.1.1.6 es.uniovi.scot.util

Contiene una clase de utilidad para estructurar los resultados de la ejecución a devolver al cliente.

6.1.2 Diagrama de Componentes

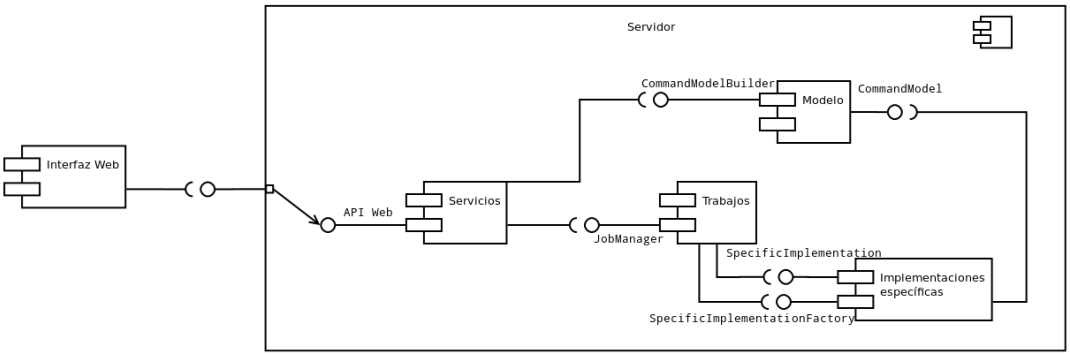


Figura 6.2: Diagrama de componentes

- **Interfaz Web:** La página Web y los scripts del cliente. Depende de los *servicios* Web del *servidor*.
- **Servidor:** El sistema propiamente dicho. Contiene todos los componentes que implementan la funcionalidad del sistema.
- **Servicios:** Los servicios Web RESTful. Sirven de interfaz del sistema con el exterior e implementan funcionalidades menores, como validación de entradas y formateo de salidas. Dependen del *modelo* y del *trabajo* a través de las interfaces *CommandModelBuilder* y *JobManager* respectivamente.
- **Modelo:** Las clases que crean o implementan el modelo.
- **Trabajo:** Las clases que gestionan, ejecutan o implementan trabajos (las peticiones de los clientes). Depende de las *implementaciones específicas* a través de la interfaz *SpecificImplementation* y la clase *SpecificImplementationFactory*.

- **Implementaciones específicas:** Las clases que implementan las transformaciones del modelo a código, su compilación y ejecución y la obtención de resultados. Dependen del *modelo* a través de la interfaz *CommandModel*.

6.1.3 Diagrama de Despliegue

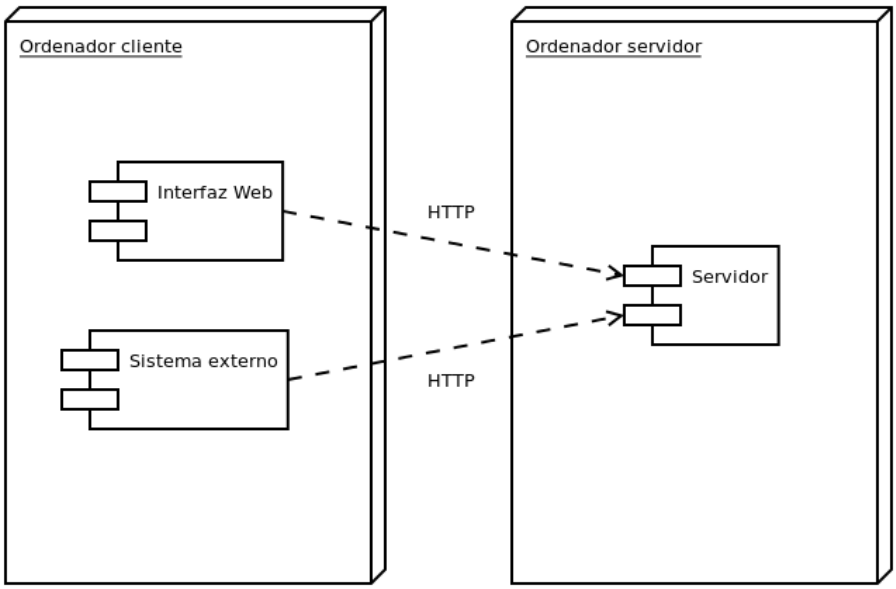


Figura 6.3: Diagrama de despliegue

- **Ordenador cliente:** ordenador en el que se ejecutan las aplicaciones clientes del sistema.
- **Ordenador servidor:** ordenador en el que se ejecuta el servidor del sistema. El servidor se ejecutará en un contenedor de aplicaciones.
- **Interfaz Web:** interfaz implementada en este proyecto. Se ejecutará en un navegador Web. Realiza peticiones al servidor vía mensajes HTTP realizados mediante AJAX.
- **Sistema externo:** aplicación externa al sistema implementado a este proyecto, pero que es cliente de él. Realiza peticiones al servidor vía mensajes HTTP.
- **Servidor:** sistema implementado en este proyecto. Ofrece como API Web una serie de servicios Web RESTful.

Todos los datos complejos de entrada y salida de los servicios están en formato JSON, excepto la entrada del servicio de peticiones, que es un mensaje multiparte.

6.2 Diseño de Clases

6.2.1 Diagrama de Clases

En el siguiente diagrama no se muestran las clases generadas por ANTLR a partir de la gramática, si bien algunas de ellas son referenciadas por clases mostradas.

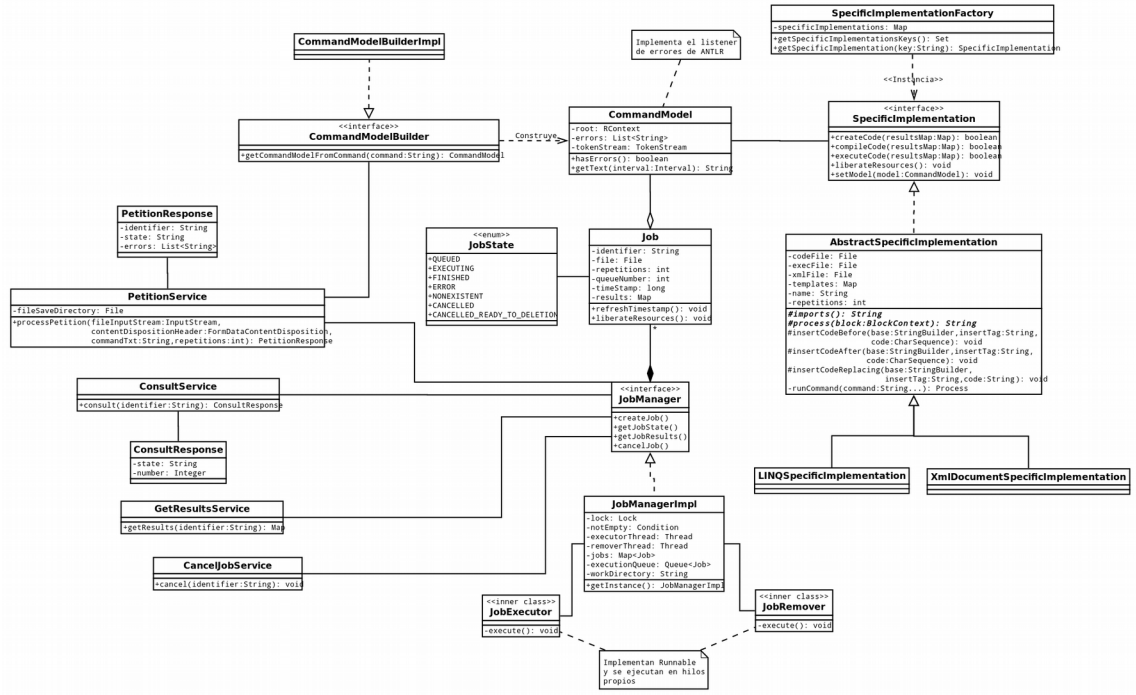


Figura 6.4: Diagrama de clases

6.3 Diagramas de Interacción y Estados

6.3.1 Caso de Uso 1: Solicitud de testeo

6.3.1.1 Diagramas de Interacción (Comunicación y Secuencia)

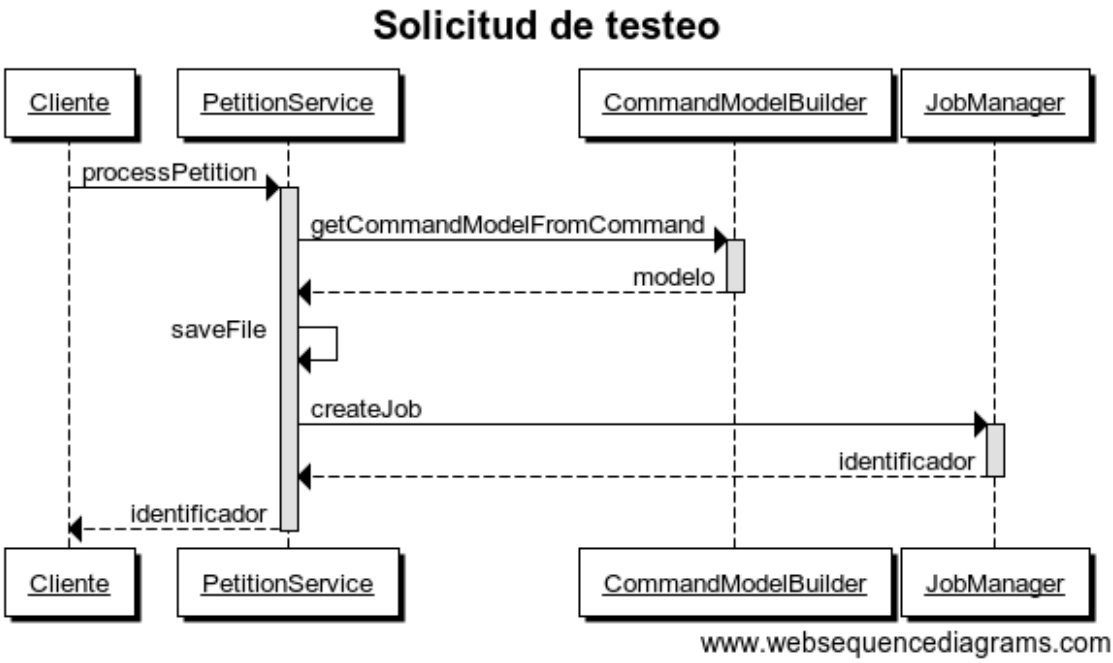


Figura 6.5: Diagrama de secuencia para el caso de uso Solicitud de testeo

En este caso de uso se reciben los datos de la petición del cliente. A partir del comando (la consulta en lenguaje específico) se obtiene el modelo, el archivo XML es guardado en disco, y si todos los datos son correctos se crea un trabajo con ellos, se le da un identificador único y se pone en la cola de ejecución. El identificador es retornado hasta el cliente.

6.3.2 Caso de Uso 2: Consulta de estado

6.3.2.1 Diagramas de Interacción (Comunicación y Secuencia)

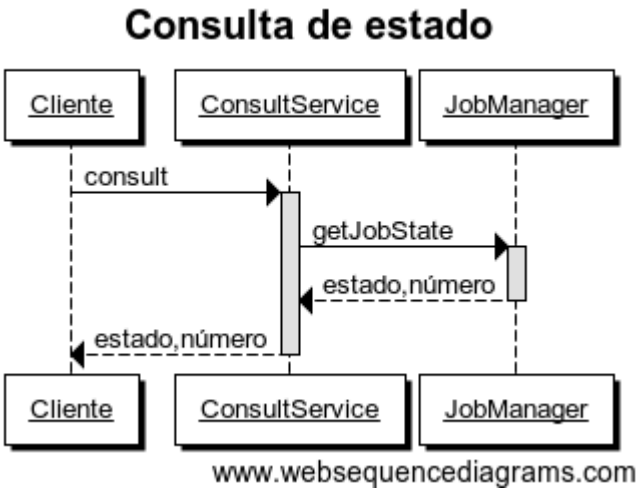


Figura 6.6: Diagrama de secuencia para el caso de uso Consulta de estado

En este caso de uso se recibe el identificador de un trabajo del cliente. Se obtiene del manejador de trabajos el estado del trabajo asociado al identificador (si no hay ninguno el estado será “no existente”) y en caso de que el estado sea encolado se obtendrá también el número en cola. Ambos datos son devueltos en un mapa. Los valores son transmitidos al cliente.

6.3.3 Caso de Uso 3: Obtención de resultados

6.3.3.1 Diagramas de Interacción (Comunicación y Secuencia)

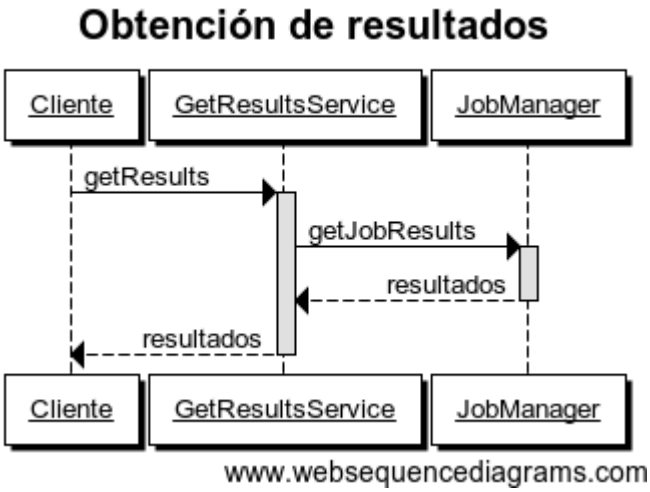


Figura 6.7: Diagrama de secuencia para el caso de uso Obtención de resultados

En este caso de uso se recibe el identificador de un trabajo finalizado del cliente. Se obtienen los resultados de la ejecución del manejador de trabajos y se devuelven al cliente.

6.3.4 Caso de Uso 4: Cancelación de testeo

6.3.4.1 Diagramas de Interacción (Comunicación y Secuencia)

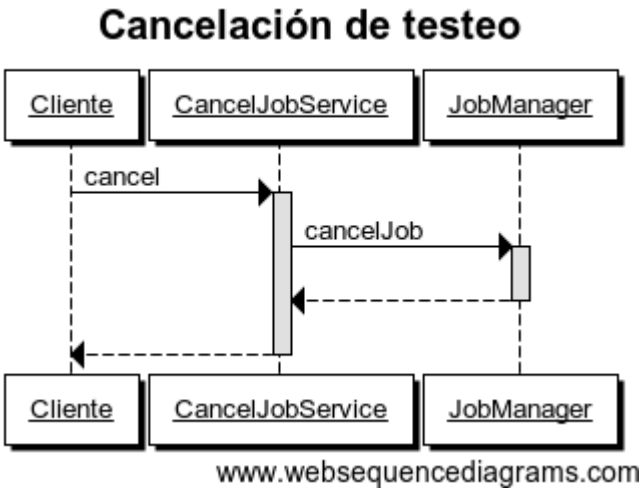


Figura 6.8: Diagrama de secuencia para el caso de uso Cancelación de testeo

En este caso de uso se recibe el identificador de un trabajo del cliente. El trabajo es eliminado en el momento y sus recursos liberados, a menos que esté en ejecución, en cuyo caso se marca su estado como cancelado para liberar los recursos y eliminarlo cuando sea posible.

6.3.5 Caso de Uso 5: Ejecución de testeo

6.3.5.1 Diagramas de Interacción (Comunicación y Secuencia)

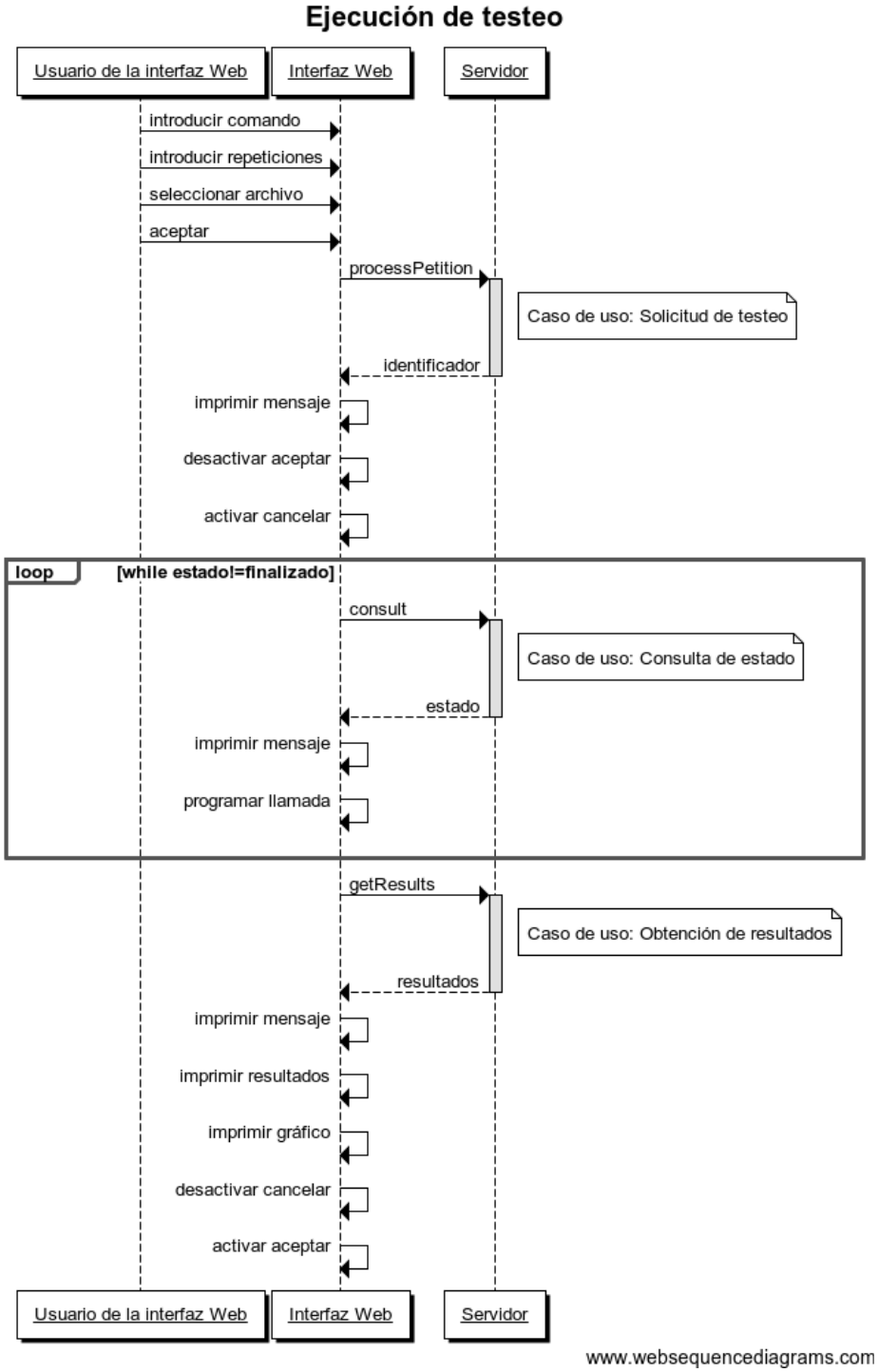


Figura 6.9: Diagrama de secuencia para el caso de uso Ejecución de testeo

En este caso de uso el usuario introduce en el formulario de la interfaz los campos (comando con la consulta en lenguaje específico, número de repeticiones de los test y el archivo XML) y pulsa el botón de aceptación.

La interfaz envía los datos al servicio de petición del servidor iniciando el caso de uso **Solicitud de consulta**. Cuando recibe el visto bueno con el identificador de trabajo del servidor, la interfaz informa al usuario y desactiva el botón de aceptación del formulario y activa el de cancelación.

Comienza entonces una consulta de estado repetitiva, enviando el identificador de trabajo al servicio de consulta del servidor iniciando el caso de uso **Consulta de estado**. La interfaz mantiene al usuario informado del estado vía mensajes. El bucle se rompe cuando se recibe el estado de finalizado.

Entonces la interfaz envía el identificador al servicio de obtención de resultado del servidor iniciando el caso de uso **Obtención de resultados**. Cuando recibe los resultados imprime un mensaje para el usurario, imprime los resultados y agrega un gráfico de barras con los tiempos de los test. También desactiva el botón de cancelación y activa el de aceptación.

6.3.6 Caso de Uso 6: Cancelación de ejecución

6.3.6.1 Diagramas de Interacción (Comunicación y Secuencia)

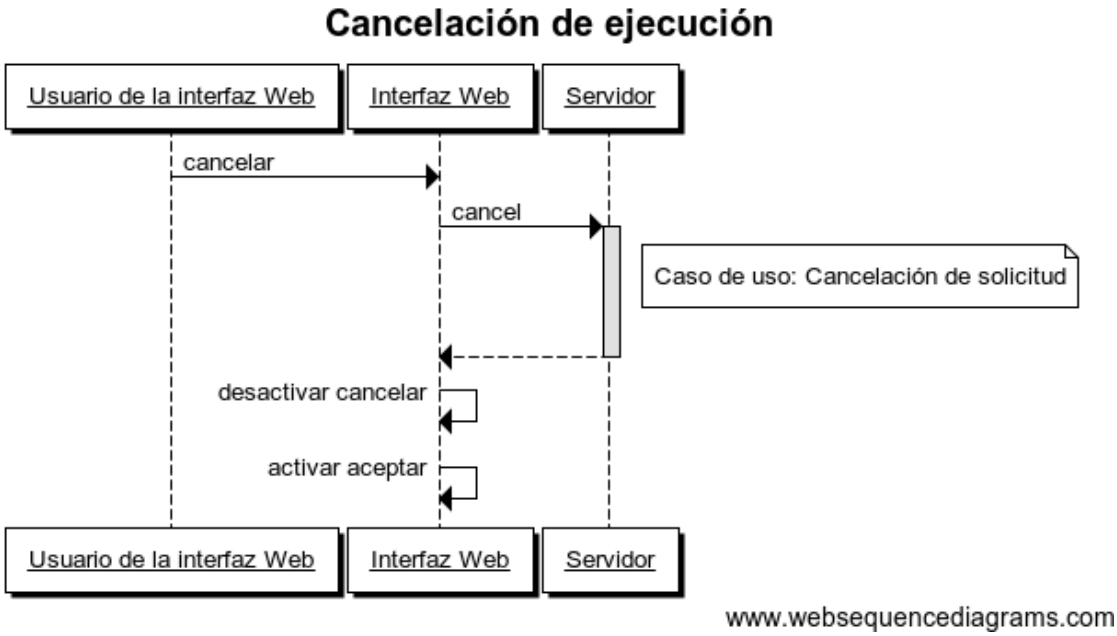


Figura 6.10: Diagrama de secuencia para el caso de uso Cancelación de ejecución

En este caso de uso el usuario pulsa el botón de cancelación. La interfaz envía el identificador de trabajo al servicio de cancelación del servidor,

iniciando el caso de uso **Cancelación de solicitud**. Acto seguido desactiva el botón de cancelación y activa el de aceptación.

6.3.7 Caso de uso 7: Ejecución de trabajo

6.3.7.1 Diagramas de Interacción (Comunicación y Secuencia)

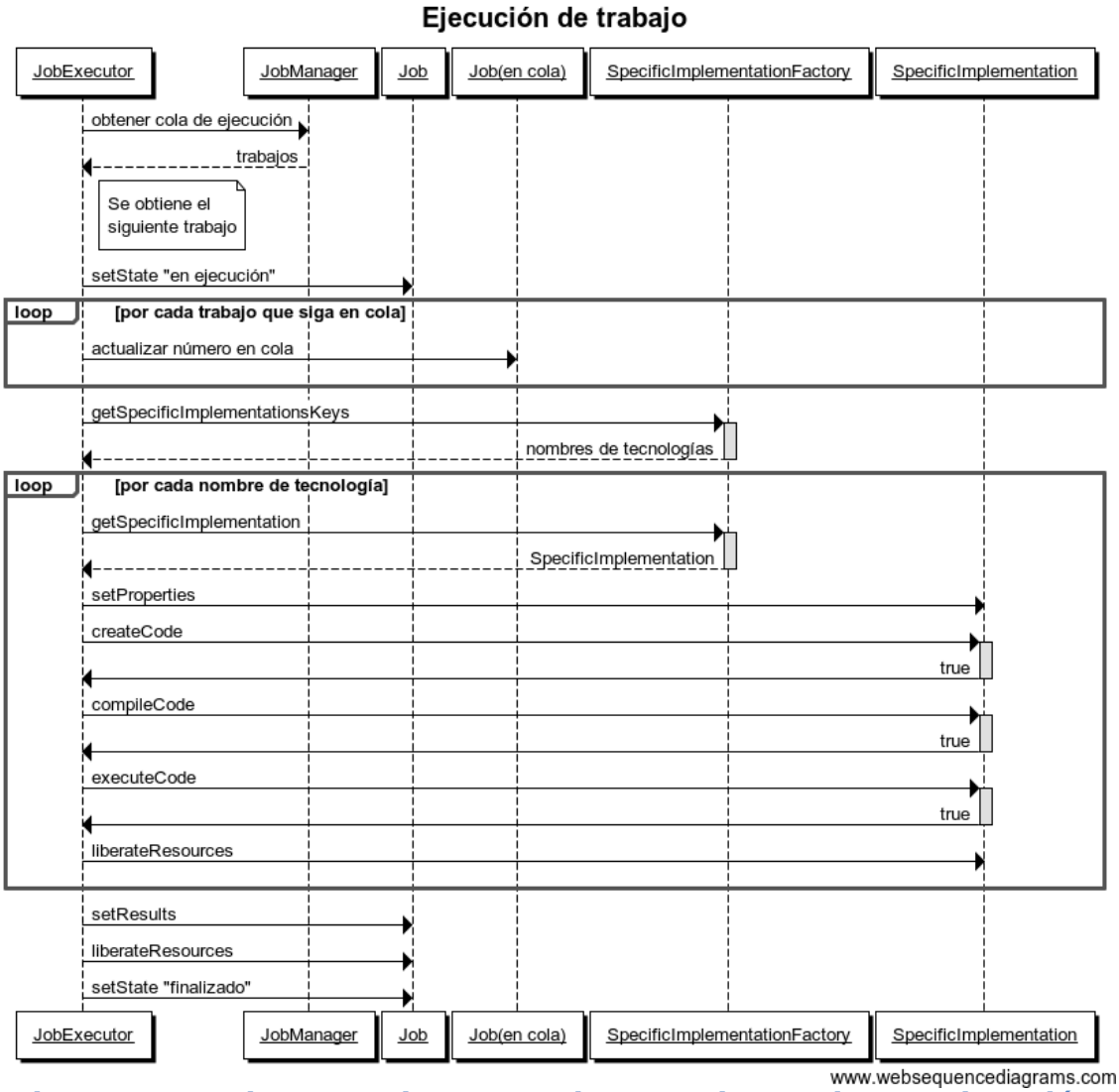


Figura 6.11: Diagrama de secuencia para el caso de uso Ejecución de trabajo

En este caso de uso el sistema ejecuta un trabajo creado por un cliente.

La ejecución consiste en la creación, compilación y ejecución del código de test obteniendo los resultados, todo ello para cada tecnología implementada.

Los recursos de los ficheros creados en la ejecución y del trabajo son liberados cuando dejan de ser necesarios.

6.3.8 Caso de uso 8: Eliminación de trabajos pasados de tiempo

6.3.8.1 Diagramas de Interacción (Comunicación y Secuencia)

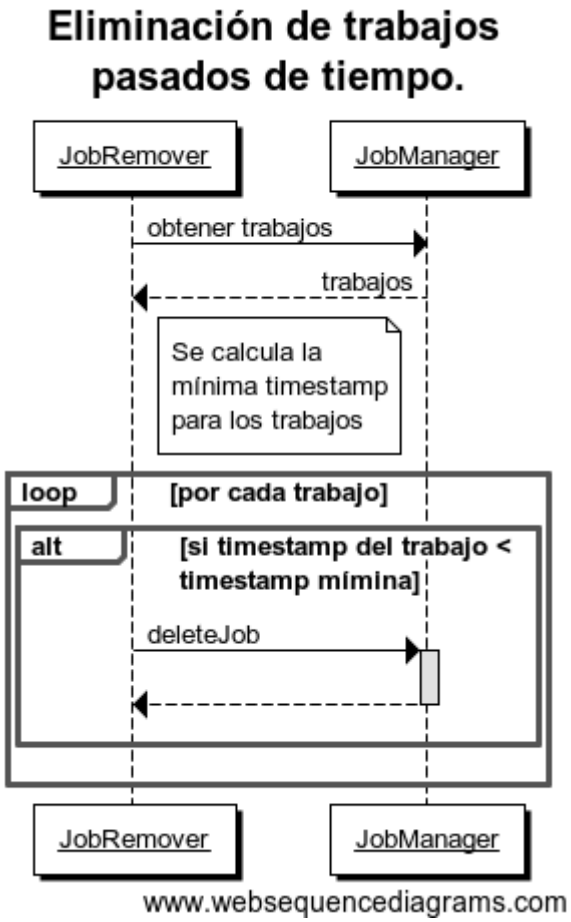


Figura 6.12: Diagrama de secuencia para el caso de uso Eliminación de trabajos pasados de tiempo

En este caso de uso el sistema libera los recursos de los trabajos que llevan demasiado tiempo sin que el cliente consulte su estado, y que por lo tanto se consideran abandonados.

6.3.9 Diagrama de estados de los trabajos

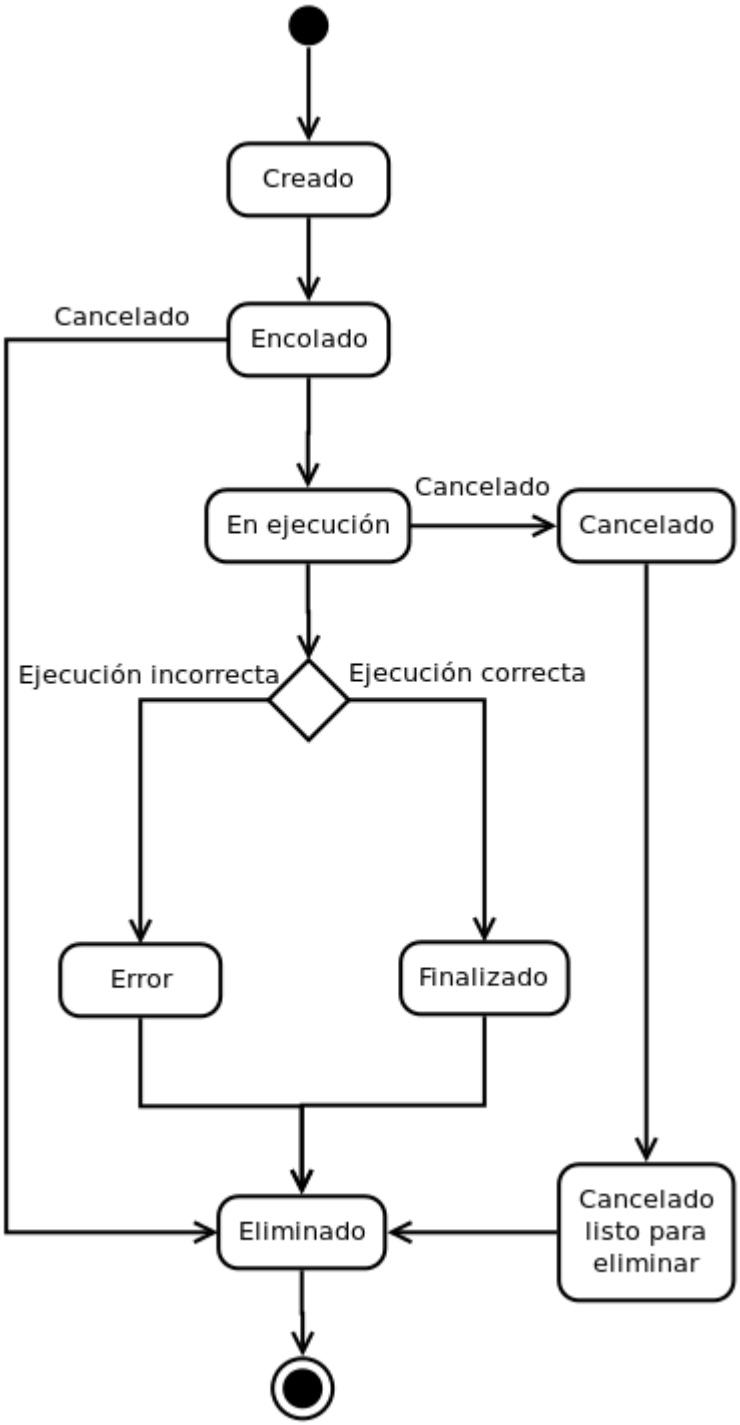


Figura 6.13: Diagrama de estados de los trabajos

Este diagrama muestra la evolución del estado de un trabajo a lo largo de su ciclo de vida.

En cuanto es creado el trabajo es agregado a la cola de ejecución, donde espera que llegue su turno para ser ejecutado.

Una vez que la ejecución termina el estado del trabajo dependerá de si la ejecución arrojó algún resultado, en cuyo caso será “finalizado”, o no, en cuyo caso será “error”.

Si el trabajo es cancelado (o si se pasa el tiempo de sesión) el trabajo es directamente eliminado, con la excepción de que se encuentre en ejecución, en cuyo caso se marcará su estado como “cancelado”. El motivo de no borrarlo directamente en este caso es que los test, que se ejecutan de manera externa al sistema, pueden tener bloqueados los ficheros que se deben borrar al eliminar el trabajo. Cuando los test se cierran y los ficheros quedan libres se marca el trabajo con el estado “cancelado listo para eliminar”, indicando así que sus recursos ya pueden ser liberados.

6.4 Diagramas de Actividades

6.4.1 Caso de uso 7: Ejecución de trabajo

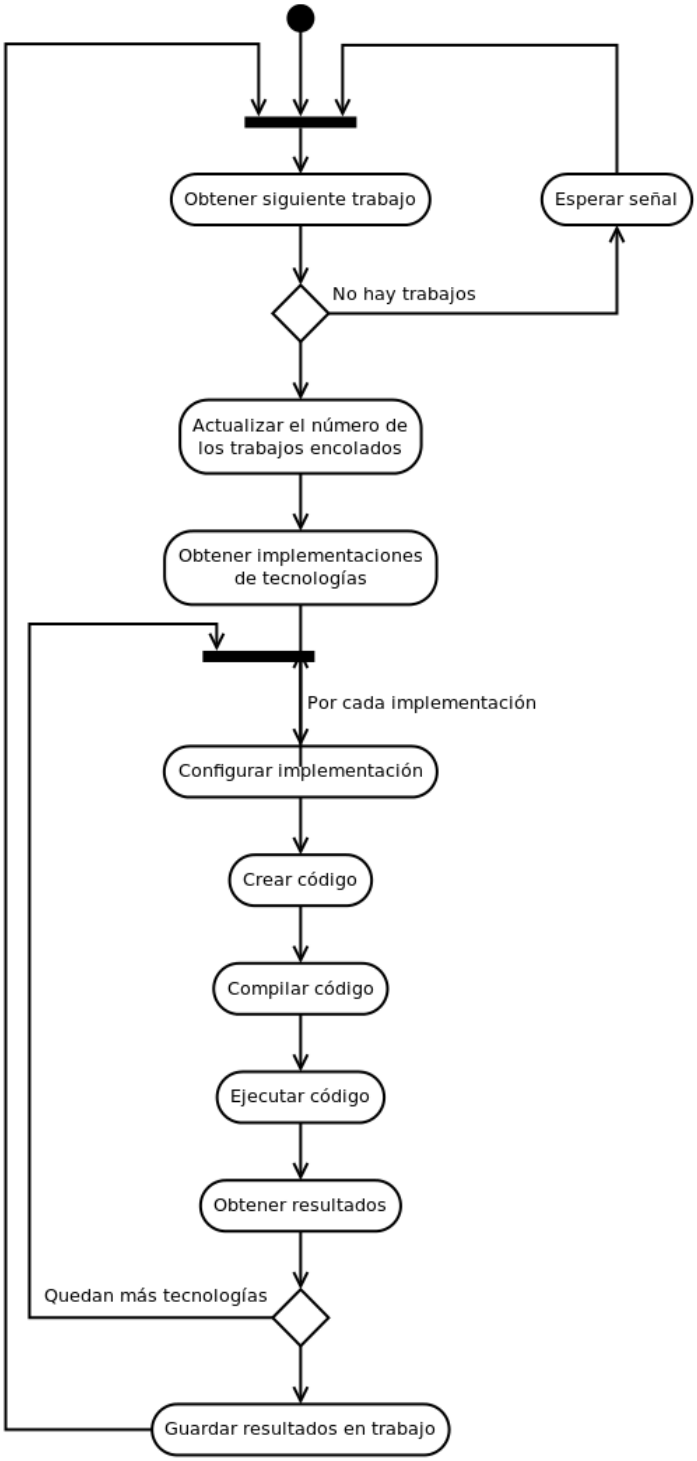


Figura 6.14: Diagrama de actividades del caso de uso Ejecución de trabajo

En este diagrama se muestran las actividades implicadas en la ejecución de un trabajo.

El proceso comienza con la obtención del trabajo de la cola de ejecución. En caso de que la cola esté vacía, el hilo de ejecución se queda en espera hasta que recibe la señal de que se agregó un nuevo trabajo. Cada vez que se extrae un trabajo es necesario actualizar el número de la posición en la cola de cada trabajo que continúe en ella.

Una vez se tiene el trabajo se obtienen los objetos encargados de implementar los test en cada tecnología a probar. Será necesario uno de ellos para cada tecnología.

Se configuran estos objetos pasándoles la información del trabajo (modelo, archivo y número de repeticiones) y el directorio a utilizar para crear los test. Una vez configurados ya se les puede solicitar que creen, compilen y ejecuten el código, obteniendo en el proceso los resultados de los test. Este proceso se repite mientras queden tecnologías por probar.

Una vez que se termine con todas las tecnologías se guardan los resultados en el trabajo y se comienza el proceso de nuevo, en un bucle que dura hasta que el sistema es cerrado.

En el esquema no se representa la liberación de recursos, que tiene lugar cuando los archivos temporales de los trabajos o de los test ya no son necesarios.

6.5 Diseño de la Interfaz

6.5.1 Interfaz gráfica Web

El aspecto final de la interfaz es el que sigue.

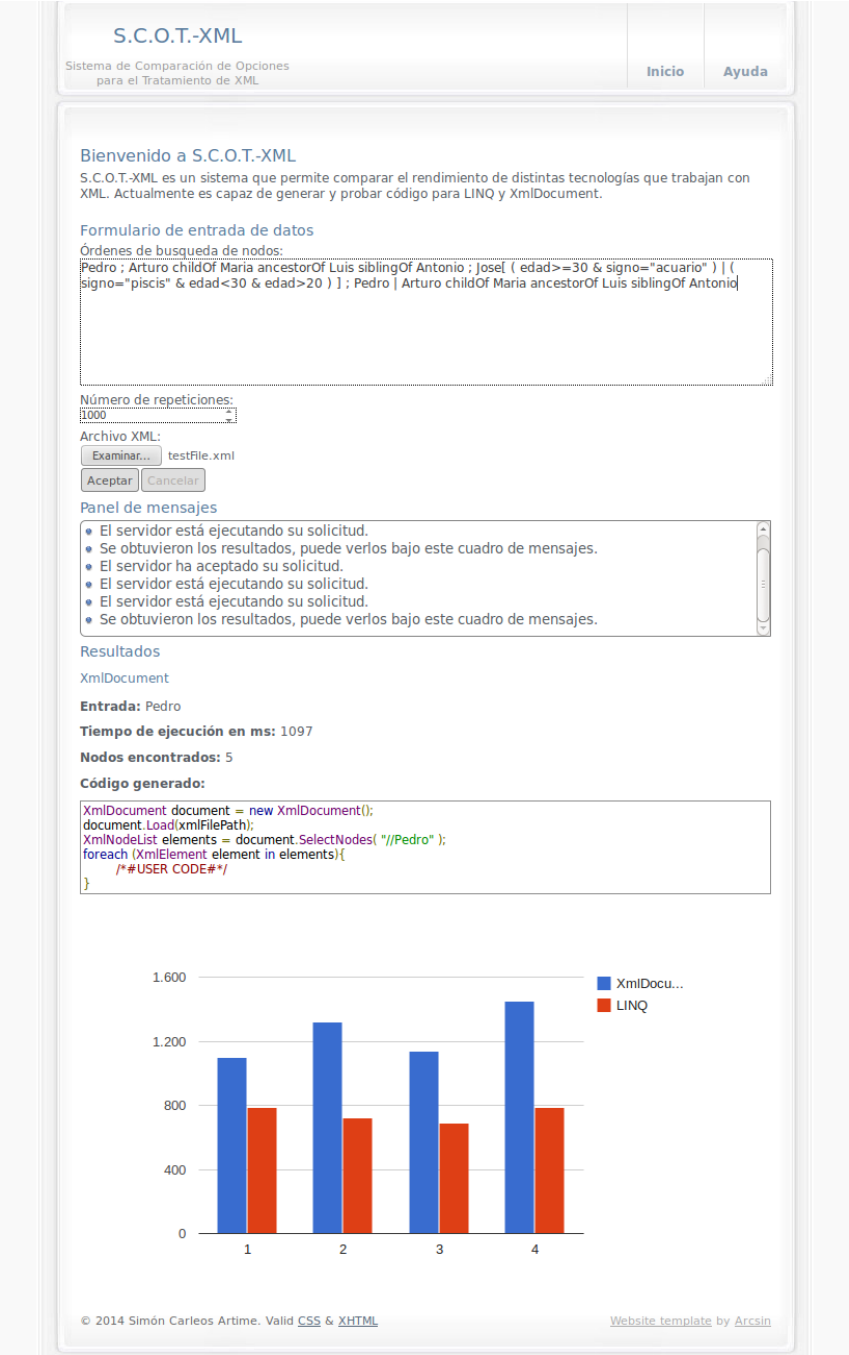


Figura 6.15: Aspecto final que tendrá la interfaz

Contiene los elementos ya explicados en el análisis.

6.5.2 Interfaz de servicios Web

El sistema cuenta con una interfaz de servicios RESTful con los métodos citados a continuación. Todos ellos aceptan peticiones POST.

Path	**Descripción**
services/petition	Servicio para las solicitudes de testeo.
Entradas	
file	El archivo XML sobre el que se ejecutarán los test.
orders	Una cadena con la consulta (o consultas) a testear expresadas en el lenguaje específico.
repetitions	Un entero mayor que cero que indica el número de repeticiones para los test.
Salida	
Si la petición es correcta, un objeto JSON que contiene el estado “correcto” y un identificador de trabajo. Si la petición es incorrecta contiene el estado “error” y uno o varios textos de error.	
Notas	
Las entradas deben ir en un mensaje multiparte.	

Path	**Descripción**
services/consult	Servicio para consultar el estado de un trabajo.
Entradas	
identifier	El identificador del trabajo.
Salida	
Un objeto JSON con el estado del trabajo, y si el estado es “en ejecución” un número entero mayor de 0 que indica la posición en la cola.	
Notas	
-	

Path	Descripción
services/results	Servicio para obtener los resultados de un trabajo.
Entradas	
identifier	El identificador del trabajo.
Salida	
Si se dispone de los resultados, un objeto JSON con los resultados organizados por tecnologías. Si no, null.	
Notas	
-	

Path	Descripción
services/cancel	Servicio para las solicitudes de testeo.
Entradas	
identifier	El identificador del trabajo.
Salida	
Si la petición es correcta, un objeto JSON que contiene el estado “correcto” y un identificador de trabajo. Si la petición es incorrecta contiene el estado “error” y uno o varios textos de error.	
Notas	
-	

6.6 Diseño del lenguaje específico del dominio y modelo

Los elementos que queremos implementar son los siguientes.

Elemento	Nodo
Descripción	Un nodo (elemento) del documento XML seleccionado por el nombre.
Esquema	<nombre del nodo>
Notas	-
Ejemplo	autor

Elemento	Relación
Descripción	Una relación entre dos nodos, como puede ser “hijo”, “descendiente”, etc.
Esquema	<nodo> <relación> <nodo>
Notas	Las relaciones se pueden encadenar. Los nodos que se seleccionan en la búsqueda son los que encajan con el primer nodo en la cadena de relaciones. A una cadena de nodos y relaciones la llamaremos consulta. Las relaciones son: **ancestorOf**, **parentOf**, **siblingOf**, **childOf** y **descendantOf**.
Ejemplo	autor **chidOf** libro **descendantOf** biblioteca (se buscarían los nodos “autor” hijos de nodos “libro” que a su vez desciendan de nodos “biblioteca”).

Elemento	Unión
Descripción	La unión de dos consultas, para buscar nodos que encajen en cualquiera de ellas.
Esquema	<consulta>|<consulta>
Notas	Las uniones se pueden encadenar
Ejemplo	autor **childOf** libro **|** autor **childOf** cuadro **|** autor **childOf** película

Elemento	Múltiples consultas
Descripción	Capacidad para poder incluir varias consultas separadas en un solo test.
Esquema	<consulta>**;**<consulta>
Notas	-
Ejemplo	autor **childOf** libro **;** autor **childOf** cuadro

Elemento	Filtrado por atributos
Descripción	Capacidad para filtrar los nodos por los valores de sus atributos.
Esquema	<nodo>**[**<filtro>**]**
Notas	Se puede filtrar por atributos textuales y numéricos. Los atributos textuales se pueden seleccionar por igualdad (**=**) o diferencia (**!=**). Los atributos textuales se pueden seleccionar por igualdad (**=**), diferencia (**!=**), mayor magnitud (**>**), menor magnitud (**<**), mayor o igual magnitud (**>=**) o menor o igual magnitud (**<=**). Los filtros se pueden combinar con los operandos lógicos OR (**|**) y AND (**&**). Se pueden usar paréntesis para agrupar los filtros.
Ejemplo	autor**[** nombre**!=”**Alejandro**”** **|** **(** edad**>**30 **&** apellido**=”**Canales**”)]** (buscaría los autores con nombre diferente a “Alejandro” o que tengan edad mayor de treinta y apellido igual a “Canales”)

El modelo usado en el sistema es el propio árbol sintáctico, por lo que la gramática se diseñó para tener una estructura útil para nuestros propósitos. La estructura es la que sigue.

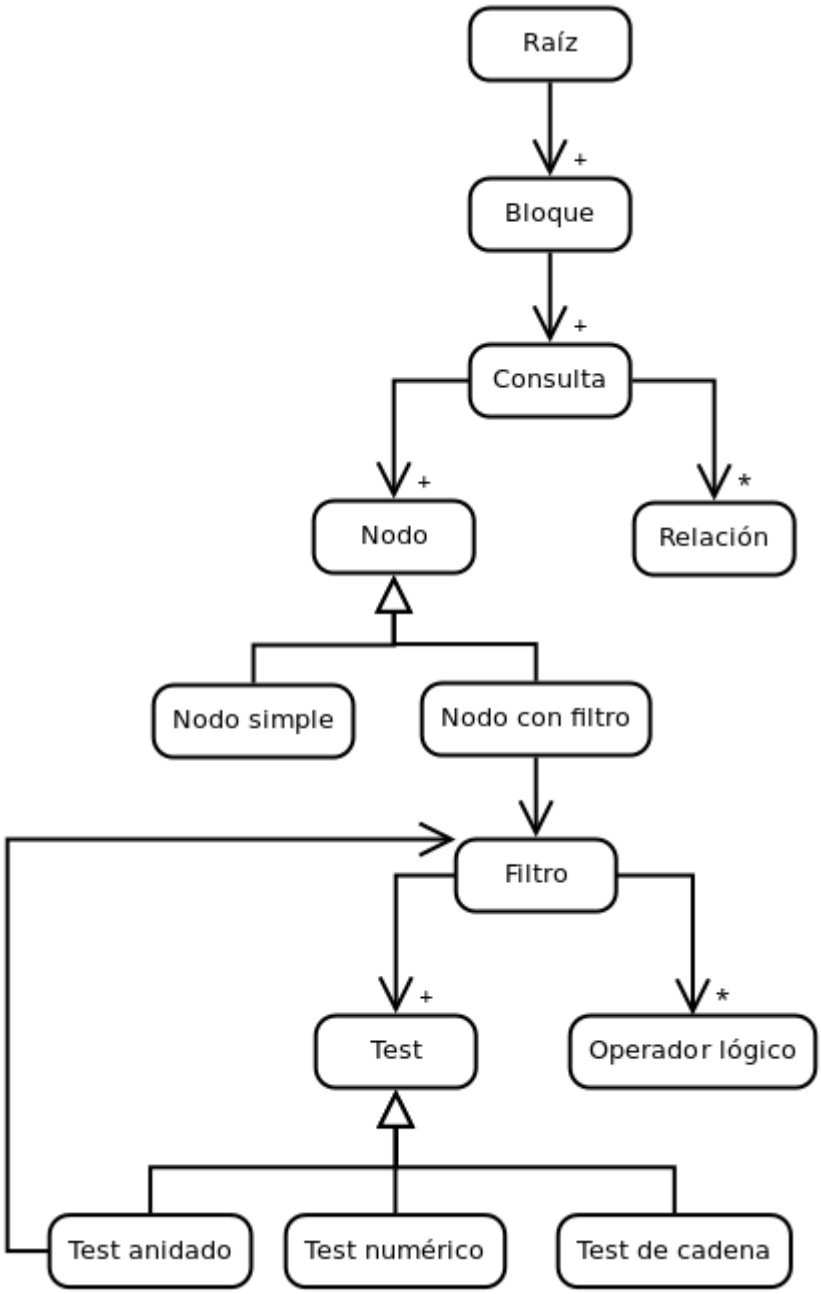


Figura 6.16: Esquema del modelo del DSL

Lo que llamamos “bloque” puede ser una consulta o una unión de consultas (los bloques están separados por '**;**'), y es lo que constituirá un test individual en el código generado.

El elemento “test” se refiere a la prueba hecha contra un atributo (por ejemplo apellido!=”Zapico”).

Un test anidado es el resultado de usar paréntesis en el filtrado por atributos.

6.7 Diseño de la transformación del modelo a código

El paso de modelo a código se hace recorriendo el árbol semántico e insertando el código que se va generando en una plantilla base, usando para ello etiquetas incrustadas en la plantilla como marcas de los lugares de inserción. En las inserciones pueden ir incluidas nuevas etiquetas para pasos posteriores.

El reto de la conversión está en lograr que se pueda transformar cualquier combinación de búsqueda a código compilable y ejecutable.

6.7.1 XmlDocument

En la implementación de los test para XmlDocument se usará para las búsquedas el método “SelectNodes”, que recibe como entrada una expresión XPath. El grueso de la generación será por tanto obtener dicha expresión. Esto hace que esta implementación sea fácilmente adaptable a otras tecnologías que funcionen con XPath.

La forma más directa de convertir las relaciones entre nodos es usar los ejes de XPath (por ejemplo “parentOf” pasaría a expresarse como “/parent::”). La excepción es la relación de hermanos, que a falta de un eje propio se usa “/../child::” (hijo del padre).

La unión se hará con el propio operador '|' .

El filtrado se hará de manera similar a nuestro propio lenguaje, pero traduciendo los operadores que sean necesarios (por ejemplo '&' pasa a ser 'and') y añadiendo una “@” antes del nombre de los atributos.

6.7.2 LINQ

En la implementación de los test para LINQ se usarán varios de los métodos proporcionados.

En la búsqueda más simple (un nodo sin relaciones ni filtrado por atributos) se usará el método “Descendants” del documento. Esto devolverá los nodos que encajen con el nombre pasado. En caso de búsquedas con más requisitos se usará el método “Where” para filtrar los resultados.

Las relaciones se modelan cada una con un método relacionado proporcionado por LINQ (por ejemplo “ancestorOf” se implementa con “Descendants”, ya que hay que poner las relaciones a la inversa). Estos métodos retornan colecciones en las que habrá que comprobar si algún elemento cumple con las especificaciones dadas, para lo que se usa el

método “Any”, que necesita el uso de un identificador que habrá que generar y mantener en una variable para posteriores usos.

La excepción son las relaciones de hijo y hermano (expresado como hijo del padre). En estos casos se debe obtener el padre y hacer las comprobaciones directamente “a mano”.

No hay que olvidarse, tanto en estos casos como en otros, de comprobar que los elementos no sean null antes de trabajar con ellos, para evitar excepciones.

LINQ cuenta con un método para las uniones, “Union”, que deberá llamarse sobre una de las queries a unir e insertar la otra query como argumento.

Por último el filtrado por atributos debe hacerse comprobando que no sean null y realizando las operaciones necesarias con Java.

6.7.3 XmlReader

Aunque en este proyecto se pretendía dar implementación para las tres tecnologías citadas, se tuvo que tomar la decisión de dejar la implementación de XmlReader como posible futura ampliación.

La razón es que, al funcionar XmlReader a tan bajo nivel, se hace muy difícil crear todas las estructuras de código que serían necesarias para crear un programa que haga lo que queremos.

Si bien no sería demasiado difícil implementar una búsqueda concreta, hacerlo para un modelo con tantas combinaciones resulta demasiado complicado.

Se intentaron pensar soluciones para el problema (uso de pilas, máquinas de estados, etc.), pero seguían siendo demasiado complicadas y no aseguraban el correcto funcionamiento (las máquinas de estados, por ejemplo, aseguraban que los nodos encontrados fuesen correctos, pero no que se encontrasen todos los nodos).

Así, se consideró que la tarea de implementar una transformación del modelo a código que use XmlReader es demasiado compleja para el presente proyecto, llegando a la decisión de dejarlo fuera de este.

6.8 Especificación Técnica del Plan de Pruebas

Las pruebas se realizarán sobre el mismo ordenador de implementación: un ordenador portátil Acer Aspire 5733Z con sistema operativo Ubuntu 12.04. El contenedor de aplicaciones utilizado es un Apache Tomcat 7.0.53.

Siendo la parte gráfica de la interfaz (la página propiamente dicha) simple y con poca variedad de elementos no se considera necesario realizar pruebas de accesibilidad. Además, la mayor parte de la interfaz estará implementada en ECMAScript y será imprescindible que esté activado para su funcionamiento, lo que ya se considera un problema de accesibilidad que no podemos salvar.

6.8.1 Pruebas de Integración y del Sistema

Las pruebas del sistema se harán a través de la interfaz gráfica, puesto que esta hace uso de la totalidad de la API Web y de toda la funcionalidad del sistema, con lo que la correcta ejecución de estas pruebas aseguran con bastante fiabilidad que el sistema funciona correctamente.

Las pruebas serán las especificadas en el análisis para los casos de uso de **Ejecución de testeo** y **Cancelación de ejecución**, que básicamente consisten en realizar peticiones y cancelar peticiones.

Para probar la correcta implementación de los test se ejecutarán diferentes consultas sobre un documento XML de prueba predeterminado y se comprobará que el número de nodos encontrados es el esperado para la consulta.

Lo que haremos será definir una serie de consultas y esquemas de elementos XML que las satisfagan. Estos esquemas los incluiremos en un documento XML sabiendo así de antemano el número de nodos que debe encontrar cada consulta.

Para las pruebas usaremos un documento XML cualquiera (el propio pom.xml del proyecto) en el que insertaremos los elementos aquí definidos en diferentes partes y niveles.

Consulta	
Pedro	
XML que debería encajar	**Nº apariciones**
<Pedro></Pedro>	3
<Pedro/>	2

Nº de apariciones indica el número de vez que se da la estructura en el fichero de prueba.

Consulta	
Arturo **childOf** Maria **ancestorOf** Luis **siblingOf** Antonio	
XML que debería encajar	**Nº apariciones**
<Maria> <Arturo> <Luis></Luis> <Antonio></Antonio> </Arturo> </Maria>	2
<Maria> <Arturo></Arturo> <relleno> <relleno> <Luis></Luis> <Antonio></Antonio> </relleno> </relleno> </Maria>	3

Consulta	
Jose**[(** edad**>=**30 **&** signo**=”**acuario**”)	(** signo**=”**piscis**” &** edad**<**30 **&** edad**>**20 **)]**	
XML que debería encajar	**Nº apariciones**
<Jose signo=”acuario” edad=30 />	1
<Jose signo=”acuario” edad=40 />	2
<Jose signo=”piscis” edad=25 />	2
XML que no debería encajar	**Nº apariciones**
<Jose signo=”acuario” edad=”29” />	1
<Jose signo=”acuario” />	1
<Jose signo=”piscis” edad=”30” />	1
<Jose signo=”piscis” edad=”20” />	1
<Jose />	1

Consulta	
Pedro **|** Arturo **childOf** Maria **ancestorOf** Luis **siblingOf** Antonio	
XML que debería encajar	**Nº apariciones**
Los que encajan en las dos primeras consultas	10

La consulta completa para las pruebas será la siguiente.

Pedro **;**

Arturo **childOf** Maria **ancestorOf** Luis **siblingOf** Antonio**;**

Jose**[(** edad**>=**30 **&** signo**=”**acuario**”) | (** signo**=”**piscis**” &** edad**<**30 **&** -edad**>**20 **)] ;**

Pedro **|** Arturo **childOf** Maria **ancestorOf** Luis **siblingOf** Antonio

[NOTA: si se copia y se pega del documento habrá que cambiar las comillas]

Sabemos que tenemos que obtener cinco resultados en cada una excepto en la unión, en la que tendremos que obtener diez resultados.

6.8.2 Pruebas de Usabilidad

Para comprobar que la aplicación es usable se realizarán actividades guiadas con usuarios.

El sistema está orientado a usuarios desarrolladores, por lo que buscaremos voluntarios lo más próximo posible a este perfil.

La actividad a llevar a cabo será realizar varias consultas a partir de una descripción textual . Usaremos las mismas consultas del apartado anterior.

Nº de prueba	Enunciado
1	Buscar todos los nodos “Pedro”.
2	Buscar todos los nodos “Arturo” que sean hijos de un nodo “Maria” que a su vez tenga como descendientes a los nodos hermanos “Luis” y “Antonio”.
3	Buscar los nodos “Jose” cuyos atributos cumplan alguna de las siguientes premisas. • Su “signo” es “acuario” y su “edad” mayor o igual a 30. • Su “signo” es “piscis” y su “edad” está entre 20 y 30, no incluidos.
4	Buscar la unión de las dos primeras consultas.

Para cada prueba se anotarán el número de intentos necesarios para su consecución y los hechos relevantes.

También se les pedirá a los voluntarios que rellenen un formulario con las siguientes preguntas. La respuesta será siempre un número entero entre 0 y 5, siendo el 0 la opinión más negativa y el 5 la más positiva.

¿Le gusta el aspecto de la interfaz?
¿Considera que es fácil de usar?
¿Considera que el DSL es fácil de entender?
¿Considera que el DSL es completo?
¿Le gusta el código generado?
¿Considera que el sistema es útil?

7 Implementación del Sistema

7.1 Lenguajes de Programación

7.1.1 Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun desde 1991 y lanzado por primera vez en 1995. La mayor parte de su sintaxis deriva de C y C++, pero con menos opciones para trabajar a bajo nivel.

Java es multiplataforma ya que es compilado en un bytecode que puede ser ejecutado en cualquier máquina virtual de Java (JVM) sin importar la arquitectura en que se ejecute. Permite que los desarrolladores escriban un programa y lo puedan ejecutar en cualquier sistema con una máquina virtual Java sin modificarlo.

Es uno de los lenguajes de programación más populares, con millones de usuarios por todo el mundo. Está especialmente extendido en el ámbito de aplicaciones cliente-servidor y aplicaciones Web, al que Java da una buena cobertura con múltiples tecnologías enfocadas a la Web.

El sistema desarrollado en este proyecto (sin contar la interfaz Web) estará implementado en Java. La elección de este lenguaje se hizo en base a que es el mejor dominado por el desarrollador y por ser muy adecuado para el tipo de sistema Web implementado en este proyecto.

7.1.2 ECMAScript

ECMAScript es una especificación de lenguaje de programación publicada por ECMA International. Está basado en JavaScript, un lenguaje de programación interpretado creado por Netscape Communications Corp.

ECMAScript está inspirado en C y adopta nombres y convenciones de Java. Soporta orientación a objetos mediante prototipos y pseudoclases.

Generalmente se utiliza en navegadores web, permitiendo la creación de páginas web dinámicas con código ejecutado en el lado del cliente. Se puede usar en el lado del servidor, aunque es poco común. También se puede usar en otras aplicaciones no relacionadas con la web, como en documentos PDF o aplicaciones de escritorio.

En este proyecto se usa ECMAScript para controlar el comportamiento de la interfaz Web creada para el sistema y para modificarla dinámicamente. Se

escogió por ser la opción más estándar para la creación de páginas dinámicas.

7.1.3 C#

C# ya ha sido descrito en el apartado de **Aspectos teóricos** de este documento.

En este proyecto se utiliza C# como lenguaje objetivo de la generación de test. Se escogió por tener varias tecnologías de tratamiento de XML por defecto, lo que permite su comparación en los test.

7.2 Herramientas y Programas Usados para el Desarrollo

7.2.1 Eclipse

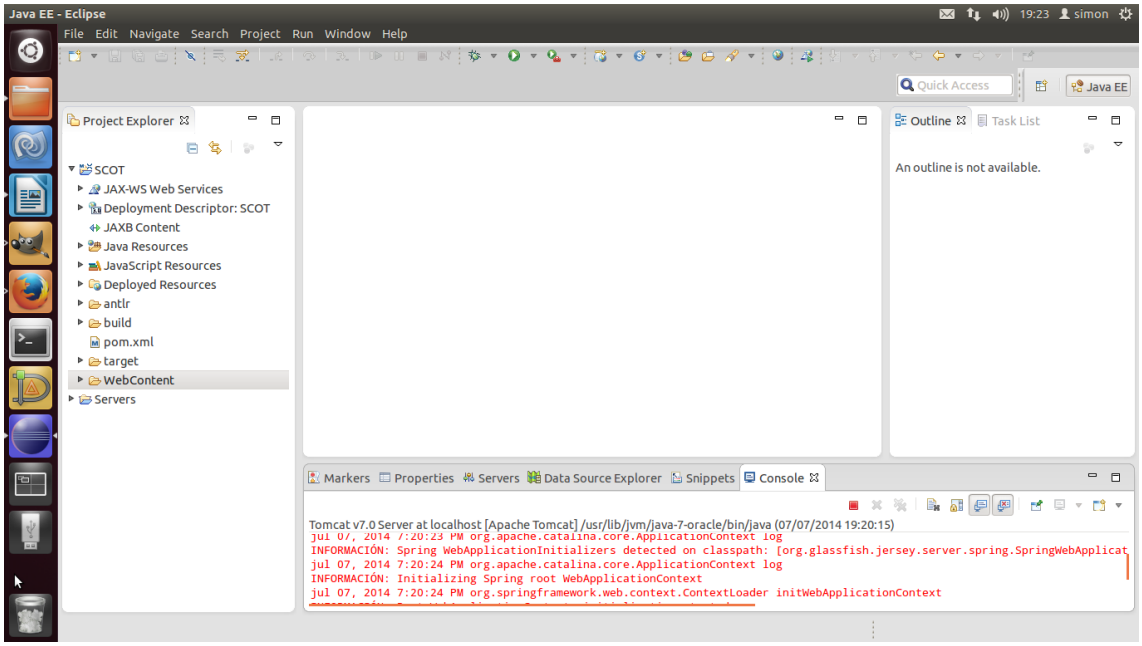


Figura 7.1: Pantalla de eclipse

Eclipse es un entorno de desarrollo integrado de código abierto y multiplataforma, desarrollado originalmente por IBM y actualmente por la Eclipse Foundation.

Eclipse, en su versión Kepler Service Release 2, es el entorno de trabajo que usamos para implementar toda la aplicación. Todos las clases, páginas web, gramáticas y demás archivos de nuestra aplicación fueron creados en Eclipse.

Eclipse también ofrece la posibilidad de integrar el servidor de aplicaciones Tomcat que se usó, por lo que Eclipse sirvió tanto para implementar la aplicación como para compilarla, empaquetarla, desplegarla y ejecutarla.

Eclipse ya viene por defecto con algunas extensiones que se usaron, como la integración con Maven.

7.2.2 Dia

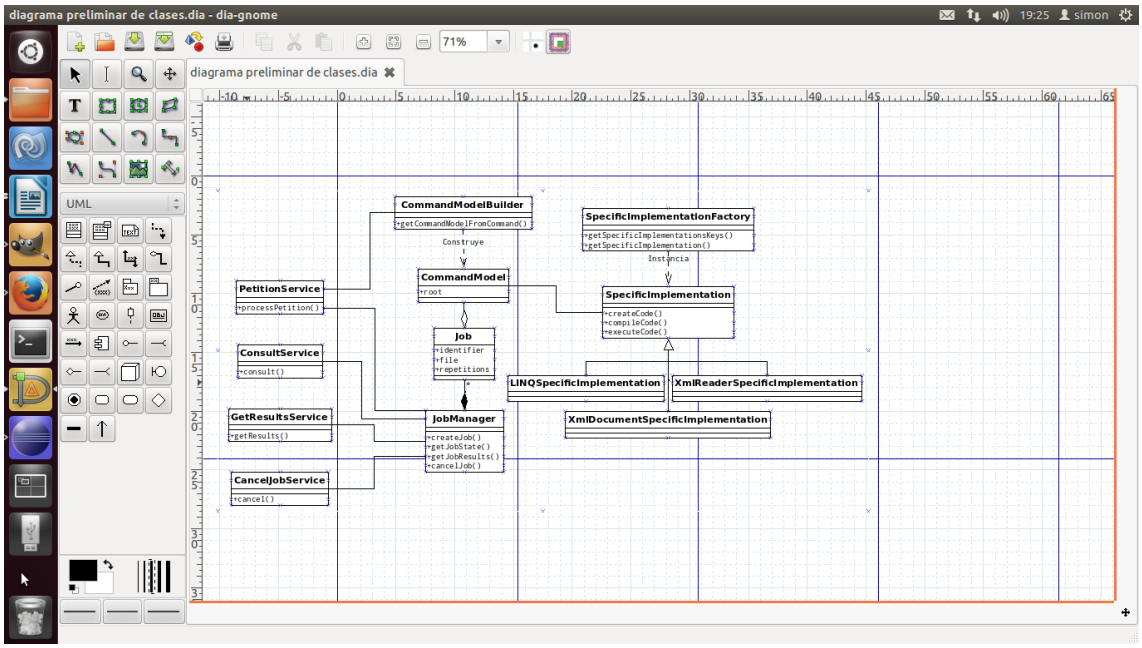
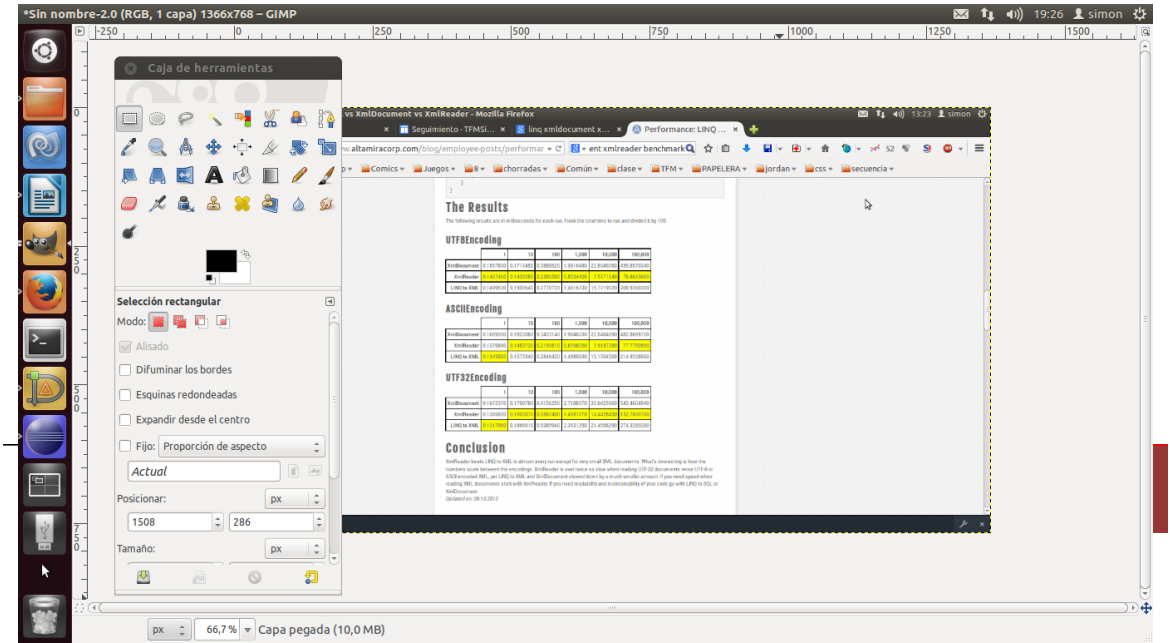


Figura 7.2: Pantalla de Dia

Día es un programa de creación de diagramas, de código abierto y multiplataforma. Es prácticamente un programa de dibujo con plantillas para los diferentes tipos de objetos de los diagramas, lo que lo hace muy flexible, pero también engorroso para determinados tipos de diagrama.

Durante el desarrollo se usó en su versión 0.97.2 para crear la mayoría de diagramas de esta documentación (a excepción de los de secuencia).

7.2.3 GIMP



7.2.4 Web Sequence Diagrams

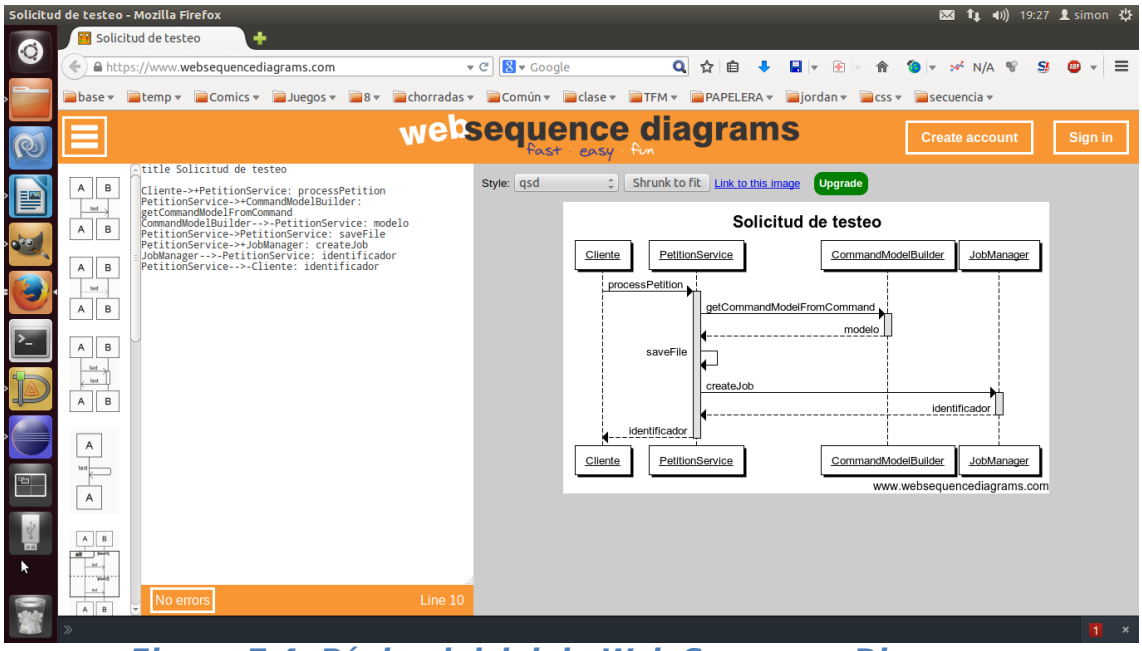


Figura 7.4: Página inicial de Web Sequence Diagrams

Web Sequence Diagrams es un sitio Web que ofrece una herramienta para la fácil creación de diagramas de secuencia introduciendo los pasos como comandos textuales.

Se usó en el proyecto para la creación de los diagramas de secuencia.

7.2.5 Microsoft Project

Microsoft Project es un conocido software para la gestión de proyectos. Permite controlar todos los aspectos de estos, desde los plazos a los recursos y a los costos.

En este proyecto se usó en su versión de 2013 para el apartado de planificación.

7.2.6 JQuery

JQuery es una librería de ECMAScript que ofrece facilidades en muchos aspectos, por ejemplo a la hora de trabajar con el árbol DOM u obtener portabilidad entre navegadores.

Es usada por la interfaz del proyecto en su versión 1.7.2.

La interfaz también hace uso del plugin JQuery Form en su versión 3.50.0-2014.02.05 para el envío de los datos del formulario por AJAX.

7.2.7 Google Code Prettify

Google Code Prettify es un conjunto de script y página de estilos que permite mostrar código en una página con sintaxis coloreada.

```
XElement document = XElement.Load(xmlFilePath);
IEnumerable elements = document.Descendants("Ana").Where( id1 => id1.Parent!=null && id1.Parent.Name=="P
foreach (XElement element in elements)
{
      /*#USER CODE#*/
}
```

Figura 7.5: Código coloreado con Google Code Prettify

En este proyecto se hace uso de este conjunto, en su versión del 4 de marzo de 2013, para mostrar al usuario el código generado formateado de manera más comprensible.

7.2.8 Google Charts

Google Charts es un servicio de Google que permite insertar gráficos en una página Web de manera sencilla.

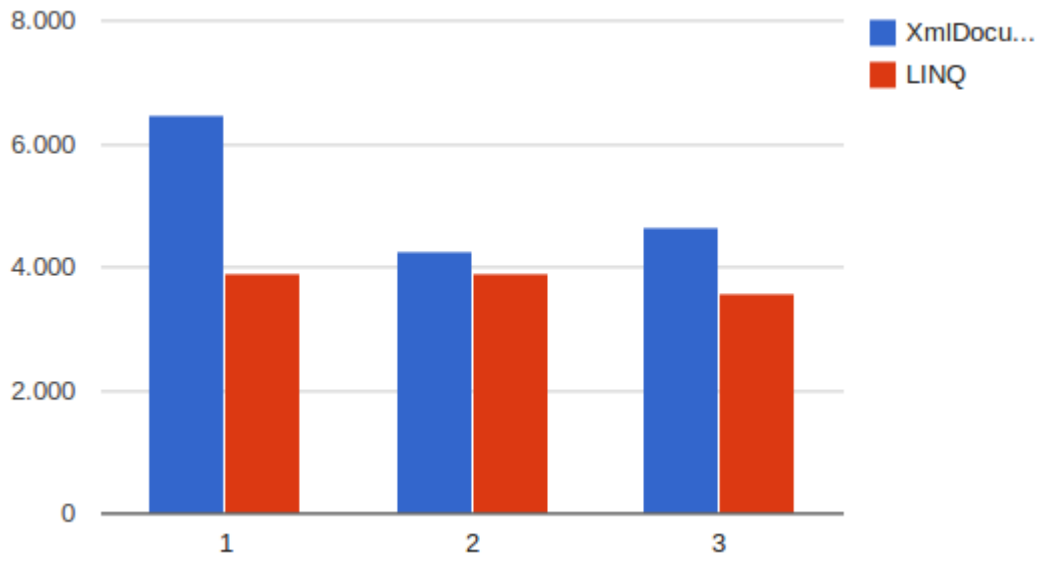


Figura 7.6: Gráfico de barras generado con Google Charts

En este proyecto se hace uso del servicio para crear la gráfica de tiempos de los resultados.

7.2.9 Maven

Maven es un sistema para la gestión y construcción de proyectos creado por la Apache Software Foundation.

Maven utiliza un modelo de configuración en XML llamado Project Object Model (POM) que permite describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos.

En este proyecto se utiliza Maven integrado en Eclipse para gestionar las dependencias de librerías externas.

7.2.10 ANTLR

ANTLR, creado principalmente por Terence Parr, es un generador de parseadores, intérpretes, compiladores y traductores de lenguajes específicos a partir de una descripción gramatical del lenguaje.

Simplemente proporcionándole un archivo de texto con las reglas léxicas y gramaticales ANTLR genera un conjunto de clases capaces de analizar el lenguaje descrito y generar un árbol sintáctico.

En este proyecto se utilizó en su versión 4.2.2 integrada con Eclipse para la generación del parseador del lenguaje específico de dominio y del modelo.

7.2.11 ANTLR IDE 4 para Eclipse

ANTLR IDE 4 es un plugin de Eclipse que permite la integración de ANTLR con este.

El plugin posibilita la creación y modificación de gramáticas y la generación de parseadores a partir de ellas de manera fácil desde Eclipse, y fue usado con este propósito en este proyecto en su versión 0.3.3.

7.2.12 Tomcat

Tomcat es un contenedor de Servlets y JSPs de código libre creado por la Apache Software Foundation. Permite el despliegue de aplicaciones web basadas en Java EE.

Cuenta con varios componentes, entre los que se cuentan Catalina, que proporciona el soporte para Servlets; Coyote, como conector HTTP; y Jasper como soporte para JSPs, compilándolos como Servlets.

Es el servidor más utilizado en Internet para desplegar aplicaciones web.

En este proyecto se usó como servidor para desplegar el sistema implementado en su versión 7.0.53.

7.2.13 Mono

Mono es una plataforma de software que implementa en código abierto el framework .NET, basada en los estándares de ECMA para C# y la Common Languaje Runtime.

Mono permite la compilación y ejecución de código C# y al contrario que las implementaciones oficiales de .NET es multiplataforma. Es por este último motivo por el que se decidió usarlo en este proyecto en su versión 2.10.8.1 para la compilación y ejecución del código generado.

7.2.14 Spring

Spring es un framework que ofrece numerosas utilidades para el desarrollo de aplicaciones, especialmente para aplicaciones Web.

Entre sus múltiples facetas se encuentra la de la inyección de dependencias, lo que permite crear sistemas cuyos componentes están poco acoplados entre sí.

En este sistema se utiliza en su versión 4.0.4 para inyectar implementaciones de interfaces en los objetos que hacen uso de ellas, así como para inyectar algunas configuraciones.

7.2.15 Jersey

Jersey es una implementación de JAX-RS que además proporciona su propia API con extensiones para esta tecnología.

En el proyecto se utiliza Jersey en su versión 2.8 para la creación de la API de servicios del sistema.

También se usa el módulo de Jackson de Jersey para la conversión a/de JSON.

7.3 Creación del Sistema

7.3.1 Problemas Encontrados

En el proceso de generación de código a partir del modelo se utiliza el patrón visitor. La idea inicial era utilizar el retorno de los métodos para formar el código, es decir, se llama al método del visitor, se obtiene como respuesta una cadena que contiene código y se inserta en su correspondiente lugar.

El problema es que la transformación a código es más compleja y algunos elementos no generan un único bloque de código, tienen que insertarlo en más de un lugar.

Para suplir este problema se creó un atributo de tipo StingBuilder en las clases que generan el código para contenerlo y que sus métodos puedan hacer inserciones en los lugares que necesiten.

No se produjo ningún problema más digno de mención, aparte de los imprevistos típicos de la implementación.

7.3.2 Descripción Detallada de las Clases

7.3.2.1 Clase Job

Nombre	**Tipo**	**Descripción**	**Hereda de…**
Job	-	Representa una tarea solicitada por un usuario.	
Responsabilidades			
Número	**Descripción**		
1	Contener la información de la petición		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	void	refreshTimestamp	-
Público	void	liberateResources	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado	-	String	identifier
Privado	-	File	file
Privado	-	CommandModel	model
Privado	-	int	repetitions
Privado	-	JobState	state
Privado	-	int	queueNumber
Privado	-	long	timeStamp
Privado	-	Map<String, Object>	results
Observaciones			
Se omiten getters y setters.			

7.3.2.2 Clase JobManagerImpl

Nombre	Tipo	Descripción	Hereda de…
JobManagerImpl	-	Es la clase responsable de manejar todo el ciclo de vida de los trabajos.	Implementa JobManager
Responsabilidades			
Número	**Descripción**		
1	Crear los trabajos		
2	Mantener la cola de ejecución		
3	Ejecutar los trabajos		
4	Eliminar los trabajos y liberar sus recursos		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público, estático	JobManagerImpl	getInstance	-
Público	String	createJob	CommandModel model, File file, int repetitions
Público	Map<Integer, Object>	getJobState	String identifier
Público	Map<String, Object>	getJobResults	String identifier
Público	void	cancelJob	String identifier
Privado	void	deleteJob	Job job
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado	Estático	JobManagerImpl	instance
Privado	-	Lock	lock
Privado	-	Condition	notEmpty
Privado	-	Thread	executorThread
Privado	-	Thread	removerThread
Privado	-	int	identifierLength
Privado	-	long	jobSessionMilliseconds
Privado	-	long	removerWaitMilliseconds
		Map<String,Job>	jobs
		Queue<Job>	executionQueue
		String	workDirectory
Observaciones			
Se omiten getters y setters. Se omiten las dos clases internas JobExecutor y JobRemover, que simplemente implementan la interfaz Runnable.			

7.3.2.3 Clase AbstractSpecificImplementation

Nombre	Tipo	Descripción	Hereda de…
AbstractSpecificImplementation	Abstracta	Es la clase base para las implementaciones de paso de modelo a código.	Implementa SpecificImplementation y SCOTVisitor
Responsabilidades			
Número	**Descripción**		
1	Crear el código de los test a partir del modelo		
2	Compilar el código		
3	Ejecutar el código		
4	Obtener los resultados		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Privado	void	saveToFile	String content
Público	boolean	createCode	Map<String, Object> resultMap
Protegido	String	imports	-
Protegido	StringBuilder	process	BlockContext block
Privado	void	process	BlockContext block, StringBuilder baseCode, Map<String, Object> resultMap
Protegido	void	insertCodeBefore	StringBuilder base, String insertTag, CharSequence code
Protegido	void	insertCodeAfter	StringBuilder base, String insertTag, CharSequence code
Protegido	void	InsertCodeReplacing	StringBuilder base, String insertTag, String code
Protegido, estático	String	getTemplate	String name
Privado, estático	String	loadTemplate	String name
Publico	boolean	compileCode	Map<String, Object> resultMap

Publico	boolean	executeCode	Map<String, Object> resultMap
Privado	Process	runCommand	String[] command
Público	void	liberateResources	-
Público	void	setProperties	Map<String, Object> properties
Público	void	setProperty	String key, Object property
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado	-	CommandModel	model
Privado	-	File	directory
Privado	-	File	codeFile
Privado	-	File	execFile
Privado	-	String	name
Privado	-	int	repetitions
Privado	Estático	Map<String, String>	templates
Privado	Estático, final	String	TAG_MAINCONTENT
Privado	Estático, final	String	TAG_USING
Privado	Estático, final	String	TAG_USERCODE
Privado	Estático, final	String	TEMPLATE_BASE
Privado	Estático, final	String	TEMPLATE_BLOCKBASE
Privado	Estático, final	String	TEMPLATE_INNERCODE
Privado	Estático, final	String	TAG_INSERTPOINT
Privado	Estático, final	String	TAG_REPETITIONS
Observaciones			
SCOTVisitor es una interfaz generada por ANTLR a partir de la gramática. Se omiten getters y setters.			

8 Desarrollo de las Pruebas

8.1 Pruebas de Integración y del Sistema

Durante las pruebas del sistema se descubrieron algunos errores en la generación de código.

Una vez comprobado el funcionamiento incorrecto se procedió a analizar el código generado en un IDE (MonoDevelop) para descubrir cuál era el error, y de ahí deducir en que parte de la generación se encontraba el error y corregirlo.

El proceso se repitió hasta obtener el funcionamiento correcto.

Los errores se debían en su mayoría a inserciones de código en lugares incorrectos a la hora de generar los test.

8.2 Pruebas de Usabilidad y Accesibilidad

8.2.1 Pruebas de Usabilidad

8.2.1.1 Actividad guiada con usuario 1

8.2.1.1.1 Perfil del usuario

El usuario es un ingeniero técnico informático que está en camino de ser ingeniero informático. Conoce bien XML y como trabajar con él.

8.2.1.1.2 Resultado de las actividades guiadas

Nº de prueba	Nº de intentos	Notas
1	1	-
2	1	-
3	1	-
4	1	-

8.2.1.1.3 Respuestas al formulario

Pregunta	Valor dado (min 0, max 5)
¿Le gusta el aspecto de la interfaz?	4
¿Considera que es fácil de usar?	5
¿Considera que el DSL es fácil de entender?	5
¿Considera que el DSL es completo?	3
¿Le gusta el código generado?	5
¿Considera que el sistema es útil?	4

8.2.1.2 Actividad guiada con usuario 2

8.2.1.2.1 Perfil del usuario

El usuario posee un módulo de programación y está familiarizado con XML.

8.2.1.2.2 Resultado de las actividades guiadas

Nº de prueba	Nº de intentos	Notas
1	1	-
2	2	Pone “childrenOf” en lugar de “childOf”.
3	2	Usa “||” y “&&” en lugar de “|” y “&”.
4	1	-

8.2.1.2.3 Respuestas al formulario

Pregunta	Valor dado (min 0, max 5)
¿Le gusta el aspecto de la interfaz?	4
¿Considera que es fácil de usar?	5
¿Considera que el DSL es fácil de entender?	5
¿Considera que el DSL es completo?	4
¿Le gusta el código generado?	5
¿Considera que el sistema es útil?	3

8.2.1.3 Actividad guiada con usuario 3

8.2.1.3.1 Perfil del usuario

El usuario no tiene estudios informáticos. Los conocimientos que posee son autodidactas. Conoce XML, sabe lo que es y tiene conocimientos básicos sobre su estructura.

En principio no es un usuario objetivo, pero se decidió incluirlo en las pruebas por falta de mejores voluntarios y para tener una opinión del sistema que no sea de un profesional.

8.2.1.3.2 Resultado de las actividades guiadas

Nº de prueba	Nº de intentos	Notas
1	1	-
2	1	-
3	4	El usuario no está familiarizado con las expresiones lógicas y necesita bastante asistencia para esta prueba.
4	1	-

8.2.1.3.3 Respuestas al formulario

Pregunta	Valor dado (min 0, max 5)
¿Le gusta el aspecto de la interfaz?	5
¿Considera que es fácil de usar?	5
¿Considera que el DSL es fácil de entender?	3
¿Considera que el DSL es completo?	4
¿Le gusta el código generado?	3
¿Considera que el sistema es útil?	4

8.2.1.4 Conclusiones de las pruebas de Usabilidad

Por las pruebas con voluntarios podemos ver que a los usuarios con conocimientos adecuados no les resulta difícil aprender el uso del sistema. Los usuarios se muestran satisfechos con el funcionamiento general del sistema, consideran que la interfaz es correcta y fácil de usar y que el DSL es sencillo de aprender. Si bien hay puntos mejorables el resultado general es bastante bueno.

9 Manuales del Sistema

9.1 Manual de Instalación y ejecución

La instalación de la aplicación final es tan fácil como desplegar el archivo WAR en un servidor de aplicaciones. Aquí se describe cómo instalar el entorno de desarrollo. Téngase en cuenta que en el WAR se deberá igualmente configurar el “application-context.xml” como se indica en este manual en el paso 18.

1. Descargar las herramientas necesarias. Se necesitarán Tomcat como servidor de aplicaciones y Eclipse como IDE. El resto de herramientas son módulos de Eclipse.
2. Descomprimir los archivos de cada herramienta.
3. Abrir el Eclipse e indicar el directorio de trabajo.
4. Para integrar ANTLR con Eclipse se necesitarán el plugin ANTLR4 IDE y XText. En el menú de Eclipse ir a “Help -> install new software...”.
5. Introducir en el campo “Work with” la URL “http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/” y pulsar enter.
6. Seleccionar en la lista XText 2.6.2 (la última versión en el momento de escribir este documento)..

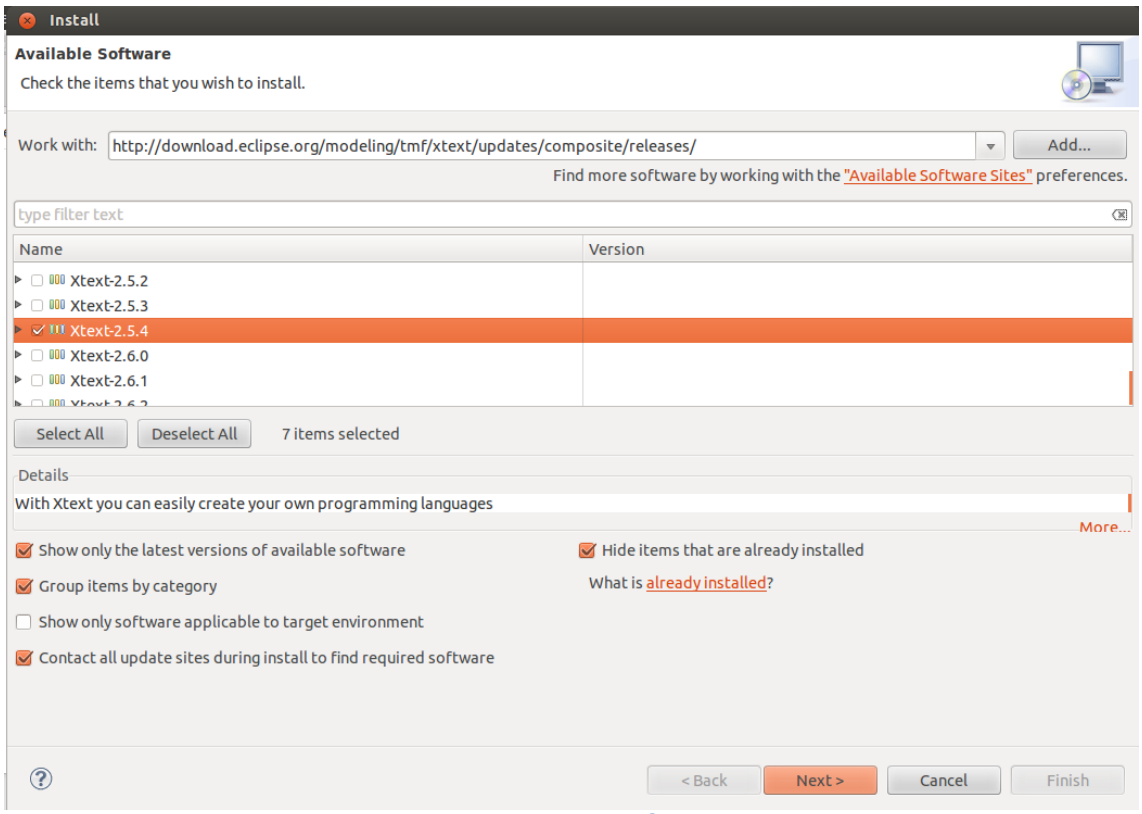


Figura 9.1: Pantalla de selección para instalar XText

7. Pulsar “Next>” para seguir, aceptar la licencia y terminar la instalación.
8. Cuando termine el proceso decir que no se quiere reiniciar Eclipse en este momento (pues vamos a seguir instalando software).
9. Volver a la pantalla de instalación de nuevo software.
10. Introducir en el campo “Work with” la URL “http://dl.bintray.com/jknack/antlr4ide” y pulsar enter.
11. Seleccionar ANTLR 4 IDE.

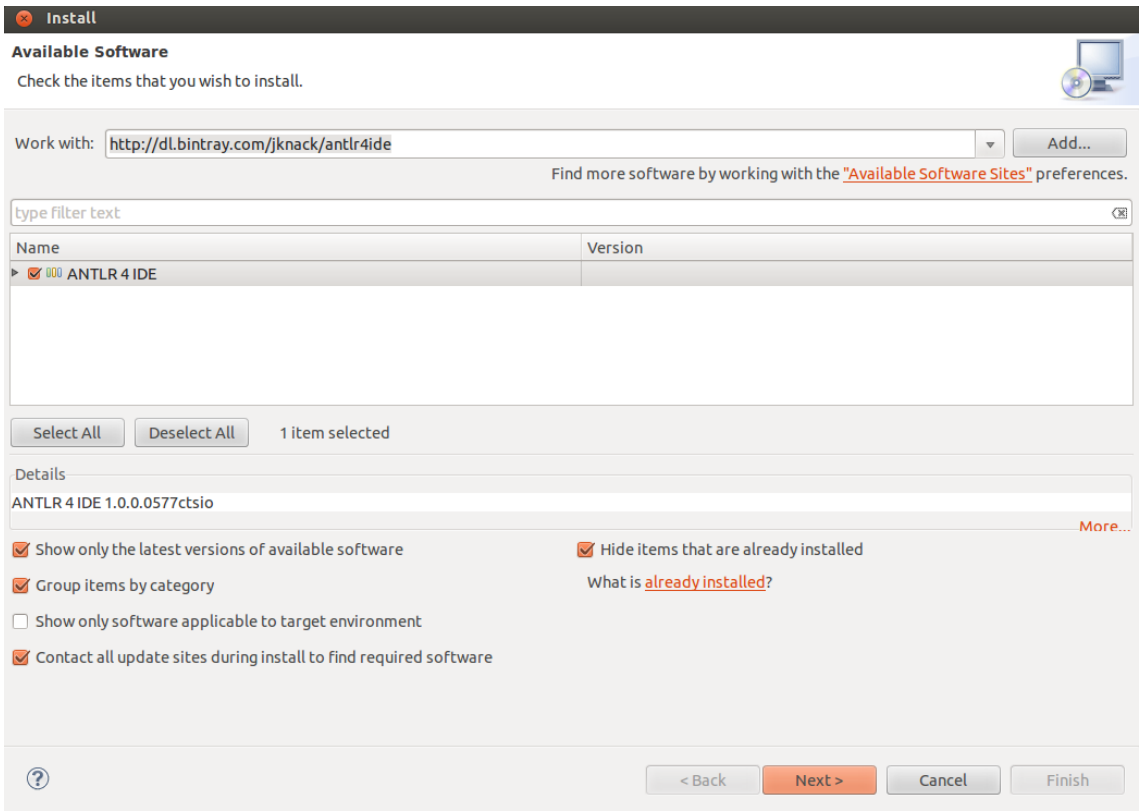


Figura 9.2: Pantalla de selección para instalar ANTLR 4 IDE

12. Completar la instalación.

13. Decir a Eclipse que se confía en el contenido sin firmar a instalar y que se quiere que se reinicie.

14. Cuando eclipse se reinicie ir al menú “Window -> Preferences”, y una vez en las preferencias ir a “ANTLR4 -> Tool” y marcar la opción “Generate parse tree visitors”. También se puede desmarcar la opción “Generate a parse tree listener” si no se va a utilizar en alguna extensión. Introducir en el campo “directory” el valor “./src-antlr”.

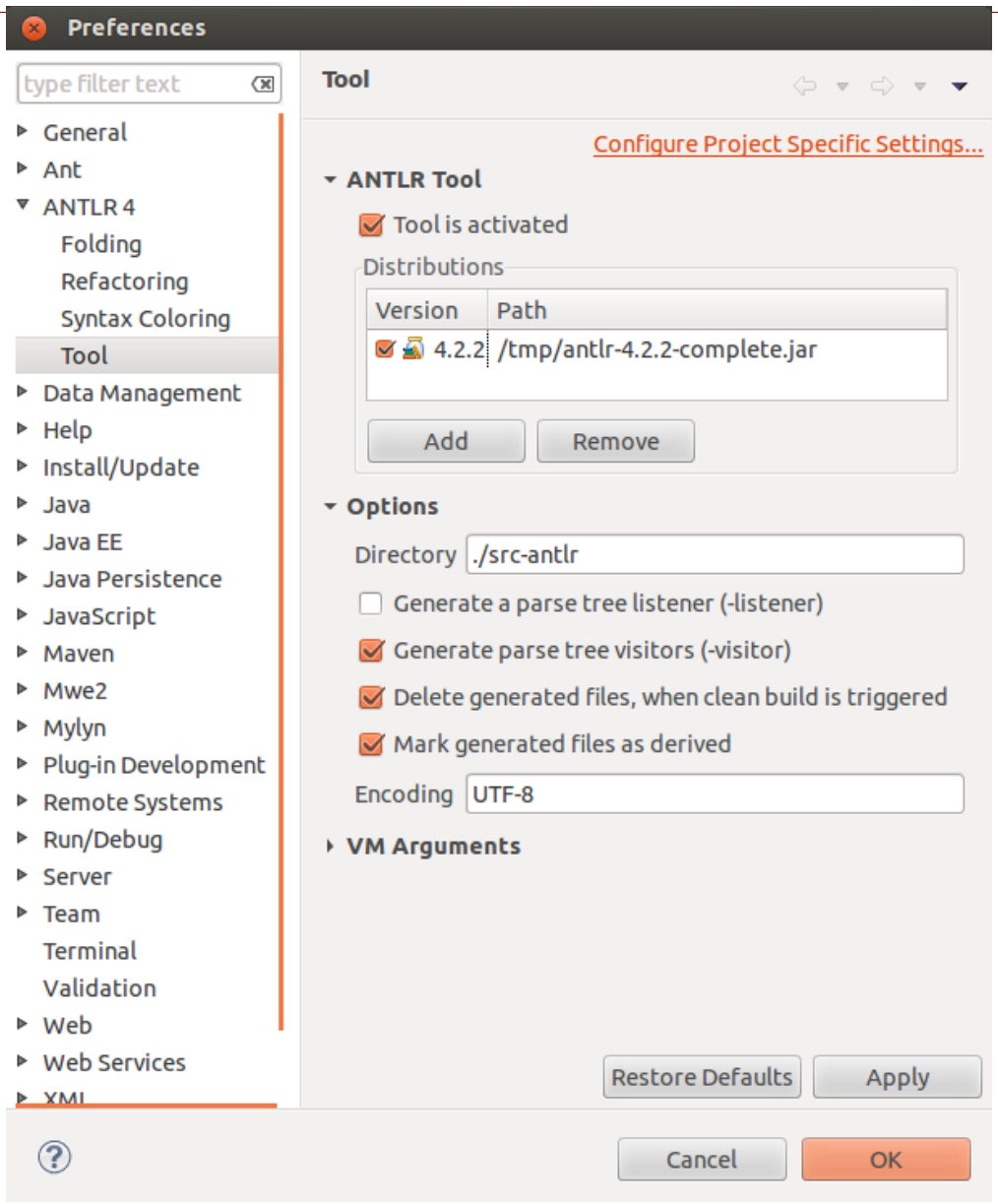


Figura 9.3: Pantalla de propiedades de ANTLR4 IDE

15. Aplicar las propiedades e ir al apartado “Server -> Runtime enviroment”.

16. Pulsar en el botón “Add”, seleccionar “Apache Tomcat v7.0” y pulsar “Next”.

17. En la siguiente pantalla pulsar “Browse”, seleccionar la carpeta del Tomcat que descomprimimos y pulsar “Finish”.

18. Configurar el propio sistema. Para abrir el archivo “application-context.xml” de la carpeta “WEB-INF” y configurar las siguientes propiedades.

```
<!-- Directorio donde se guardan los archivos subidos -->
<bean id="scot.fileupload.path" class="java.io.File">
      <constructor-arg value="/path/al/directorio">
```

```
      </constructor-arg>
</bean>
```

```
<!-- Indicamos al manejador de trabajos el directorio de archivos
temporales -->
<bean
class="org.springframework.beans.factory.config.MethodInvokingFactory
Bean">
      <property name="staticMethod"
value="es.uniovi.scot.job.JobManagerImpl.setWorkDirectory">
      </property>
      <property name="arguments"
value="/path/al/directorio"></property>
</bean>
```

19. El paso final es crear el servidor. Pulsar el enlace para crear un servidor en la pestaña “Servers”.

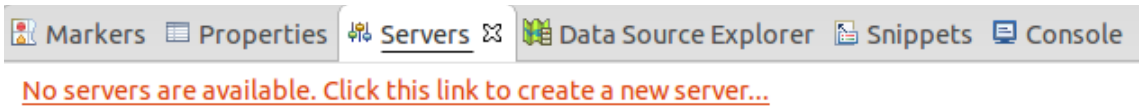


Figura 9.4: Pestaña Servers de Eclipse

20. En la pantalla que se abre seleccionar “Tomcat v7.0 Server” y pulsar “Next>”.

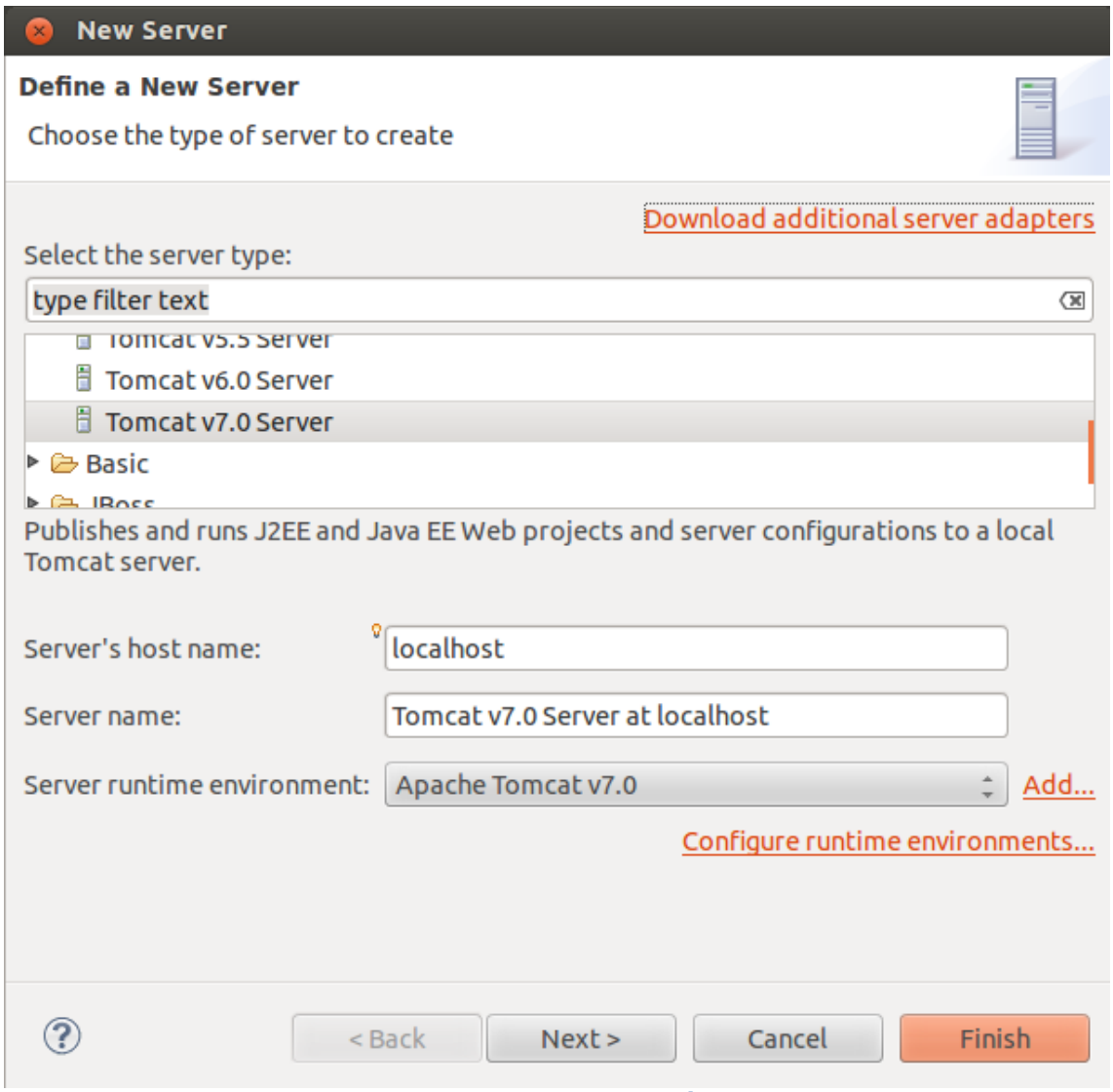


Figura 9.5: Pantalla de creación de servidor

21. Agregar la aplicación SCOT al despliegue del servidor y finalizar.

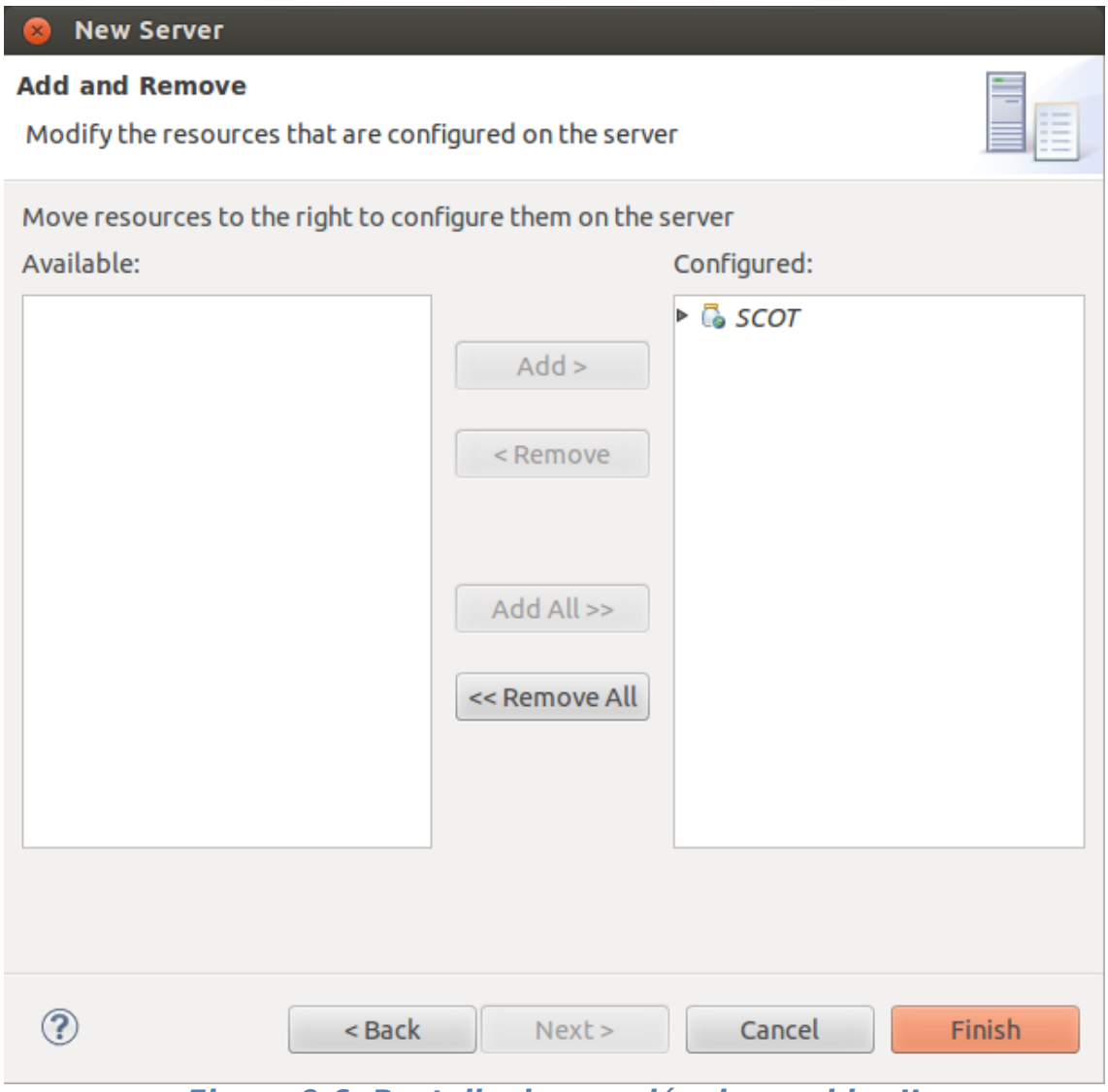


Figura 9.6: Pantalla de creación de servidor II

22. Para iniciar el servidor solo hay que seleccionarlo y pulsar el botón de inicio.

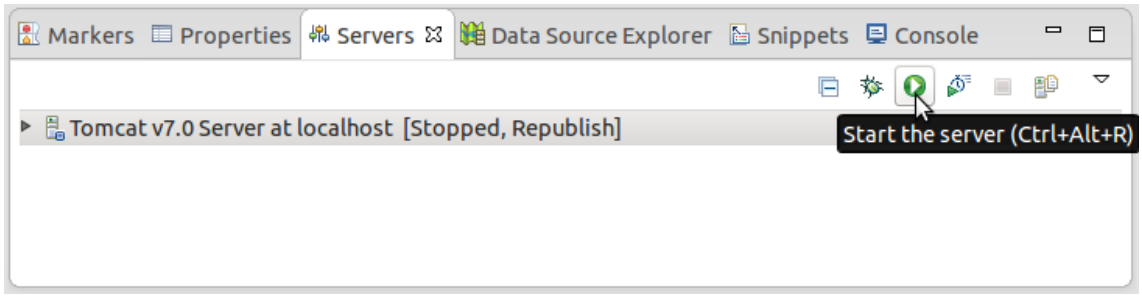


Figura 9.6: Inicio del servidor desde Eclipse

El sistema en que se ejecute la aplicación tendrá que tener instalado Mono. La instalación dependerá del sistema operativo. En Ubuntu se puede instalar con el comando “sudo apt-get install mono”.

9.2 Manual de Usuario

Para hacer uso de la interfaz Web de usuario se deben seguir los siguientes pasos.

1. Acceder a la interfaz desde un navegador Web en la dirección en que esté desplegada. Si está desplegada en la dirección local por defecto será “http://localhost:8080/SCOT”.

2. Rellenar los campos del formulario. Estos campos son la expresión de la consulta que se quiere probar (cuya sintaxis se explica más adelante), el número de veces que se quiere repetir cada test y el archivo XML sobre el que se quieren ejecutar los test.

Figura 9.7: Formulario de la interfaz Web

3. Pulsar el botón “Aceptar” y la petición será enviada. En el panel de mensajes situado debajo del formulario se indicará si fue aceptada o si hubo algún error.

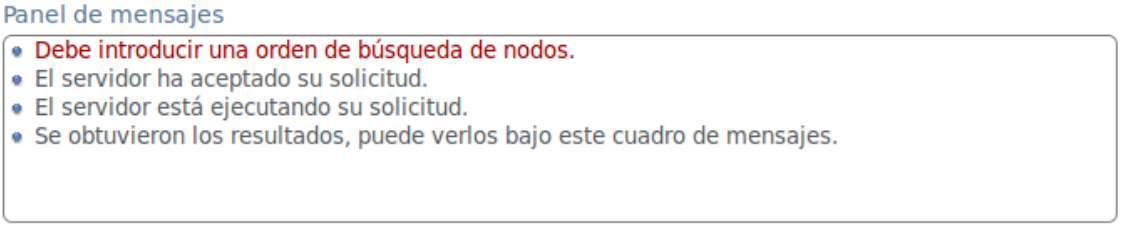


Figura 9.8: Panel de mensajes de la interfaz Web

4. Si la petición es aceptada se mostrará en el panel de mensajes la evolución de su estado. Este estado puede ser en espera en la cola de ejecución (se indicará en que posición) o ejecutándose.

5. Una vez que el trabajo se ejecute se obtendrán los resultados y se mostrará un mensaje indicándolo en el panel de mensajes.

6. Los resultados son mostrados debajo del panel de mensajes. Para cada test se muestra:

- La parte de la consulta indicada en el formulario que generó el test.
- La cantidad de nodos que se encontraron en el test sobre el archivo XML suministrado en el formulario.
- El tiempo en milisegundos que llevó la ejecución del test el número de veces indicado en el formulario.
- El código generado para realizar el test en esa tecnología.

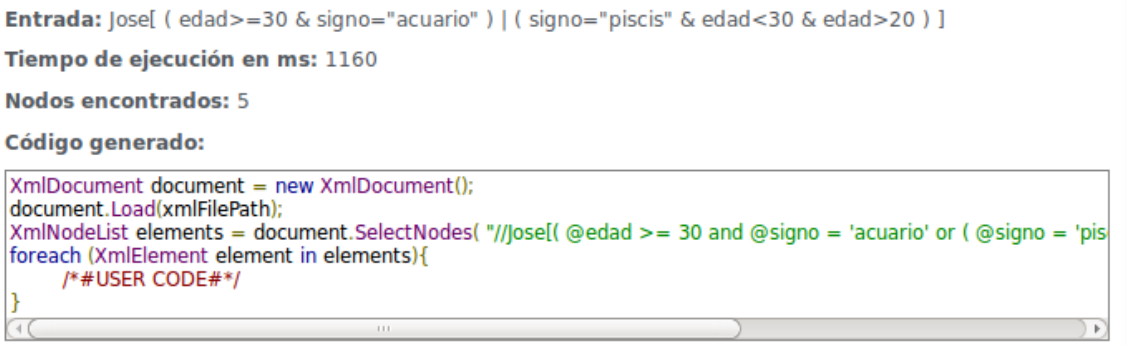


Figura 9.9: Pantalla de un resultado mostrado en la interfaz Web

7. Debajo de todos los resultados se muestra un gráfico de barras con los tiempos de las diferentes tecnologías en cada test.

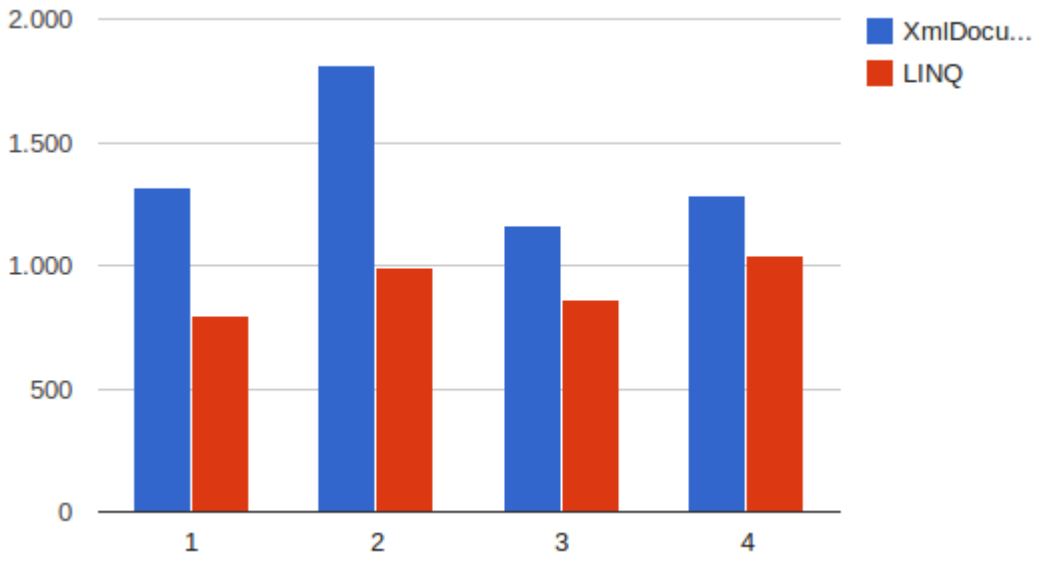


Figura 9.10: Pantalla del gráfico de resultados de la interfaz Web

Para introducir la consulta en el formulario se utiliza un lenguaje específico de dominio, cuya sintaxis se explica a continuación.

Para buscar nodos por su etiqueta se introduce el nombre. La siguiente consulta buscaría todos los nodos “foo” del documento.

```
foo
```

Se pueden buscar nodos que cumplan cierta relación con otros nodos. La siguiente consulta buscaría los nodos “foo” que son padres de algún nodo “bar”.

```
foo parentOf bar
```

Las relaciones que hay son:

- **parentOf:** el primer nodo es padre del segundo.
- **ancestorOf:** el primer nodo es ancestro del segundo.
- **childOf:** el primer nodo es hijo del segundo.
- **descendantOf:** el primer nodo es descendiente del segundo.
- **siblingOf:** los nodos son hermanos.

Las relaciones se pueden encadenar para búsquedas más complejas. La siguiente consulta busca los nodos “foo” que son padres de algún nodo “bar” que a su vez sea ancestro de algún nodo “qux” que a su vez sea hermano de algún nodo “baz”.

```
foo parentOf bar ancestorOf qux siblingOf baz
```

Los nodos se pueden filtrar por sus atributos. La siguiente consulta busca los nodos “foo” que tienen una propiedad “bar” con valor “qux”.

```
foo[bar=”qux”]
```

En este ejemplo se hace una comparación con una cadena, que va entre comillas. Los operadores para usar con cadenas son:

Símbolo	Uso
=	Selecciona los nodos cuyo atributo coincide con la cadena.
!=	Selecciona los nodos cuyo atributo no coincide con la cadena.

En el caso de los atributos numéricos no se usan comillas y los operadores son:

Símbolo	Uso
=	Selecciona los nodos cuyo atributo coincide con el valor.
!=	Selecciona los nodos cuyo atributo no coincide con el valor.
<	Selecciona los nodos cuyo atributo es menor que el valor.
>	Selecciona los nodos cuyo atributo es mayor que el valor.
<=	Selecciona los nodos cuyo atributo es menor o igual que el valor.
>=	Selecciona los nodos cuyo atributo es mayor o igual que el valor.

Los atributos se pueden combinar como se quiera usando operadores lógicos y paréntesis. Los operadores lógicos son:

Símbolo	Uso
|	Operador lógico OR.
&	Operador lógico AND.

La siguiente consulta busca los nodos “foo” cuyo atributo “bar” sea mayor que 10 o aquellos cuyo atributo “bar” sea menor o igual que 7 y su atributo “qux” diferente de la cadena “baz” y que sean padres de algún nodo “quux” con un atributo “bar” igual a 6.

```
foo[ bar>10 | ( bar<=7 & qux!=”baz” ) ] parentOf quux[ bar=6 ]
```

También se puede hacer la unión de dos consultas usando el operador “|”. La siguiente consulta busca los nodos “foo” que son ancestros de algún nodo “bar” y los nodos “qux” con un atributo “foobar” menor que 5.

```
foo ancestorOf bar | qux[ foobar<5 ]
```

Por último, en una sola expresión se pueden concatenar varias consultas separándolas con punto y coma. Cada consulta generará un test separado.

9.3 Manual del Programador

9.3.1 Cómo agregar implementaciones de test para nuevas tecnologías

Para agregar una nueva tecnología a los test hay que crear una clase que implemente la interfaz *es.uniovi.scot.specifiImplementations .SpecificImplementation*. Se recomienda también implementar la interfaz *es.uniovi.scot.model.antlr.SCOTVisitor* o heredar de la clase *es.uniovi.scot.model.antlr.SCOTBaseVisitor* para recorrer el árbol del modelo.

La interfaz SpecificImplementation consta de los siguientes métodos:

- **setProperties(Map<String, Object> properties):** recibe un mapa de propiedades para configurar el proceso. Estas propiedades se enumeran más adelante.
- **setProperty(String key, Object property):** igual que el anterior pero recibiendo una propiedad suelta.
- **setModel(CommandModel model):** recibe el modelo de la consulta o consultas a convertir en código.
- **createCode(Map<String, Object> resultsMap):** crea el código a partir del modelo y retorna un booleano que indica si la operación tuvo éxito. Recibe el mapa de resultados para agregar la información que sea pertinente.
- **compileCode(Map<String, Object> resultsMap):** como el anterior pero en vez de crear el código lo compila.
- **executeCode(Map<String, Object> resultsMap):** como los anteriores pero ejecutando el código. Este será el método que obtenga los resultados de los test.
- **liberateResources():** libera cualquier recurso que tenga asignada la clase.

Las propiedades que pasa actualmente el sistema a *setProperty* y *setProperties* son las siguientes:

- **name:** un String con el nombre de la tecnología. Debe ser el que se use para guardar datos en el mapa de resultados.
- **directory:** un File que apunta al directorio para los archivos de código y ejecutables.
- **file:** un File que apunta al archivo XML sobre el que ejecutar los test.

- **repetitions:** un Integer con el número de repeticiones para los test.

Los niveles de la estructura de resultados son los siguientes:

1. **El mapa base:** como llaves usa los nombres de las tecnologías y guarda los resultados de los test de cada una.
2. **La lista de resultados de cada tecnología:** almacena los resultados de cada test para una tecnología. Existe aunque la petición tenga una única consulta. Los test están en el orden en que aparecen en la petición del usuario.
3. **El mapa de resultados de cada test:** almacena los resultados obtenidos en un test asociándolos a una cadena con el nombre del dato. Los datos son los que siguen.
    1. **input:** la parte de la entrada del usuario que generó el test.
    2. **nodes:** el número de nodos encontrados al ejecutar el test sobre el archivo XML.
    3. **time:** el tiempo en milisegundos que llevó la ejecución sobre el archivo XML. Es la suma de todas las repeticiones del test, no la media.
    4. **code:** el código generado para el test.

Se recomienda, por facilidad de implementación, usar la clase de utilidad *es.uniovi.scot.util.ResultsUtils* para agregar los resultados al mapa. Consta de dos métodos: *addBlockInputAndCode* (para ser usado en el método de generación de código) y *setExecutionTimeAndNodes* (para ser usado tras la ejecución).

Si la tecnología que se quiere implementar pretende generar código C# se puede crear una clase que herede de *es.uniovi.scot.specificImplementations.AbstractSpecificImplementation* e implemente los métodos abstractos *imports()* y *process(BlockContext)*. El primero simplemente debe retornar una cadena con los imports adicionales que necesite la tecnología o una cadena vacía si no necesita ninguno. El segundo recibe un objeto generado por ANTLR que representa una consulta a convertir en código, y retorna una cadena con el código generado.

Esta clase contiene algunos métodos útiles para usar en la creación de código:

- **insertCodeBefore:** recibe un *StringBuilder* con el código base, un *String* con la etiqueta (la cadena que indica dónde insertar) y la cadena a insertar. Inserta la cadena justo antes de la etiqueta manteniendo esta.
- **insertCodeAfter:** lo mismo que la anterior pero haciendo la inserción detrás.

- **insertCodeReplacing:** lo mismo que las anteriores pero sustituyendo la etiqueta por la inserción.
- **getTemplate:** recibe un nombre de plantilla y la retorna en forma de cadena. La plantilla debe estar en el paquete de la clase y tener como extensión *.template*.

La clase *AbstractSpecificImplementation* también implementa la interfaz *SCOTVisitor* que se recomienda usar para recorrer el subárbol a partir del objeto recibido.

Una vez implementada la clase hay que configurar la factoría de implementaciones para que la use. Esto se hace editando el archivo de Spring *application-context.xml*, situado en la carpeta *WEB-INF*. Hay que agregar al mapa de implementaciones la clase creada indicando como llave el nombre de la tecnología y como valor la clase (Class) de la nueva implementación. Se muestra a continuación la parte del archivo donde se crea el mapa con las dos implementaciones que hay actualmente.

```
<!-- Agregamos las implementaciones específicas para el modelo -->

<bean
class="org.springframework.beans.factory.config.MethodInvokingFactory
Bean">
      <property name="staticMethod"
value="es.uniovi.scot.specificImplementations.SpecificImplementationF
actory.setSpecificImplementations">

      </property>
      <property name="arguments">
            <map>
                  <entry key="LINQ">
                        <value type="java.lang.Class">
es.uniovi.scot.specificImplementations.LINQSpecificImplementation
                        </value>
                  </entry>
                  <entry key="XmlDocument">
                        <value type="java.lang.Class">
es.uniovi.scot.specificImplementations.XmlDocumentSpecificImplementat
ion
                        </value>
                  </entry>
            </map>
      </property>
```

9.3.2 Cómo crear un cliente para la API Web del sistema

Los clientes del sistema deberían seguir el siguiente protocolo de actuación para utilizar la API Web:

1. Solicitar al sistema que lleve a cabo un testeo. Para ello se debe enviar al path **“services/petition”** un mensaje POST multiparte que contenga los siguientes elementos:
    1. **file:** el archivo sobre el que se ejecutarán los test.
    2. **repetitions:** el número de repeticiones que se ejecutarán los test. Un entero mayor que cero.
    3. **command:** la consulta o consultas a ejecutar expresadas en el DSL del sistema.
2. El sistema retornará un objeto JSON con el siguiente contenido.
    1. **state:** una cadena que indica si la petición fue aceptada (la cadena será “ok”) o si se produjo algún error (la cadena será “error”).
    2. **identifier:** una cadena que identifica al trabajo creado en el servidor y debe ser almacenada para las siguientes interacciones con el servidor. Sólo está presente si state=”ok”.
    3. **errors:** un array de cadenas de error. Sólo presente si state=”error”;
3. A partir de entonces se puede consultar el estado del trabajo en el path **“services/consult”** enviando un mensaje POST adjuntando el identificador del trabajo con el nombre **identifier**. Queda a elección del implementador cada cuanto hacer la consulta, pero el sistema elimina los trabajos que llevan demasiado tiempo sin ser consultados (por defecto los que llevan más de diez minutos).
4. El sistema retornará un objeto JSON con un atributo **state** que es una cadena que indica el estado del trabajo. Puede tener los siguientes valores.
    1. **queued:** indica que el trabajo está esperando en la cola de ejecución. Cuando este es el estado el objeto JSON incluye un atributo **number** que indica la posición en la cola.
    2. **executing:** indica que el trabajo se está ejecutando.
    3. **finished:** indica que el trabajo ya se ejecutó y se obtuvieron los resultados.

4. **error:** indica que por algún motivo no se pudieron ejecutar correctamente los test.

5. **nonexistent:** indica que no hay ningún trabajo en el sistema asociado al identificador dado.

5. Cuando el estado es finished se pueden obtener los resultados enviando al path **“services/results”** una petición POST adjuntando el identificador del trabajo con el nombre **identifier**. La estructura de los resultados ya ha sido explicada en el apartado anterior.

6. Para cancelar una petición se debe enviar al path **“services/cancel”** una petición POST adjuntando el identificador del trabajo con el nombre **identifier**.

10 Conclusiones y Ampliaciones

10.1 Conclusiones

Exceptuando el crear una implementación para los test de XmlReader, todos los objetivos del proyecto se cumplieron. El sistema funciona correctamente y es reutilizable y extensible, el DSL es adecuado y la interfaz Web es funcional.

Se obtuvieron dos implementaciones de tecnologías que funcionan muy bien, generando código limpio y robusto. Esto le da al sistema no solo la utilidad de testeo de tecnologías, sino también utilidad como generador automático de código, aumentando sus posibilidades y valor como herramienta.

El sistema funciona bien y podría ser usado en otros proyectos ahorrando tiempo y dinero en pruebas durante el desarrollo, automatizando el proceso de selección de tecnologías o ayudando a usuarios sin experiencia a aprender a implementar en una determinada tecnología.

El trabajo con modelos hace bastante más complicada la tarea de diseño, implementación y búsqueda de errores. No solo hay que tener en cuenta lo que se implementa, si no lo que ello va a implementar a su vez, multiplicándose las probabilidades de error y dificultando el localizarlo.

Esto se ve compensado por las posibilidades que brinda esta metodología y los buenos resultados que se consiguen. La funcionalidad que brinda esta aplicación no se podría lograr de no ser por ella.

Aunque siempre se desea poder haber hecho más el autor está contento con el producto conseguido, lo que siempre está bien, más aún tratándose de un trabajo al que se le dedicaron tantas horas.

Este proyecto también fue para el autor el primero en involucrar el uso de generación de código. Al principio del proyecto había subestimado la dificultad de la tarea y no fue hasta que se adentró en el diseño e implementación que se dio cuenta. No obstante, el autor está contento con la experiencia, y sin duda se planteará esta metodología en proyectos futuros. También fue la primera ocasión en que hizo uso de algunas herramientas que le resultaron muy útiles, como ANTLR y Maven.

No lograr la implementación para XmlReader fue una pequeña decepción, pues precisamente por ser la tecnología más básica y difícil de usar es para la que resultaría más interesante la obtención automática de código. El

problema con esta tecnología es que el DSL es demasiado complejo para obtener una implementación en una herramienta que funciona a tan bajo nivel. La consecución de este reto bien podría ser un proyecto aparte.

Por ello se tuvo que decidir entre crear un DSL muy limitado y que se pudiese implementar en XmlReader en un tiempo aceptable o crear un DSL más completo y dejar la implementación de XmlReader para un futuro. Esta última opción fue la que se consideró más deseable.

Con este proyecto se demuestra la utilidad del diseño orientado a modelos en general, y en particular como metodología para automatizar el testeo de distintas tecnologías con una base común. Este sistema es sin duda más fiable y dinámico que los análisis y comparativas generales que sólo pueden ofrecer resultado genéricos que no se puede asegurar que sean extrapolables a un caso concreto. También es mucho más rápido y menos propenso a errores humanos que implementar las pruebas a mano. Es sin duda una opción a tener en cuenta en campos que necesiten de comparativas adaptadas y fiables, y sin duda puede ser aplicada a muchos otros campos de la informática aparte del presentado en este proyecto.

10.2 Ampliaciones

Este sistema tiene varios puntos en los que puede ser extendido y mejorado. Algunas posibles ampliaciones para el sistema son:

- Agregar la posibilidad de usar caracteres comodín y expresiones regulares en los nombres de los nodos, de esta manera se podrían hacer búsquedas de nodos que coincidan con una expresión en lugar de con un valor literal, aumentando la flexibilidad del sistema.
- Agregar operaciones aritméticas para los atributos numéricos (suma, resta, división, etc.), de manera que se puedan hacer búsquedas por valores combinados de los atributos (p.e. buscar que la suma de los salarios del último año sea mayor que un valor).
- Agregar funciones (subString, concat, etc.) que permitan operaciones más complejas así como el acceso a los nodos de texto.
- Crear implementaciones para más tecnologías de tratamiento de XML empezando por XmlReader. Esta ampliación no tiene por qué limitarse a C#, pueden crearse implementaciones para tecnologías de otros lenguajes.

11 Presupuesto

En este apartado se detallan el presupuesto de costes y el presupuesto del cliente. En la planificación se detalla que el proyecto tiene como fecha de inicio el 14 de abril del 2014 y como fecha final el 16 de junio del 2014. Los presupuestos se realizan en correlación a estas fechas, 2 meses aproximadamente de duración.

11.1 Presupuesto del cliente

Concepto	Cantidad	Coste unitario (€)	Coste total (€)
Sistema de creación automática de test para tecnologías XML	1	7000	7000
Interfaz Web para el sistema	1	2500	2500
3 meses de garantía	1	1199,35	1199,35
Subtotal			10699,35
Total (IVA 21%)			12946,21

11.2 Presupuesto de costes

En este presupuesto se ven reflejados todos los gastos en la realización del proyecto: amortización del equipo, gastos de personal y gastos de oficina.

Amortización

La amortización refleja el gasto en el equipamiento de desarrollo. Puesto que el material se utilizará en más proyectos se cobra un valor proporcional al tiempo que se tuvo asignado al proyecto respecto a la que se espera sea su vida útil.

En los equipos informáticos se puede amortizar hasta un 25% del coste anualmente.

Calculando la proporción respecto a los dos meses tendremos un 4,17% de amortización para el proyecto.

El único elemento hardware utilizado en el proyecto es un portátil Acer Aspire cuyo costo fue de aproximadamente 400€, por lo que el precio a mostrar en el presupuesto será de 16,68€.

Personal

El personal presente en el proyecto fueron los dos directores,Jordán y Cristina, que ejercieron el papel de consultores y el autor del proyecto, Simón, que ejerció los papeles de analista, diseñador, implementador y documentador.

Los salarios aplicados en el presupuesto son:

- Consultor 60€/hora.
- Analista 45€/hora.
- Diseñador 50€/hora.
- Implementador 40€/hora.
- Documentador 40€/hora.

Gastos de oficina

En este apartado se contabilizan los gastos corrientes en el espacio de trabajo: renta, luz, internet y material.

Beneficio e IVA

Al subtotal obtenido se le añade un 10% de margen de beneficio, y al total se le aplica un 21% de IVA.

Nombre	Descripción	Unidades	Coste unitario (€)	Coste total (€)
Amortización				
Portátil	Equipo de trabajo	1	16,68	16,68
Oficina				
Oficina	Espacio de trabajo	2 meses	125	250
Luz	Servicio	2 meses	50	100
Internet	Servicio	2 meses	30	60
Material	Material de trabajo	1	20	20
Personal				
Consultor Jordán	Personal	4 horas	60	240
Consultora Cristina	Personal	4 horas	60	240
Analista	Personal	40 horas	45	1800
Diseñador	Personal	60 horas	50	3000
Implementador	Personal	72 horas	40	2880
Documentador	Personal	28 horas	40	1120
Subtotal				9726,68
Subtotal+10%				10699,35
Total (IVA 21%)				12946,21

12 Referencias Bibliográficas

12.1 Referencias

- XML en el W3C. http://www.w3.org/standards/xml/core. 2014.
- XML en Wikipedia inglesa. http://en.wikipedia.org/wiki/XML. 2014.
- Ingeniería Orientada a Modelos en Wikipedia inglesa. http://en.wikipedia.org/wiki/Model-driven_engineering. 2014.
- LINQ. http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx. 2014.
- Clase XmlDocument. http://msdn.microsoft.com/es-es/library/system.xml.xmldocument(v=vs.110).aspx. 2014.
- Clase XmlReader. http://msdn.microsoft.com/es-es/library/system.xml.xmlreader(v=vs.110).aspx. 2014.
- JAX-RS. https://jax-rs-spec.java.net/. 2014.
- Eclipse. http://eclipse.org/. 2014.
- Dia. https://wiki.gnome.org/Apps/Dia/. 2014.
- GIMP. http://www.gimp.org/. 2014.
- Web Sequence Diagrams. https://www.websequencediagrams.com/. 2014.
- JQuery. https://jquery.com/. 2014.
- Google Code Prettify. https://code.google.com/p/google-code-prettify/. 2014.
- Google Charts. https://developers.google.com/chart/. 2014.
- Maven. http://maven.apache.org/. 2014.
- ANTLR. http://www.antlr.org/. 2014.
- ANTLR4 IDE. https://github.com/jknack/antlr4ide. 2014.
- Tomcat. http://tomcat.apache.org/. 2014.
- Mono. http://www.mono-project.com/. 2014.
- Spring. http://spring.io/. 2014.

- Jersey. https://jersey.java.net/. 2014.
- Validador de XHTML del W3C. http://validator.w3.org/. 2014.
- Validaddor de CSS del W3C. http://jigsaw.w3.org/css-validator/. 2014.
- WAVE Toolbar. http://wave.webaim.org/toolbar/. 2014.

13 Apéndices

13.1 Glosario y Diccionario de Datos

- **Consulta**: búsqueda de datos que se realiza sobre una fuente de información. En este proyecto la fuente de información son los documentos XML.
- **DSL**: domain-specific language, lenguaje específico de dominio.
- **Modelo**: representación abstracta de alguna cosa. En este proyecto el modelo es la estructura de objetos que representa una consulta sobre XML.

13.2 Contenido Entregado en el CD-ROM

13.2.1 Contenidos

Directorio	Contenido
./ Directorio raíz	Contiene un fichero leeme.txt explicando toda esta estructura.
./SCOT	Contiene toda la estructura de directorios del proyecto para desarrollo.
./SCOT/antlr	Contiene la gramática del DSL.
./SCOT/conf	Contiene los archivos de mensajes de la aplicación.
./SCOT/src	Contiene las principales clases e interfaces del sistema.
./SCOT/src-impl	Contiene las implementaciones de las interfaces del sistema.
./SCOT/src-antlr	Contiene las clases generadas por ANTLR a partir de la gramática.
./SCOT/WebContent	Contiene los archivos de la interfaz Web y el archivo de configuración de Spring y el de despliegue.
./instalacion	Contiene el .war de la aplicación
./documentacion	Contiene toda la documentación asociada al proyecto.
./documentacion/img	Contiene las imágenes utilizadas en la documentación.
./documentacion/uml	Ficheros que genera la herramienta con la que se han generado los diagramas UML.
./test	Contiene el archivo XML utilizado en las pruebas del sistema.

13.2.2 Código Ejecutable e Instalación

Para instalar la aplicación solo hay desplegar el .war en un servidor de aplicaciones como Tomcat.

13.2.3 Ficheros de Configuración

El fichero de configuración más importante es “application-context.xml”, situado en la carpeta “WEB-INF” del proyecto. En él no solo se indican qué implementaciones de interfaces se usan, si no que también se inyectan algunas propiedades (como los directorios de trabajo).

Los ficheros “userInterfaceText_es.properties” y “userInterfaceText_en.properties”, situados en la carpeta “conf” contienen los textos de la interfaz.

El archivo “web.xml”, situado en la carpeta “WEB-INF” contiene la descripción de despliegue del sistema.