UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Convertidor de lenguaje de marcas a formato QTI/IMS

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Gestión

Autor: Héctor Herraiz Muñoz

Director: Antonio Martí Campoy

Septiembre 2015

Resumen

(CAS) El siguiente proyecto pretende crear un programa convertidor que permita a los docentes generar, a partir de un conjunto de preguntas escritas en un sencillo lenguaje de marcas, un complejo examen en el sofisticado estándar QTI/IMS, formato utilizado por Sakai para importar exámenes completos a dicha plataforma docente.

(VAL) El següent projecte pretén crear un programa convertidor que permeta als docents generar, a partir d'un conjunt de preguntes escrites en un senzill llenguatge de marques, un complex examen en el sofisticat estándar QTI/IMS, format utilitzat per Sakai per a importar exàmens complets a aquesta plataforma docent.

(ENG) This project attempts to create a converter program to allow teachers to generate a complex IMS QTI test by using a list of markup language written questions. The resulting file will be compatible with Sakai-based learning management systems.

Palabras clave: convertidor, lenguaje de marcas, QTI, examen, preguntas, C#, XML, Sakai.

Tabla de contenidos

1. Introducción

En este apartado se tratarán la motivación del proyecto, dada por la necesidad de mejora de las herramientas docentes, y el objetivo del proyecto, que no es otro que tratar de aplicar los conocimientos y tecnologías disponibles para implementar una solución que suponga dicha mejora. Se incluye también una breve descripción del contenido de la memoria.

1.1. Motivación del proyecto.

Hoy en día las llamadas plataformas docentes o plataformas educativas virtuales están muy presentes en la educación superior. Este proyecto trata de dar solución a un problema o a la falta de una funcionalidad en una de estas plataformas.

La tarea de publicar en la red exámenes accesibles al alumnado no siempre es sencilla. La plataforma estudiada permite la introducción de las preguntas del examen una a una, lo que resulta tedioso y poco práctico. La otra opción que ofrece es importar un examen en un formato bastante complejo para el conocimiento de la mayoría de los docentes.

Surge así la necesidad de buscar otros procedimientos para publicar exámenes que permitan importar todas las preguntas a la vez, y por otra parte no supongan un esfuerzo superior a la opción existente. Un mecanismo intermedio que permita pasar de un listado de preguntas más sencillo al formato que posibilita ser importado por la plataforma.

Los llamados lenguajes de marcas suponen añadir poca información al lenguaje natural, por lo que en este caso parece apropiado su uso. Un programa que tradujese un listado de preguntas escritas en lenguaje natural y acompañadas de ciertas marcas al sofisticado formato aceptado por la plataforma virtual supondría sin duda una solución racional y relativamente sencilla.

Se plantea así la posibilidad de llevar a cabo un proyecto de implementación de un programa convertidor de lenguaje de marcas a formato QTI/IMS.

Con esta herramienta intermedia se pretende dotar a los docentes de un método para facilitarles la tarea de importar exámenes a Sakai, la plataforma de la Universitat Politècnica de València y de muchas otras. Esto les permitiría poder importar todas las preguntas de una vez sin la necesidad de aprender los entresijos del estándar QTI/IMS.

Otra motivación del proyecto fue la no posibilidad de introducir en el editor de preguntas de Sakai formatos de texto, subíndices, etc. pero actualmente el editor de preguntas ya cuenta con un editor de texto que posibilita estas funciones.

1.2. Objetivo del proyecto.

El objetivo principal del proyecto es la implementación de un programa convertidor que genere, a partir de un conjunto de preguntas escrito en un determinado lenguaje de marcas, un examen en el formato QTI/IMS.

Se decide que los datos de entrada se obtendrán de un fichero de texto plano, concretamente uno con la extensión txt. La forma en que se presentan los exámenes que siguen el estándar QTI/IMS es la de un archivo con la extensión XML.

El primer objetivo derivado será le creación de un lenguaje de marcas que sea sencillo y funcional y permita al programa convertidor identificar fácilmente la estructura de las preguntas.

Un prerrequisito del lenguaje de marcas es que debe dar soporte a tres tipos de preguntas concretos: preguntas de respuesta numérica, preguntas de rellenar los espacios en blanco, y preguntas de selección múltiple (tipo test).

Una vez definido el lenguaje de marcas, se deberá implementar una aplicación que a partir del archivo txt obtenga un archivo XML que contenga un examen con las preguntas proporcionadas y esté estructurado siguiendo el estándar QTI/IMS. Es necesario que el archivo XML generado presente compatibilidad con la plataforma Sakai, lo que hará posible importarlo y mostrarlo correctamente.

Se decide implementar el programa bajo el sistema operativo Windows para aprovechar las facilidades que ofrece la plataforma Microsoft .NET, como son la posibilidad de crear aplicaciones de formularios y la creación de archivos XML. Se decide para ello desarrollar la aplicación bajo Windows 7.

La aplicación deberá obtener a partir de las preguntas y las marcas que las acompañan la información necesaria para, usando las librerías adecuadas, generar el código QTI para un examen tipo y para cada una de las preguntas. Dicha información se almacenará en forma de archivo XML.

Para generar correctamente el código QTI, deberán estudiarse modelos de exámenes y preguntas en dicho formato. Se opta por la solución más sencilla, tomar como modelo los exámenes generados por Sakai. Con esto aumentan las posibilidades de éxito a la hora de importar a dicha plataforma los exámenes generados.

Lograr importar los exámenes generados a Sakai permitirá visualizarlos en el formato real que verían los alumnos a los que va destinado, permitiendo la detección de errores y comprobando las funcionalidades implementadas.

Así pues, las tareas se resumen en diseñar el lenguaje de marcas aceptado por el programa convertidor, estudiar los modelos de exámenes en QTI/IMS, generar el código que obtenga la información a partir de las preguntas escritas en lenguaje de marcas, e implementar la función de conversión que genere la estructura QTI que siga el estándar y contenga la información de las preguntas.

1.3. Contenido de la memoria.

La presente memoria incluye una parte de información general relacionada con las tecnologías de las que trata el proyecto, y una parte de información específica del proceso de desarrollo del programa convertidor.

El punto 2 hace un repaso a los distintos conceptos necesarios para comprender y aproximarse al contexto del problema que se pretende solucionar. Sin profundizar en exceso, se tratará de clarificar algunas tecnologías que, si bien pueden ya ser conocidas por un sector de la población, no siempre lo son por todos, y en el caso que sí lo sean esta sección puede servir de referencia en más de una ocasión.

En el punto 3 se entra de lleno a hablar del programa desarrollado. Se expone la idea inicial y se introducen las tecnologías utilizadas para llevarla a cabo. Si bien no es necesario conocer los entresijos del código para hacer uso del programa, sí lo es para aquel interesado en ampliarlo o mejorarlo. Constituye un detallado recorrido por todo el proceso de desarrollo, desde el diseño, hasta la implementación, y las pruebas.

El punto 4 es un pequeño apunte acerca del tiempo estimado que ha llevado el proceso de desarrollo del programa, y aunque no deja de ser una aproximación, da una idea de cuánto tiempo se ha invertido en cada fase del trabajo.

El punto 5 habla de la posibilidad de reutilizar el código desarrollado en un futuro para ampliarlo o mejorarlo, y da ideas sobre las vías más razonables para ello.

Finalmente, el punto 6 expone resumidamente las conclusiones que se pueden extraer una vez concluido el proyecto de implementar un convertidor plenamente funcional.

Se incluye en el Anexo I un útil manual de usuario, lectura imprescindible para una primera aproximación al funcionamiento básico y al normal uso del programa.

2. Contexto tecnológico

En este capítulo se tratarán los conceptos necesarios para conocer el ámbito del proyecto. Esto ayudará a comprender mejor su motivación y su objetivo. Desde un concepto general como es el aprendizaje electrónico, se abordarán otros sucesivamente más concretos, hasta llegar al caso que nos ocupa. Con esto no se pretende profundizar en exceso en los pequeños detalles, sino más bien que el lector se haga una idea general. Sirva así de aproximación para los no familiarizados y de referencia para el resto.

2.1. Aprendizaje electrónico.

Desde el nacimiento de Internet, herramientas como el correo electrónico han posibilitado la comunicación electrónica a distancia entre personas. Dicha funcionalidad se ha aprovechado con fines diversos, entre los que se incluyen la educación y el aprendizaje. Nace así el aprendizaje electrónico, en inglés *e-learning*.

Hoy en día con *e-learning* nos referimos a aquellas prácticas de aprendizaje que se ayudan de las redes de computadores, en especial Internet, para llevarse a cabo a distancia. Se ofertan e imparten toda clase de cursos *on-line*, incluso formación universitaria reglada, a millones de personas de todo el mundo.

Tecnologías inicialmente diseñadas con fines puramente comunicativos, como la videoconferencia, son incorporadas al *e-learning*, con lo que éste evoluciona al mismo ritmo que las tecnologías de comunicación electrónica. Es el caso de la mensajería instantánea, los sistemas de compartición de archivos, o el *streaming*.

Las ventajas que ofrece esta modalidad de aprendizaje respecto a la tradicional son varias. Por una parte, la posibilidad de que el profesor y el alumno estén en lugares distintos, lo que permite por ejemplo realizar un examen mediante videoconferencia ante un docente radicado en otro país. Por otra parte, el envío de datos hace posible compartir trabajos, artículos, lecciones, tutoriales, etc. con todo aquel que tenga acceso a la red, maximizando la difusión de contenidos docentes y educativos y propiciando la divulgación científica.

Cuando se habla de *e-learning* se puede sobreentender que necesita de dos agentes implicados, el docente y el alumno. Hoy en día existe el concepto de autoaprendizaje, propiciado por la gran fuente de información que es Internet, y que define a aquellas personas que deciden buscar información o estudiar algún tema por su cuenta, bien por ser de su interés, o por necesidad de su trabajo. Desde ese punto de vista, el autoaprendizaje mediante herramientas electrónicas podría también considerarse como *e-learning*.

Pero el ámbito tradicional del *e-learning* es el académico. Tanto en centros de formación profesional, como en universidades, la educación a distancia está ampliamente presente, con más de 35 millones de estudiantes en todo el mundo[1]. Existen incluso centros especializados en esta modalidad de enseñanza, como la Universidad Nacional de Educación a Distancia, con más de 250.000 estudiantes[2].

Además de en el ámbito académico, público y privado, el *e-learning* también está muy implantado en el ámbito empresarial. La formación como valor al alza en la nueva sociedad del conocimiento obliga a las empresas a invertir en *e-learning* para no quedarse atrás. Las redes que permiten a las compañías posicionarse visualmente a nivel internacional son también un instrumento de divulgación, reciclaje formativo y adquisición de conocimientos.

Más información:

Usuario pandahero0. *Historia y evolución de E-learning*.
https://line.do/es/historia-y-evolucion-de-e-learning/dnc/vertical

Altillo.com. *Universidades a distancia en España*.
http://www.altillo.com/universidades/universidades_espol.asp

Learning Light Ltd. *eLearning Companies in Spain*.
http://www.learninglight.com/elearning-companies-spain/

2.2. Sistemas de gestión del aprendizaje.

El aprendizaje electrónico se ha beneficiado enormemente de la evolución de la informática, valiéndose de programas y sistemas para agilizar su tarea. Los sistemas de gestión del aprendizaje, en inglés *Learning Management Systems* (LMS), han permitido automatizar infinidad de tareas y hacer llegar el *e-learning* al gran público.

El caso más típico de LMS es el de un software instalado en un servidor web, que permite a los usuarios realizar una serie de tareas, como identificarse, acceder a recursos como materiales docentes, comunicarse con otros usuarios, etc.

Con la masificación de la posibilidad de acceso a Internet, los LMS son pues la herramienta perfecta para llevar a cabo la enseñanza a distancia, si bien exigen unos mínimos conocimientos para su correcto uso. Alguien que pretende acceder a un curso *online* no tiene por qué ser un experto en informática, por lo que la tendencia es a orientar la apariencia e interfaz de los LMS al usuario medio.

[1] Asociación Nacional de Centros de E-learning y Distancia. *Quienes somos*.
http://www.anced.es/index.php/inicio/quienes-somos/74-quienes-somos
[2] UNED. *La UNED*.
http://portal.uned.es/portal/page?_pageid=93,25451643&_dad=portal&_schema=PORTAL

Estos sistemas pueden ampliar sus funcionalidades implementando nuevos módulos de software, permitiendo gestionar el proceso completo de aprendizaje, desde el acceso, pasando por el seguimiento de la evolución del alumno, hasta la evaluación y la muestra de los resultados, lo que unido al uso de bases de datos y herramientas estadísticas constituye un completo sistema de información.

Una de las claves del éxito de implantación de los LMS en el aprendizaje electrónico es que facilitan sobremanera la reutilización de la información. Alojar documentos en un servidor conectado a Internet permite a millones de personas acceder a ellos hoy, la semana que viene, o dentro de un año, con un mínimo de mantenimiento.

Existen varios tipos de LMS. Una clasificación es la que los separa en aquellos desarrollados como un producto por las empresas de software de cara a su explotación comercial, y aquellos desarrollados según la filosofía del código abierto y el software libre. Cada uno de estos dos tipos de LMS se asocia tradicionalmente a un entorno, el entorno académico en el caso de los LMS *open-source*, y el entorno empresarial en el caso de los LMS comerciales, si bien esto no es una norma sino una tendencia y también se dan el resto de combinaciones.

Ejemplos de los LMS más utilizados o implantados serían Moodle en el caso de LMS *open-source*, y Blackboard en el caso de LMS comercial[3].

Más información:

Página web de Moodle.
https://moodle.org/?lang=es

Página web de Blackboard.
http://es.blackboard.com/sites/international/globalmaster/

Capterra Inc. *Top LMS Software*.
http://www.capterra.com/learning-management-system-software/

2.3. Plataformas docentes.

Una plataforma docente es una plataforma *online* creada con el fin de ofrecer un apoyo a la docencia tradicional o un medio para la educación a distancia.

Aunque tradicionalmente las plataformas docentes se han concebido como una adaptación o implementación de un determinado *Learning Management System*,

[3] Markos Goikolea. *¿Qué son los sistemas de gestión del aprendizaje? LMS*.
http://noticias.iberestudios.com/ques-es-sistema-gestion-aprendizaje-lms/

existen otras alternativas, como utilizar un blog mas un *plugin* emulando el comportamiento de los LMS[4].

En los últimos años las plataformas docentes se han convertido en una herramienta imprescindible en ámbitos como la educación universitaria, al facilitar la tarea de profesores y alumnos. Los usos más extendidos son la compartición de recursos educativos, los foros de comunicación, la publicación de resultados académicos o el envío de tareas por parte de los alumnos.

Cada universidad elige qué LMS utiliza su plataforma docente. Aunque en los últimos años el más popular ha sido Moodle[5], algunas universidades españolas apuestan por otros LMS, como la Universidad de Vigo, que usa Claroline[6], o la Universitat Politècnica de València, cuya plataforma docente está basada en Sakai.

Además de los mencionados usos más extendidos de las plataformas docentes, hoy en día también se ocupan de integrar las nuevas herramientas tecnológicas docentes, en un continuo proceso de mejora y evolución. De cara al futuro se empieza a hablar incluso de pasar del modelo tradicional a otro basado en la gestión de metodologías más que en la gestión de contenidos[7].

En un entorno como la web, la tendencia es a la comunicación entre las distintas plataformas y tecnologías, y se busca la conectividad y la compatibilidad. Las plataformas docentes modernas siguen esta pauta y se conectan con todo tipo de dispositivos con acceso a Internet, ya sean ordenadores, tabletas, o teléfonos inteligentes.

Además de las plataformas docentes, la comunidad educativa hace uso de otras tecnologías, como los blogs, la videoconferencia, las redes sociales, o la plataforma de videos YouTube[8]. Si bien existe cierto debate respecto a si en un futuro las redes sociales podrían llegar a sustituir a las plataformas docentes, no parece una opción probable a corto o medio plazo, y es más realista pensar en su actual coexistencia.

---

[4] Miguel Florido. *Cómo crear tu propia Plataforma de Cursos Online*.
http://miposicionamientoweb.es/como-crear-tu-propia-plataforma-de-cursos-online/
[5] Merce Molist. *Moodle llena la geografía educativa española de campus virtuales*.
http://elpais.com/diario/2008/12/04/ciberpais/1228361065_850215.html
[6] Carlos Roberto. *El futuro ya está en la universidad. Así se han adaptado a la tecnología los mejores centros*.
http://www.bloglenovo.es/el-futuro-ya-esta-en-la-universidad-asi-se-han-adaptado-a-la-tecnologia-los-mejores-centros/
[7] Faraón Llorens. *Plataformas docentes: evolucionar de gestores de contenidos a gestores de metodologías*.
http://blogs.ua.es/faraonllorens/2013/08/08/plataformas-docentes-evolucionar-de-gestores-de-contenidos-a-gestores-de-metodologias/
[8] Amaya Quincoces Riesco. *Redes sociales, plataformas educativas y una nueva docencia*.
http://www.efefuturo.com/noticia/redes-sociales-plataformas-educativas-y-una-nueva-docencia/

2.3.1. El proyecto Sakai.

Sakai es un *Learning Management System* de código abierto. En su web se definen como una comunidad cuyo fin es mejorar la enseñanza, el aprendizaje, la colaboración y la investigación, así como una alternativa a los LMS comerciales[9].

Este LMS se autodenomina proyecto al estar en un continuo proceso de mejora mediante las aportaciones de los miembros de dicha comunidad, tales como nuevas herramientas multimedia, mejoras estructurales, o corrección de los errores detectados durante su uso.

Sakai se financia sobretodo mediante aportaciones de las universidades que hacen uso de él, cuyo número asciende actualmente a más de cien[10]. Entre ellas se encuentra la Universitat Politècnica de València.

Durante años el proyecto Sakai ha estado administrado por la *Sakai Foundation*. Hoy en día ha pasado a formar parte de la *Apereo Foundation*, unión de la *Sakai Foundation* con la organización sin ánimo de lucro *Jasig*[11]. Desde la *Apereo Foundation* se pretende facilitar la colaboración entre instituciones educativas y la creación colaborativa de tecnologías de aprendizaje[12].

Si se compara con otros LMS, Sakai destaca por su robustez a la hora de albergar un gran número de alumnos, de ahí su óptima acogida en las universidades. Una de sus particularidades es clasificar sus herramientas por la función que se les va a dar, ya sea ésta la docencia, la administración, la colaboración, etc.[13]

Sakai permite crear un sitio web para cada asignatura o curso. Esto permite una disposición personalizada de la información, donde cada alumno visualizará y podrá acceder exclusivamente a sus cursos o asignaturas. En cada uno de estos sitios web, se ofrecen una serie de funcionalidades tales como crear anuncios visibles por todos los alumnos, una agenda actualizable, encuestas, un foro propio, distribuir recursos multimedia, o mostrar estadísticas de los alumnos[14].

---

[9] Apereo Foundation. *How Open Works*.
https://sakaiproject.org/
[10] Juan Leyva. *¿Cómo se financian Moodle y Sakai?*
http://openlearningtech.blogspot.com.es/2009/12/como-se-financian-moodle-y-sakai.html
[11] Wikipedia. *Jasig*.
https://en.wikipedia.org/wiki/Jasig
[12] Apereo Foundation. *Sakai History. The Sakai Project Today*.
https://sakaiproject.org/sakai-history
[13] Joaquin Lucas. *Las mejores plataformas e-learning de software libre*.
http://www.lanavetic.com/las-mejores-plataformas-e-learning-de-software-libre/
[14] Pontificia Universidad Javeriana de Bogotá. *Sakai para Profesores y Tutores*.
http://www.uvirtualjaveriana.co/atico/ova/DEMOS_SK/SAKAI/SAKAI.html

Más información:

Daniel Hernandez del Peso. *Proyecto Sakai: Una plataforma de e-learning libre (I).* http://www.adictosaltrabajo.com/tutoriales/elearning-sakai/

Daniel Hernandez del Peso. *Proyecto Sakai: Una plataforma de e-learning libre (II).* http://www.adictosaltrabajo.com/tutoriales/elearning-sakai-2/

2.3.2. PoliformaT.

PoliformaT es la plataforma docente de la Universitat Politècnica de València. Es una adaptación del *Learning Management System* de código abierto Sakai, y se utiliza para todas sus asignaturas y cursos[15].

Cuenta con varios perfiles de utilización. En el perfil de Titulación se gestionan cursos de una duración determinada, tales como masters o diplomas de especialización, y sus funcionalidades incluyen organización de contenidos, anuncios, o foros. El perfil de Asignatura tiene como fin servir de apoyo a la docencia presencial, sirviéndose el profesorado de él para publicar recursos docentes o anuncios destinados a los alumnos. Un tercer perfil es el de Espacio de colaboración, destinado a grupos de usuarios[16].

PoliformaT ofrece múltiples herramientas de gestión. El apartado Recursos se utiliza para alojar contenidos, y sustituye en esta función a las antiguas *microwebs*[17]. La sección Anuncios permite publicar comunicaciones visibles por todos los miembros del grupo. Los Foros y Chats de las distintas asignaturas sirven de canal interactivo de comunicación entre profesores y alumnos. Finalmente, Exámenes (junto a Recursos la sección más utilizada en muchas asignaturas) permite, además de la publicación de exámenes, la realización *online* de dichas pruebas, ya sea de forma presencial o a distancia, de ahí su gran utilidad.

La publicación de exámenes en PoliformaT puede hacerse bien importando un fichero en estándar QTI/IMS conteniendo todas las preguntas, o bien introduciendo las preguntas una a una mediante un asistente web. Este proyecto consiste en la implementación de una herramienta para simplificar la primera opción.

Mediante el convertidor de lenguaje de marcas a formato QTI/IMS que se ha desarrollado en este proyecto, no es necesario escribir el examen en dicho formato, lo que resulta tedioso, sino que basta con escribir las preguntas en un sencillo lenguaje de marcas.

---

[15] Vicerrectorado de las TIC de la Universitat Politècnica de València. *PoliformaT.* http://www.upv.es/entidades/VTIC/info/524220normalc.html
[16] Blog de Formación Online del Centro de Formación Permanente. *Qué es PoliformaT.* http://formaciononline.blogs.upv.es/poliformat-2/poliformat/
[17] Área de Sistemas de la Información y las Comunicaciones de la UPV. *PoliformaT.* http://www.upv.es/entidades/ASIC/alumnos/727654normalc.html

A continuación se muestran imágenes del asistente web para introducir preguntas, cuyo funcionamiento es similar para los distintos tipos de preguntas, y del asistente web para importar exámenes de PoliformaT.

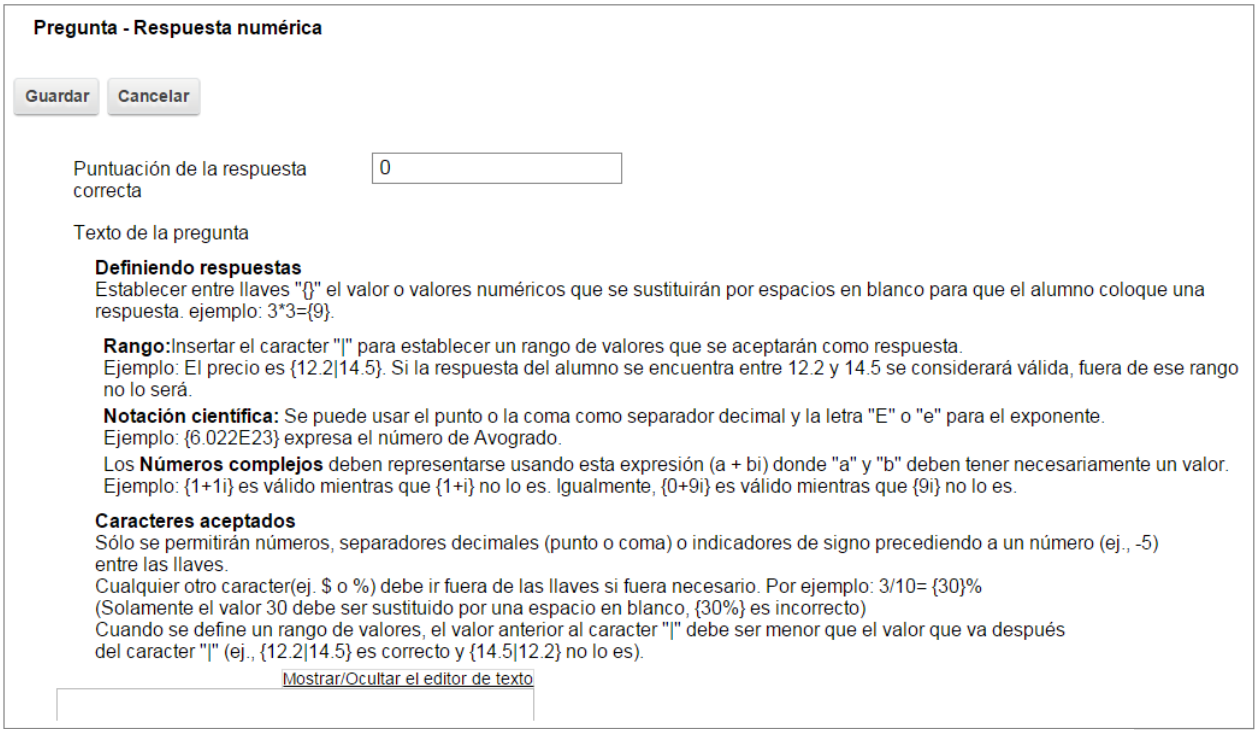

Figura 2.3.2.F1 Apariencia del asistente para introducir una pregunta de tipo respuesta numérica.

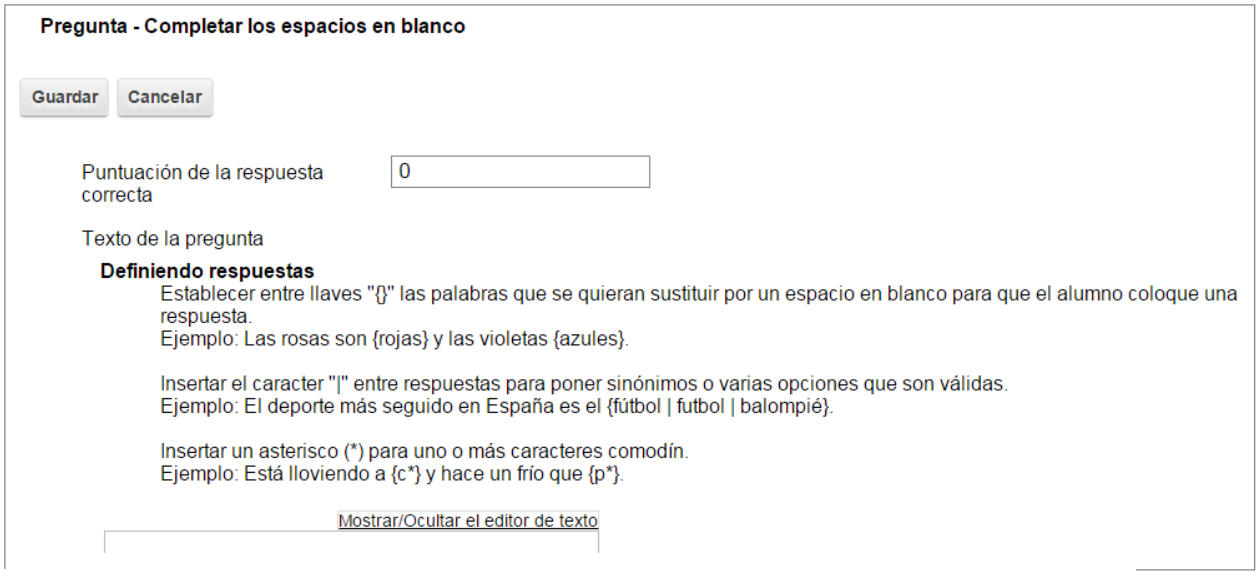

Figura 2.3.2.F2 Apariencia del asistente para introducir una pregunta de tipo rellenar los blancos.

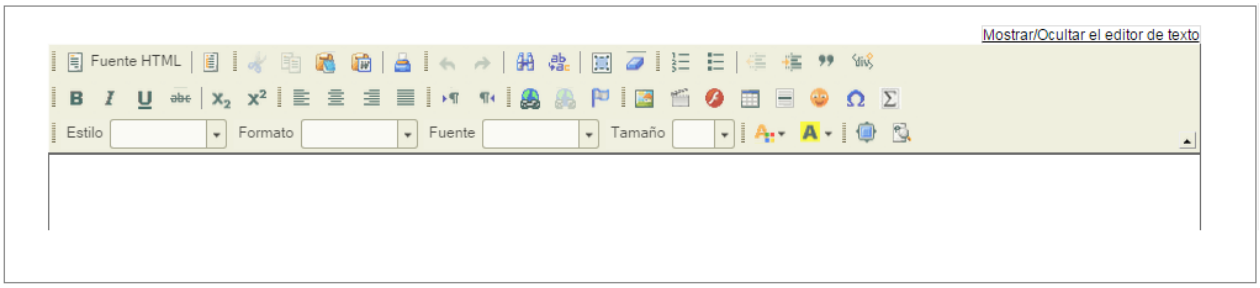

Figura 2.3.2.F3 Detalle del editor de texto del asistente de introducción de preguntas.

Figura 2.3.2.F4 Apariencia del asistente para importar un examen a PoliformaT.

2.4. Formatos de representación de exámenes.

En el proceso de digitalización de los recursos docentes que ha propiciado el avance tecnológico y la implantación del *e-learning*, los exámenes y cuestionarios son, junto a los temarios y materiales didácticos, los tipos de documentos más presentes.

Si bien la función de los temarios se reduce a su alojamiento en la web para su difusión, los exámenes tienen funcionalidades más complejas como la posibilidad de ser resueltos por los alumnos en un proceso interactivo, almacenar las respuestas enviadas, comparar las respuestas enviadas con las correctas, calcular la puntuación conseguida, o almacenar dicha puntuación vinculada a un alumno.

Al igual que en el caso de las plataformas docentes, se busca acogerse a unos estándares internacionales con el fin de poder compartir y reutilizar la información, así como para desarrollar colaborativamente tanto los estándares como las herramientas que hacen uso de ellos.

El estándar de representación de exámenes relacionado con el presente proyecto es el estándar QTI de IMS. Las siglas corresponden a *Question and Test Interoperability* e

Instructional Management Systems respectivamente, si bien en el segundo caso se usa exclusivamente el acrónimo[18].

El *IMS Global Learning Consortium* define su misión como la de proporcionar tecnologías que puedan mejorar la participación educativa. A través de su comunidad, formada por instituciones educativas y organizaciones gubernamentales, desarrolla estándares abiertos de interoperabilidad[19].

Uno de esos estándares es la especificación QTI, que permite el intercambio de datos de tipo examen y resultados del mismo entre herramientas, repositorios, plataformas docentes y sistemas que proporcionan este tipo de datos[20].

QTI es una extensión del lenguaje de marcas extensible XML, por lo que es un lenguaje de marcas. La apariencia usual de un examen que sigue la especificación QTI es la de un fichero XML con la información de las preguntas y respuestas y las marcas características de QTI que permiten identificarlas. También se incluyen en dicho archivo todo tipo de opciones y parámetros personalizables que hacen posible un gran nivel de adaptación y sofisticación pero que para el usuario no familiarizado pueden dificultar la ubicación de las preguntas y respuestas.

Como se explica en el párrafo anterior, y se observará en las imágenes de este apartado, la complejidad del estándar QTI puede resultar excesiva para redactar manualmente las preguntas una a una, y hacerlo de ese modo resultaría tedioso y poco práctico. De ahí surge la necesidad de un lenguaje intermedio como puede ser un lenguaje de marcas más sencillo que permita, mediante un programa convertidor, generar exámenes que cumplan la especificación.

A continuación se muestran algunas imágenes del aspecto que presentan distintos tipos de preguntas en un archivo en formato QTI.

[18] Wikipedia. *IMS Global. History*.
https://en.wikipedia.org/wiki/IMS_Global
[19] IMS Global Learning Consortium, Inc. *About IMS Global Learning Consortium.*
http://www.imsglobal.org/aboutims.html
[20] IMS Global Learning Consortium, Inc. *IMS Question & Test Interoperability™ Specification.*
http://www.imsglobal.org/question/

```
▼<item ident="9960885" title="Numeric Response">
   <duration/>
  ▼<itemmetadata>
   ▶<qtimetadata>...</qtimetadata>
   </itemmetadata>
  ▼<rubric view="All">
   ▼<material>
      <mattext charset="ascii-us" texttype="text/plain" xml:space="default"/>
    </material>
   </rubric>
  ▼<presentation label="FIN">
   ▼<flow class="Block">
    ▼<flow class="Block">
     ▼<material>
      ▼<mattext charset="ascii-us" texttype="text/plain" xml:space="default">
          <![CDATA[ 1+1= ]]>
        </mattext>
       </material>
     ▼<material>
        <mattext charset="ascii-us" texttype="text/plain" xml:space="default"/>
       </material>
      ▼<response_str ident="FIN00" rcardinality="Ordered" rtiming="No">
        <render_fin columns="5" fintype="String" prompt="Box" rows="1"/>
       </response_str>
     </flow>
    </flow>
   </presentation>
  ▼<resprocessing>
   ▼<outcomes>
      <decvar defaultval="0" maxvalue="1" minvalue="0" varname="SCORE" vartype="Integer"/>
    </outcomes>
   ▼<respcondition continue="Yes">
    ▼<conditionvar>
     ▼<or>
      ▼<varequal case="No" respident="FIN00">
          <![CDATA[ 2 ]]>
        </varequal>
       </or>
     </conditionvar>
     <setvar action="Add" varname="SCORE">0</setvar>
    </respcondition>
   </resprocessing>
  ▶<itemfeedback ident="Correct" view="All">...</itemfeedback>
  ▶<itemfeedback ident="InCorrect" view="All">...</itemfeedback>
 </item>
```

Figura 2.4.F1 Apariencia de una pregunta de tipo numérico en un archivo XML siguiendo QTI.

```
▼<item ident="10107159" title="Fill in Blank">
   <duration/>
  ▶<itemmetadata>...</itemmetadata>
  ▼<rubric view="All">
    ▼<material>
       <mattext charset="ascii-us" texttype="text/plain" xml:space="default"/>
     </material>
   </rubric>
  ▼<presentation label="FIB">
    ▼<flow class="Block">
      ▼<flow class="Block">
        ▼<material>
          ▼<mattext charset="ascii-us" texttype="text/plain" xml:space="default">
             <![CDATA[ blanco + negro = ]]>
           </mattext>
         </material>
        ▼<material>
           <mattext charset="ascii-us" texttype="text/plain" xml:space="default"/>
         </material>
        ▼<response_str ident="FIB00" rcardinality="Ordered" rtiming="No">
           <render_fib charset="ascii-us" columns="5" encoding="UTF_8" fibtype="String" prompt="Box" rows="1"/>
         </response_str>
       </flow>
     </flow>
   </presentation>
  ▼<resprocessing>
    ▼<outcomes>
       <decvar defaultval="0" maxvalue="0.0" minvalue="0.0" varname="SCORE" vartype="Integer"/>
     </outcomes>
    ▼<respcondition continue="Yes">
      ▼<conditionvar>
        ▼<or>
          ▼<varequal case="No" respident="FIB00">
             <![CDATA[ gris ]]>
           </varequal>
         </or>
       </conditionvar>
       <setvar action="Add" varname="SCORE">0</setvar>
     </respcondition>
   </resprocessing>
  ▶<itemfeedback ident="Correct" view="All">...</itemfeedback>
  ▶<itemfeedback ident="InCorrect" view="All">...</itemfeedback>
```

Figura 2.4.F2 Apariencia de una pregunta de tipo rellenar los blancos en un archivo XML siguiendo QTI.

2.5. Lenguajes de marcas.

Se denomina lenguaje de marcas a aquel en el cual el texto está dispuesto de tal modo que permite su manipulación por parte de un programa informático. Esto se consigue mediante anotaciones o marcas que acompañan al lenguaje corriente. Los lenguajes de marcas suelen ser legibles por los humanos al estar las anotaciones dispuestas de manera que se distingan como tales[21].

Cada lenguaje de marcas utiliza unos caracteres para delimitar sus marcas, siendo las más usadas los símbolos menor y mayor a modo de comillas (<>) o la barra invertida o contrabarra (\).

[21] Rafael Menéndez-Barzanallana Asensio. *¿Qué son los lenguajes de marcado o de marcas?* http://www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/Que-son-lenguajes-marcado.html

La popularización de los lenguajes de marcas se da a partir de los años noventa del pasado siglo con la aparición del lenguaje HTML (*HyperText Markup Language*), usado para la elaboración de páginas web. HTML utiliza las marcas <*elemento*> y </*elemento*> como etiquetas de apertura y de cierre respectivamente, y distintos tipos de comillas para marcar el valor de los atributos[22].

La relación del lenguaje de marcado HTML con el presente proyecto viene dado por el uso del programa convertidor de un fichero de texto plano como datos de entrada. Al no ser posible almacenar el texto de las preguntas y respuestas con formatos y añadiduras, se hace necesario el uso de un lenguaje de marcas, y se opta por un subconjunto de HTML al ser este lenguaje aceptado por la plataforma docente.

Para mejorar el uso de los lenguajes de marcas por parte del gran público, surgió el metalenguaje XML (*eXtensible Markup Language*), que posibilita crear etiquetas según la necesidad del usuario. En la siguiente figura se puede observar el aspecto de un documento XML sencillo[23].

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">

<Edit_Mensaje>
     <Mensaje>
          <Remitente>
               <Nombre>Nombre del remitente</Nombre>
               <Mail> Correo del remitente </Mail>
          </Remitente>
          <Destinatario>
               <Nombre>Nombre del destinatario</Nombre>
               <Mail>Correo del destinatario</Mail>
          </Destinatario>
          <Texto>
               <Asunto>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Asunto>
               <Parrafo>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Parrafo>
          </Texto>
     </Mensaje>
</Edit_Mensaje>
```

Figura 2.5.F1 Apariencia de un documento XML sencillo.

[22] Wikipedia. *HTML. Marcado HTML.*
https://es.wikipedia.org/wiki/HTML#Marcado_HTML
[23] Wikipedia. *Extensible Markup Language.*
https://es.wikipedia.org/wiki/Extensible_Markup_Language

La relación del metalenguaje de marcado XML con este proyecto es la especificación de preguntas y respuestas QTI, que es una extensión de XML. El resultado generado por el programa convertidor de lenguaje de marcas a QTI será por tanto un fichero XML.

Al igual que XML, QTI es un lenguaje de marcas. Es la complejidad de QTI lo que hace conveniente el diseño de un lenguaje de marcas más sencillo que facilite la tarea de escribir las preguntas y respuestas.

Para el uso del convertidor se ha creado un sencillo lenguaje de marcas que usar en el fichero de texto con las preguntas y respuestas. Sus marcas se asemejan a las utilizadas por HTML y XML al estar identificadas por los mismos caracteres (<>). En el apartado I.3 del Anexo I (Manual de usuario) se incluye una descripción detallada de dicho lenguaje de marcado.

Más información:

Alberto Molina Coballes. *Introducción a los lenguajes de marcas*.
http://informatica.gonzalonazareno.org/plataforma/mod/resource/view.php?id=2732

2.6. Conversores.

Los conversores o convertidores son programas informáticos que tomando como entrada un archivo en un determinado formato proporcionan un archivo equivalente en otro formato o estándar de datos.

Existen conversores multimedia, usados para obtener distintos formatos de video y audio, y otros como los conversores de texto, utilizados para generar archivos compatibles con las distintas plataformas y sistemas de información.

La función de los conversores de texto es leer el archivo origen y aplicarle las transformaciones necesarias para generar el resultado deseado de forma automática. Un ejemplo sencillo de conversor de texto sería el de un programa traductor de idioma que sustituyese cada palabra por su traducción al idioma destino. Obviamente los traductores actuales son mucho más sofisticados y analizan el contexto y otros factores para obtener un resultado óptimo.

El programa desarrollado en el presente proyecto es un convertidor de texto entre dos lenguajes de marcas, por una parte un sencillo lenguaje de marcas para escribir distintos tipos de preguntas, y por otro el estándar QTI de IMS, que es una extensión del metalenguaje de marcas XML. El programa lee el archivo de texto entrante y a partir de la información obtenida genera un archivo XML añadiendo gran cantidad de datos, como etiquetas y atributos.

Con la evolución de las tecnologías para web, surgen los conversores de formatos *online*, que evitan la necesidad de disponer de una copia del programa en nuestro ordenador, siendo suficiente una conexión a Internet y un navegador web.

Más información:

Jordi Martí. *Conversores*.
http://www.xarxatic.com/herramientas-2-0/conversores/

José M. López. *Cómo convertir archivos entre formatos por Internet*.
http://hipertextual.com/archivo/2013/06/conversores-de-archivos-online/

3. El programa

En este apartado se expone el proceso de desarrollo del programa convertidor de lenguaje de marcas a formato QTI/IMS, desde la idea inicial, las herramientas utilizadas, el diseño, la implementación, y las pruebas. Se pretende con ello hacer comprensible el funcionamiento interno del programa de una forma lo más sencilla posible, así como argumentar las decisiones tomadas a lo largo de su proceso de desarrollo.

3.1. Idea inicial.

El proyecto surge por una necesidad de algunos docentes de la Universitat Politècnica de València. La plataforma docente de la universidad, PoliformaT, basada en Sakai, ofrece la posibilidad de importar y exportar exámenes o baterías de preguntas en formato QTI/IMS mediante archivos XML, o bien introducir las preguntas una a una mediante un sencillo asistente web.

Disponer de un convertidor de lenguaje de marcas a formato QTI/IMS permite generar un archivo XML a partir de un archivo de texto plano conteniendo una batería de preguntas. Así se evitan tanto la tarea de introducir las preguntas una a una, como la de escribirlas todas en un archivo XML siguiendo el estándar QTI, que puede resultar excesivamente complicado o tedioso.

Utilizando un lenguaje de marcas también se pretendían solventar varias limitaciones del asistente web como la imposibilidad de introducir saltos de línea, formatos de texto, subíndices, letras griegas o enlaces web en las preguntas y respuestas, si bien recientemente se ha dotado al asistente web de introducción de preguntas de un editor de texto que permite estas funciones.

A la hora de definir las marcas que compondrán el lenguaje aceptado por el convertidor, se opta por usar un subconjunto de HTML para las marcas de texto, ya definidas en dicho estándar, y crear marcas específicas para identificar las preguntas, su tipo, su estructura, y su puntuación.

La idea inicial es crear un programa que lea el contenido de un fichero con la extensión txt, y genere otro con la extensión xml. El fichero xml debe de poder ser interpretado por la plataforma docente de forma correcta, y mostrar el resultado esperado. Es por ello que a la hora de generar dicho fichero con el programa convertidor, se toman como modelo los ficheros generados por el asistente web al exportar los exámenes en estándar QTI, asegurándonos así de su compatibilidad con PoliformaT.

3.2. Entorno de desarrollo SharpDevelop.

SharpDevelop es un IDE[24] libre para la plataforma Microsoft .NET. Permite desarrollar aplicaciones .NET para sistemas Windows en un entorno libre[25]. Se podría considerar a SharpDevelop como el equivalente libre al IDE comercial Microsoft Visual Studio.

Los lenguajes de programación soportados por ente entorno de desarrollo son C#, Visual Basic.Net, F#, Python, Ruby, Boo y C++[26]. El presente proyecto se ha desarrollado en lenguaje C# al haberse estudiado su uso en asignaturas como Ingeniería del Software de Gestión, donde al mismo tiempo se ha estudiado la interfaz de programación Windows Forms, soportada igualmente por SharpDevelop.

SharpDevelop supone una herramienta útil a la hora de programar aplicaciones Win32, al disponer de completado de código, diseñador de formularios, sintaxis coloreada, compilador y depurador de errores, posibilidad de ejecución del programa paso a paso o insertar puntos de parada, y en definitiva todas las opciones necesarias en un ambiente de programación avanzado[27].

A continuación se muestra mediante imágenes los aspectos más significativos y utilizados del programa en la versión utilizada, la 5.0.

[24] *Integrated Development Environment*, entorno de desarrollo integrado.
[25] F. Javier Carazo Gil. *SharpDevelop, el IDE libre para .NET cumple 10 años*. http://www.linuxhispano.net/2010/09/13/sharpdevelop-el-ide-libre-para-net-cumple-10-anos/
[26] Wikipedia. *SharpDevelop*. https://es.wikipedia.org/wiki/SharpDevelop
[27] Alberto Arroyo Raygada. *SharpDevelop a free .Net Development Environment*. http://www.elguille.info/colabora/puntoNET/aarroyo_SharpDevelop.htm

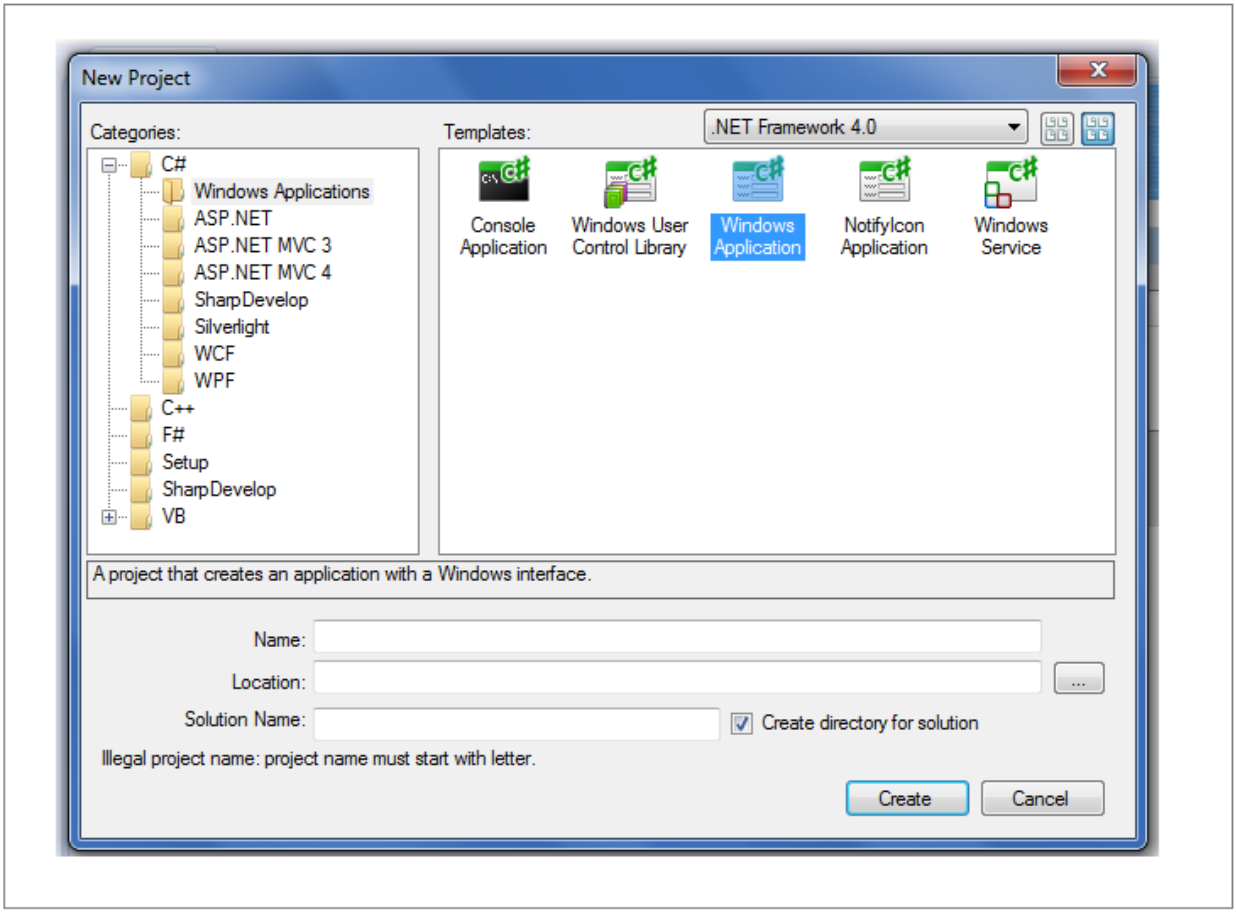

Figura 3.2.F1 Ventana de creación de nuevo proyecto en SharpDevelop.

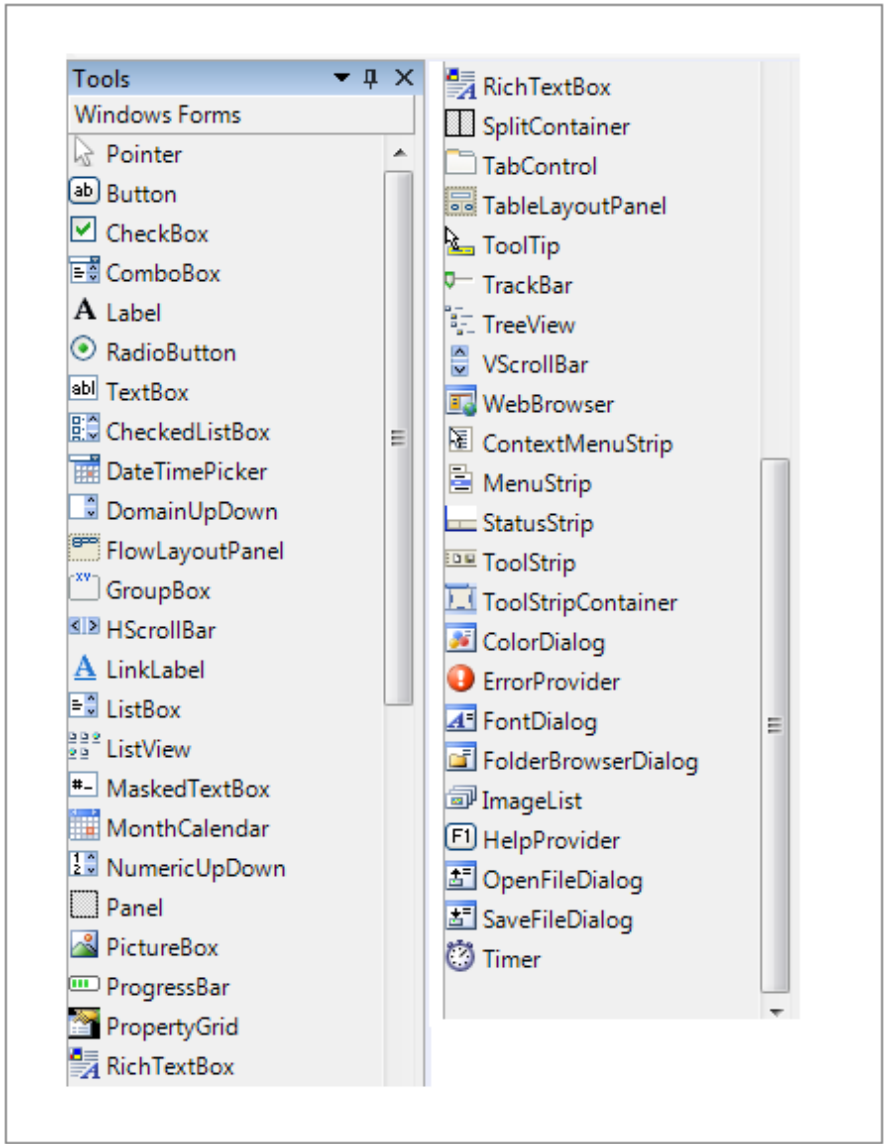

Figura 3.2.F2 Herramientas de diseño de formularios en SharpDevelop.

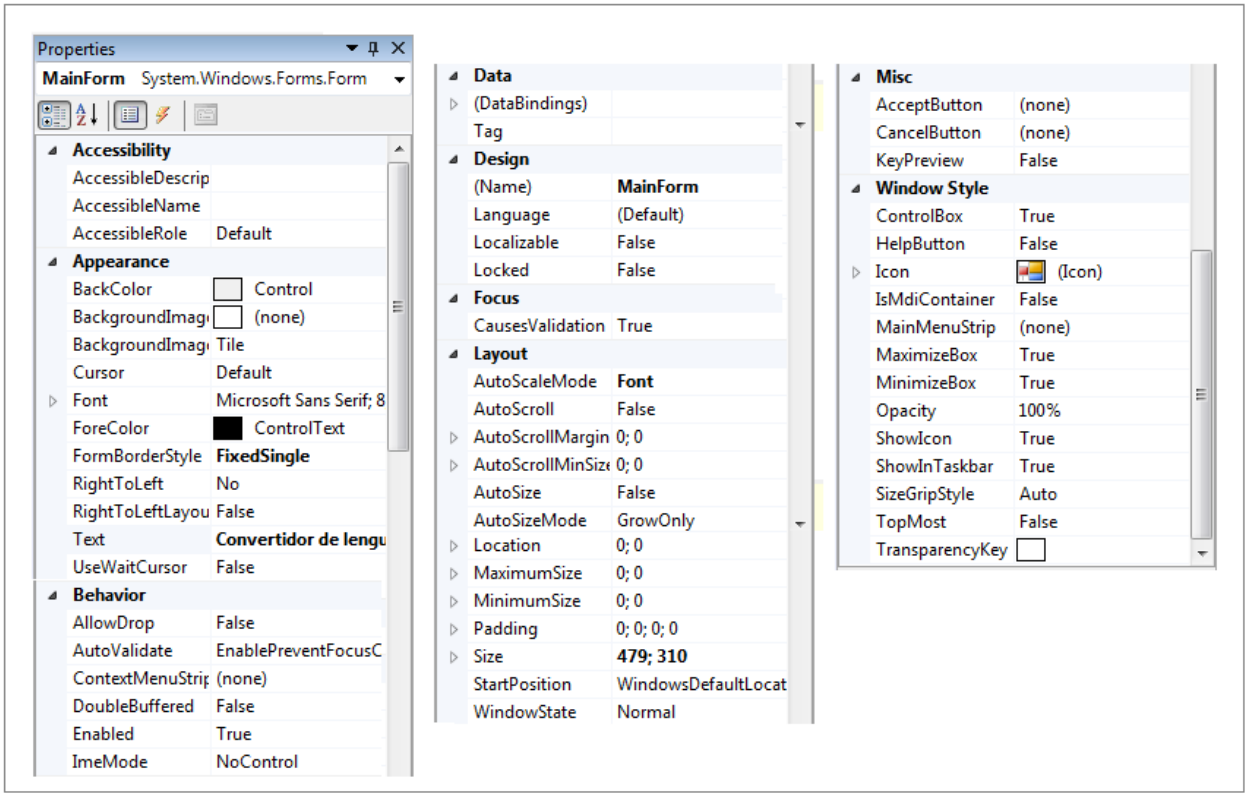

Figura 3.2.F3 Propiedades de un formulario en SharpDevelop.

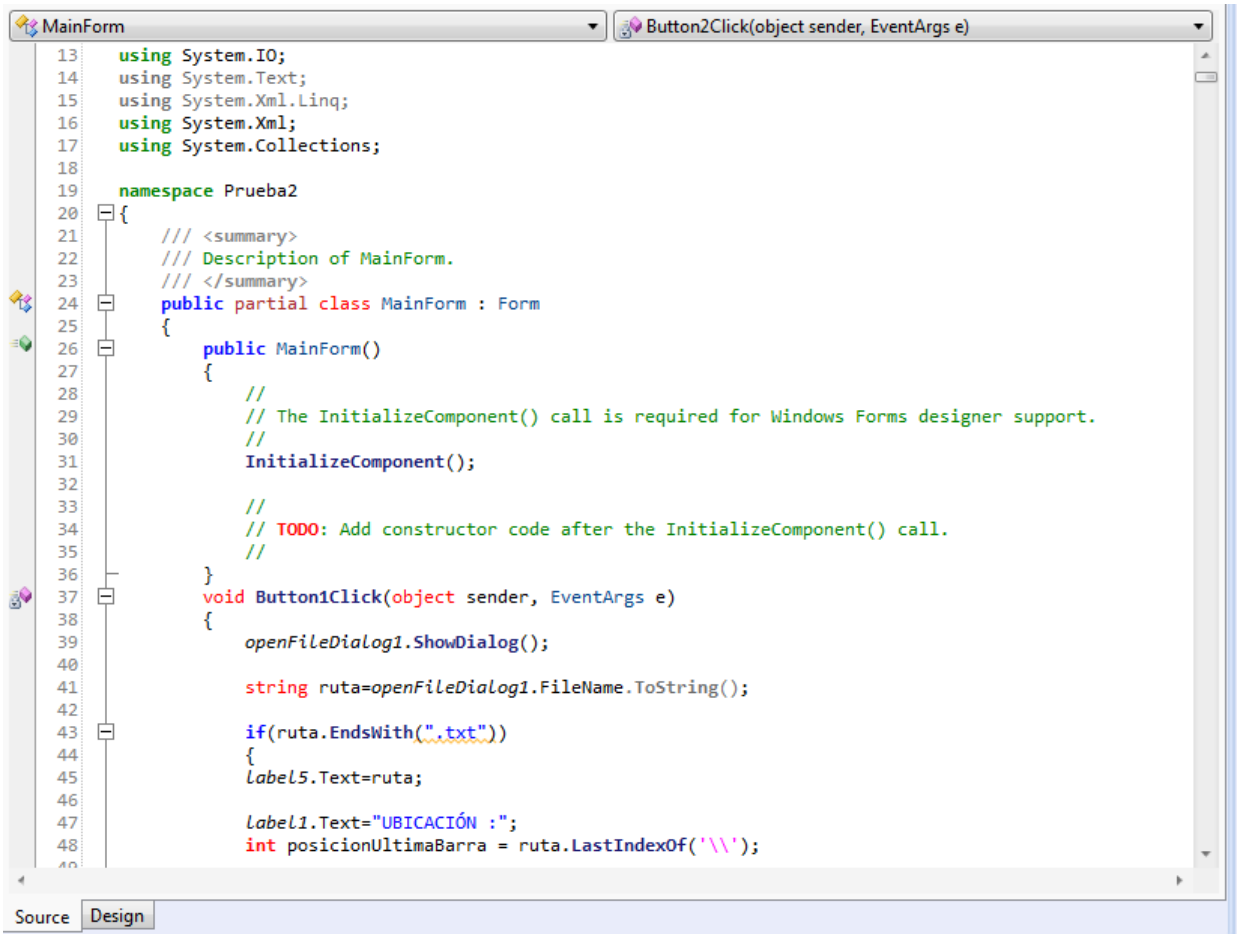

Figura 3.2.F4 Aspecto del editor de código en SharpDevelop.

Más información:

Fernando Berzal Galiano y Francisco Cortijo Bon. *La plataforma .NET*.
http://elvex.ugr.es/decsai/csharp/dotnet/index.xml

3.3. Interfaz de programación Windows Forms.

Windows Forms es la interfaz de programación de aplicaciones de formularios incluida en la plataforma .NET. Permite crear formularios (*Forms*) complejos y con un gran número de funcionalidades.

El proceso usual de creación de una aplicación de formularios de Windows comienza con el diseño de los *Forms*. Entornos de desarrollo como SharpDevelop o Microsoft Visual Studio ofrecen un editor de diseño con todas las herramientas y componentes necesarios.

Los componentes de formulario más utilizados en el presente proyecto son los botones (*Button*), las etiquetas (*Label*), los cuadros de texto (*TextBox*) y el asistente de búsqueda de archivo (*OpenFileDialog*).

Cada componente dispone de toda una serie de propiedades para personalizarlo con su comportamiento deseado, que se mostrarán al hacer clic simple sobre ellos. Al hacer doble clic sobre los componentes incorporados a nuestro *Form* se mostrará el editor de código en la línea adecuada para incorporar el código relativo a su activación o inicialización, lo que facilita esta tarea.

En definitiva Windows Forms es una interfaz de programación relativamente sencilla pero muy especializada y potente, y el método más accesible para introducirse en el mundo de la programación de aplicaciones de formularios de Windows.

Más información:

Guillermo Benitez. *Qué es Windows Forms*.
http://www.ingenieriasystems.com/2012/10/programacion-plataforma-desktop-vb-net.html

Microsoft. *Diseñar una interfaz de usuario (Visual C#)*.
https://msdn.microsoft.com/es-es/library/ms173080(v=vs.90).aspx

3.4. Diseño del programa.

En este apartado se explican las decisiones de diseño tomadas en el proceso de desarrollo del programa, tanto en el diseño de su interfaz, como en el diseño de su funcionamiento interno. En ambos casos se ha tendido a la simplicidad y a minimizar la cantidad de líneas de código, con el fin de facilitar la localización y corrección de los errores de compilación durante el desarrollo. La sencillez del funcionamiento interno permite comprenderlo con un pequeño esquema en pseudocódigo.

3.4.1. Diseño de la interfaz.

En el proceso de diseño de la interfaz se ha seguido la máxima de no dar al usuario más opciones de las necesarias. Esto reduce las posibilidades de que éste cometa algún error en la introducción de datos.

Al iniciar el programa una ventana ofrece al usuario una única opción. Según avanza la ejecución del programa, nuevas opciones aparecen en el menú en el momento en el que son necesarias. Así pues el programa consta de una única ventana que se va modificando. En el punto I.2 del Anexo I, dedicado al manual de usuario, se incluyen imágenes mostrando todas las posibles pantallas del programa.

La primera opción que se le da al usuario es la de buscar un fichero de texto en su directorio. Al apretar el botón, se muestra la pantalla del explorador de directorios, por la que el usuario puede navegar y seleccionar el archivo conteniendo las preguntas y sus correspondientes respuestas acompañadas de las marcas correspondientes.

Es necesario que la extensión de dicho fichero sea txt, lo que comprueba el programa. De no ser esa la extensión, se muestra un mensaje de error indicándolo. Una vez el programa tiene la ruta del fichero txt, se muestran el resto de opciones. A continuación se muestra el aspecto de la ventana una vez ejecutado el programa, lo que ayudará a visualizar el diseño de la interfaz fácilmente.

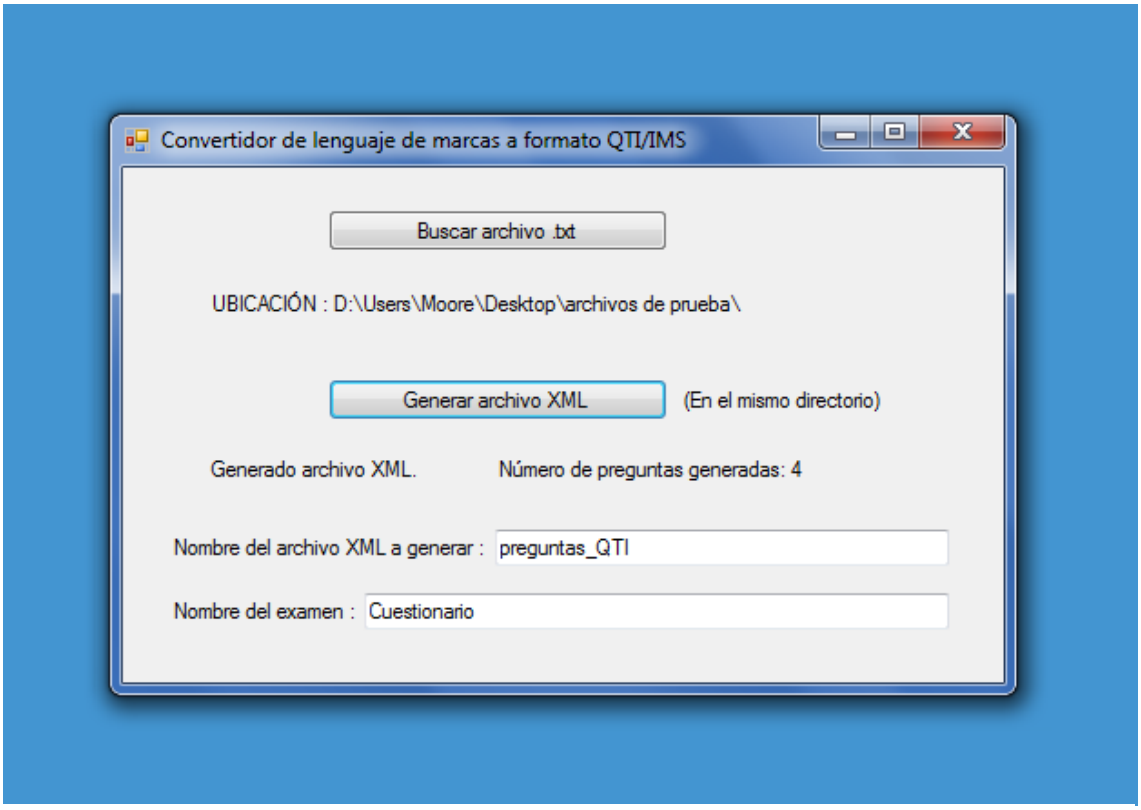

Figura 3.4.1.F1 Ventana principal una vez generado el resultado del programa.

Tras obtener la ruta del fichero txt, el programa la muestra para facilitar al usuario ubicarlo fácilmente. Al mismo tiempo se muestran dos campos de entrada de texto que permiten introducir el nombre del archivo a generar, y el nombre que se asignará en el mismo al conjunto de preguntas proporcionado. Esto es de utilidad para identificar el

examen cuanto se dispone de varios en una carpeta o cuando se importan varios a PoliformaT.

Se muestra un nuevo botón, cuya función es leer el contenido del fichero y proceder a generar la conversión a QTI. Al finalizar, se muestra un mensaje indicando que se ha generado un archivo xml.

Como se aprecia, la interfaz del programa busca la simplicidad, permitiendo al usuario primerizo familiarizarse fácilmente con ella, y evitando sobrecargarlo de opciones que no va a utilizar, dotándolo de las estrictamente necesarias.

En este apartado del diseño de la interfaz del programa podría incluirse el diseño del lenguaje de marcas aceptado por el mismo. Se incluye una descripción detallada en el apartado I.3 del Anexo I dedicado al manual de usuario. En la elección del tipo de preguntas que el programa acepta se ha elegido las de respuesta numérica, rellenar espacios en blanco y elección múltiple (tipo test) porque son los que se prestan a crear preguntas de forma automática, utilizando para ello pequeños programas o bien *scripts*. El estándar QTI contempla además otros tipos de preguntas, los cuales no maneja la versión actual del programa convertidor.

En las preguntas de tipo numérico y rellenar espacios en blanco el lenguaje de marcas diseñado permite introducir varias cuestiones en una misma pregunta. A continuación se muestra un ejemplo de este uso, incluyendo además una marca tomada del lenguaje HTML.

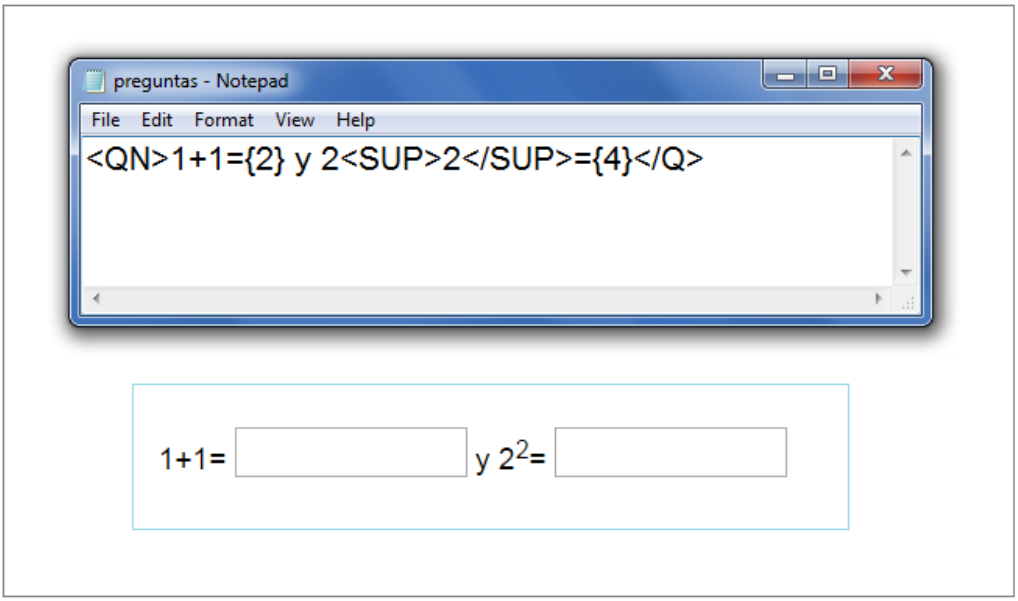

Figura 3.4.1.F2 Ejemplo de uso del lenguaje de marcas y aspecto final en PoliformaT.

3.4.2. Diseño del funcionamiento.

El programa convertidor realiza la tarea de generar un archivo XML en estándar QTI a partir de un archivo con la extensión txt conteniendo un listado de preguntas y respuestas siguiendo un sencillo lenguaje de marcas.

A continuación se muestra en pseudocódigo la idea original de funcionamiento.

Función de conversión.

Entrada: listado de preguntas y respuestas siguiendo lenguaje de marcas.
Salida: archivo XML siguiendo estándar QTI de IMS.

-Generar código QTI de los datos del examen hasta llegar a la sección de preguntas.

-Para cada conjunto de pregunta y respuesta/s

generar código QTI según el tipo de pregunta
(respuesta numérica, rellenar blancos o selección múltiple).

-Con todo el código QTI, crear archivo XML en el directorio previamente proporcionado.

Figura 3.4.2.F1 Pseudocódigo simplificado de la función de conversión.

La función de conversión se ejecutará cada vez que el usuario pulse el botón de Generar archivo XML y haya rellenado los campos de introducción de nombre del archivo XML y nombre del examen. Al ejecutarse, se volverá a leer línea a línea el contenido del archivo de texto proporcionado.

Esto permite no tener que reiniciar el programa ni volver a seleccionar la ruta del archivo de texto aunque se modifique el contenido del mismo, por ejemplo para corregir algún error en el uso del lenguaje de marcas. Bastará con hacer clic de nuevo en el botón de Generar archivo XML.

Entrando más en detalle, la función de conversión deberá identificar las marcas de las preguntas y respuestas para la creación del código QTI, haciendo uso de funciones de procesamiento de cadenas de caracteres para extraer la información. Deberá pues obtener a partir de una pregunta y su/s respuesta/s el texto correspondiente a la pregunta, y el correspondiente a su respuesta o respuestas, así como la respuesta o respuestas correctas en el caso de preguntas de tipo test (selección múltiple).

Al haberse incluido en el lenguaje de marcas las marcas de puntuación por respuesta correcta y puntuación negativa por respuesta incorrecta, el programa deberá igualmente detectar dichas marcas y almacenar su información para utilizarla posteriormente en la creación del código QTI asociado a esa pregunta.

Para empezar, se han consultado las características generales y la estructura básica del estándar en la literatura disponible. Para conocer los detalles y asegurar el funcionamiento con PoliformaT, se han realizado distintas pruebas de exportación de exámenes con distinto número de preguntas, distintos tipos de preguntas, preguntas con distinto número de respuestas, o preguntas con distinto número de respuestas correctas.

Estudiando la estructura de los archivos QTI obtenidos, se identificará la estructura tipo del código QTI de los distintos tipos de pregunta. Mediante las funciones de creación de documentos XML disponibles en la plataforma .NET, se creara el código QTI y todos sus distintos elementos.

3.5. Implementación del programa.

En este apartado se explica qué funciones concretas del lenguaje C# se han utilizado para realizar las distintas tareas del programa, tanto de la interfaz, como del convertidor. También se profundiza en la función que genera las preguntas y respuestas en estándar QTI, su comportamiento y el proceso de composición del archivo XML resultante.

3.5.1. Implementación de la interfaz.

A la hora de implementar la interfaz, se han aprovechado las herramientas de la interfaz de programación Windows Forms, como son los botones (*Button*), las etiquetas (*Label*), y los campos de introducción de texto (*TextBox*). A continuación se exponen las distintas funciones utilizadas para implementar el comportamiento de la interfaz del programa convertidor.

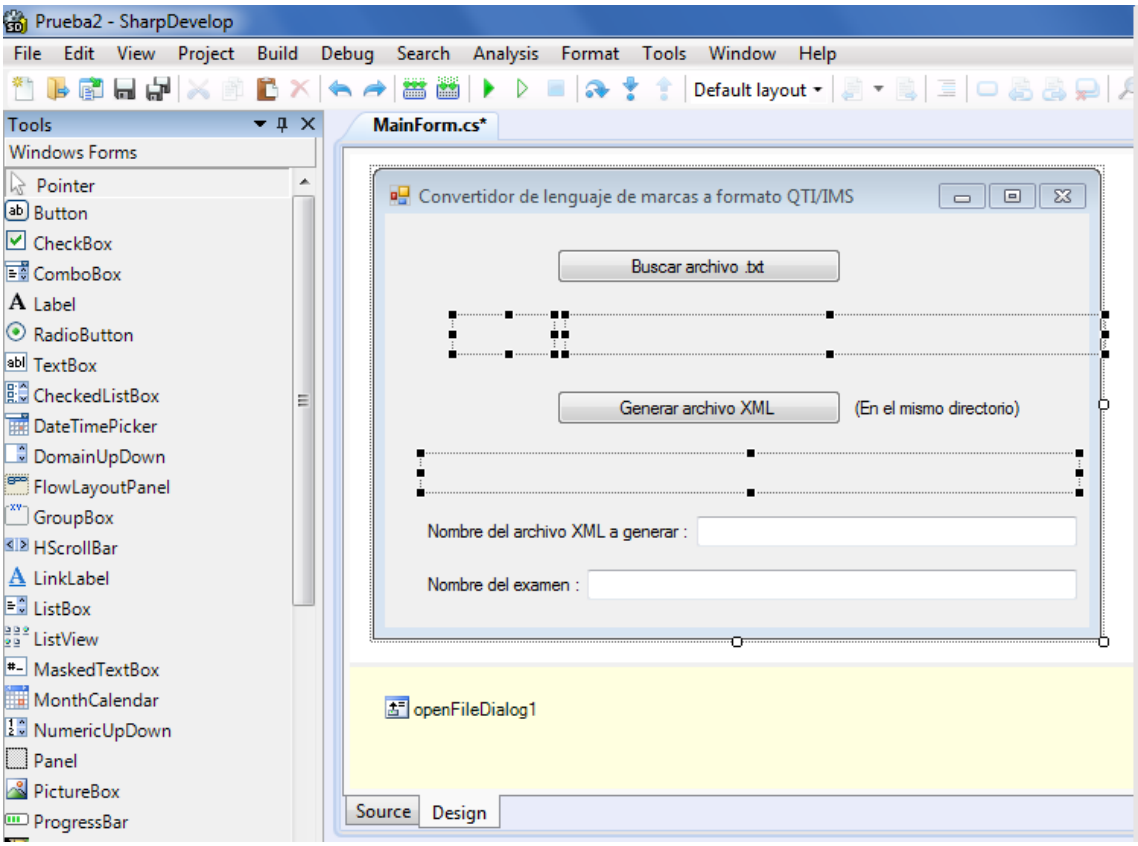

Figura 3.5.1.F1 Apariencia del entorno de desarrollo SharpDevelop durante el proceso de diseño.

Al hacer clic sobre el botón de búsqueda de archivo de texto se muestra la ventana del explorador de archivos. Para ello se utiliza una instancia del componente de Windows Forms *OpenFileDialog*, que se encarga de guardar la ruta del archivo de texto seleccionado en su atributo *FileName*.

Seguidamente se comprueba que el archivo seleccionado tenga la extensión txt, para lo cual se utiliza la función *EndsWith* de la clase *string*. En caso de no tener dicha extensión, se mostrará un mensaje de error en la misma ventana mediante el uso de etiquetas (*Label*), asignándoles a estas el texto a mostrar modificando el valor de su atributo *Text*.

Si la extensión es correcta, se mostrará la ubicación en otra etiqueta, para lo cual se recortará la ruta proporcionada por *OpenFileDialog*. Se eliminará de la ruta el nombre del archivo utilizando la función *Substring*. Recortando del mismo modo se obtendrá el nombre del archivo de texto.

A continuación el programa mostrará el resto de elementos de la ventana principal, para lo cual variará las dimensiones del formulario activo modificando su atributo *Size*. Así se mostrarán el botón para generar el archivo XML y los dos campos de introducción de texto para introducir respectivamente el nombre del archivo a generar y el nombre del examen. En ambos campos aparecerá por defecto el nombre del archivo de texto obtenido anteriormente.

Al hacer clic sobre el botón para generar el archivo XML se comprobará que los dos campos de texto no estén vacíos, primero utilizando el método *IsNullOrEmpty* de la clase *string* y luego accediendo al atributo *Text* de ambas instancias de la clase *TextBox*. Si alguno de los dos campos está vacío se mostrará en una etiqueta un mensaje de error emplazando al usuario a rellenarlo.

Si se han rellenado los dos campos, el programa continuará su ejecución leyendo el contenido del archivo de texto proporcionado línea a línea. Esto se consigue creando un elemento *StreamReader*. En la creación de dicho elemento, el método utilizado es el que requiere dos parámetros de entrada, por una parte la ruta del archivo de texto, y por otra la codificación del texto mediante el atributo *Encoding* de la clase *Text*.

Asignar la codificación de texto adecuada es clave para que nuestro *StreamReader* lea correctamente el contenido del archivo de texto, de lo contrario podría haber problemas a la hora de leer acentos o caracteres específicos de nuestro idioma. Para evitar estos problemas se asigna al atributo *Encoding* el valor *Default*. *Default* significa que se utilizará la codificación de caracteres del sistema donde se ejecuta el convertidor, por lo que si el usuario puede usar acentos y similares, no habrá problemas.

Según se van leyendo las líneas del archivo de texto, se almacenan en un elemento *ArrayList* para su posterior procesado. Lo que se hace a continuación es identificar mediante las marcas de inicio y fin de pregunta las preguntas obtenidas, entendiendo por pregunta el conjunto de pregunta y su correspondiente respuesta. La marca de inicio permite además conocer el tipo de pregunta, ya sea éste numérico, de rellenar los blancos, o de test.

Se crean dos nuevos elementos de tipo *ArrayList*. En uno se almacenarán las preguntas con su correspondiente respuesta, y en el otro, en el mismo índice, el tipo de pregunta. Con esta información más la ruta del archivo de texto para saber donde generar el archivo XML resultante, el nombre del archivo XML a generar y el nombre del examen, se invocará a la función de conversión.

La función de conversión implementada, de nombre *crearXml*, devolverá un atributo de tipo *int* conteniendo el número de preguntas generadas. Se mostrará en una etiqueta un mensaje informando de que se ha generado el archivo XML, así como del número de preguntas generadas.

3.5.2. Implementación de la conversión.

En este apartado se detalla la implementación de la función *crearXml* y la obtención de los parámetros de entrada necesarios para su correcta ejecución, desde la lectura y ordenación de la información, hasta la generación del resultado.

3.5.2.1. Lectura y ordenación de la información.

En el apartado 3.5.1 dedicado a la implementación de la interfaz del programa se ha explicado qué métodos se utilizan para obtener los parámetros de entrada de la función *crearXml*. A continuación se muestra el aspecto de la declaración del método.

```
public int crearXml(ArrayList listado, ArrayList tipoPreguntas,
                    string ruta, string nombreXML, string nombreExamen)
```

Figura 3.5.2.1.F1 Declaración de la función de conversión.

La obtención de la ruta, el nombre del archivo XML y el nombre del examen se vio en el punto anterior. A continuación se explicará con más detalle la obtención de los listados de preguntas y de tipo de preguntas a partir del fichero de texto.

Utilizando un elemento *StreamReader* se consigue leer las líneas del archivo de texto una a una, almacenándolas en un *ArrayList* auxiliar denominado *listadoPreguntas*.

Como las preguntas pueden ocupar más de una línea en el fichero de texto, especialmente las de tipo test en su disposición clásica, es necesario identificar el inicio y el final de cada pregunta para crear el listado de preguntas necesario. Mediante el método *Contains* de *string* el programa detecta la marca de inicio, que es a la vez la de tipo de pregunta.

Efectuando comparaciones de caracteres, el programa obtiene el tipo de pregunta a partir de la marca de inicio, y lo almacena en un *string* auxiliar para su posterior uso. El programa continua leyendo líneas de *listadoPreguntas* mientras queden líneas y no haya detectado los caracteres de la marca de fin de pregunta, almacenando los fragmentos de pregunta en un *string* auxiliar que irá creciendo progresivamente.

Una vez detectada la marca de fin de pregunta, se procede a almacenar la información obtenida en los dos *ArrayList* creados para tal fin. Se almacenará la pregunta ya sin marcas de inicio, fin o puntuación en el *ArrayList listadoPreguntasConSaltos*, y el tipo de pregunta en el *ArrayList listadoTipoPreguntas*.

El proceso se repite hasta que no quedan más líneas en *listadoPreguntas* en las que buscar una marca de inicio de pregunta, momento en que se invoca a la función de conversión.

La función de conversión ejecuta las instrucciones necesarias para generar el resultado final, entre las que se incluyen tareas de lectura y ordenación de la información, como el procesamiento por fragmentos de las preguntas.

El procesamiento por fragmentos de las preguntas es necesario porque en las preguntas de tipo numérico o rellenar espacios en blanco existe la posibilidad de incluir varios campos de respuesta. A continuación de muestra un ejemplo de este uso.

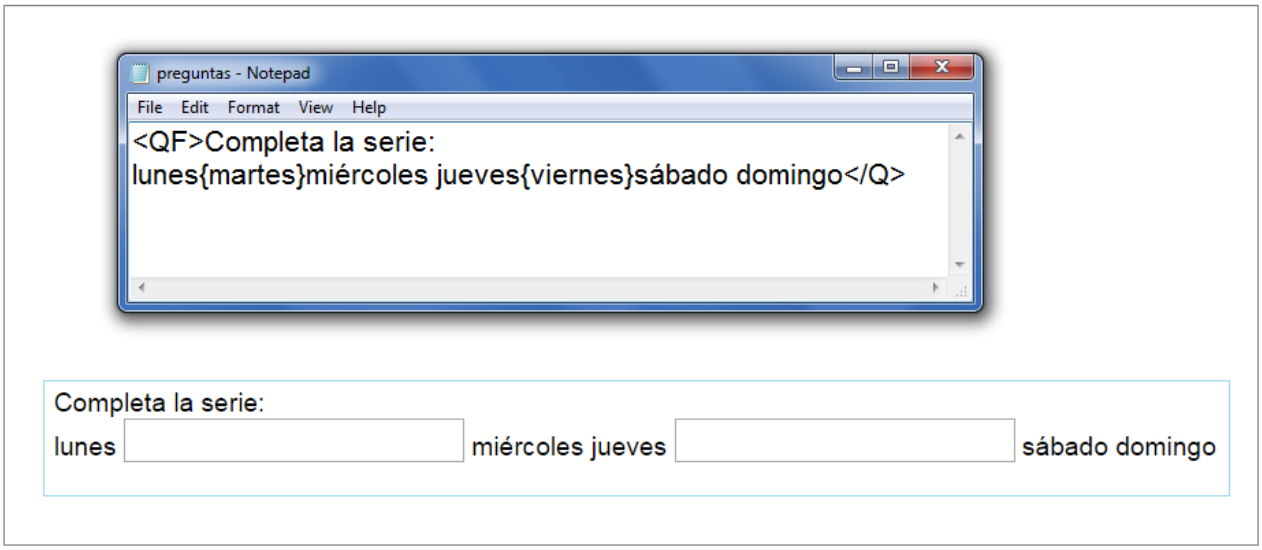

Figura 3.5.2.1.F2 Pregunta con varios campos de respuesta, usando el lenguaje de marcas y en PoliformaT.

Las instrucciones de lectura y ordenación de la información ejecutadas por la función de conversión, de nombre *crearXml*, así como el resto de sus instrucciones, se explican en el siguiente apartado.

```
int numeroDePreguntasGeneradas = crearXml(listadoPreguntasConSaltos, listadoTipoPreguntas,
                                          label2.Text, textBox1.Text, textBox2.Text);
```

Figura 3.5.2.1.F2 Invocación de *crearXml* con los dos *ArrayList* y los datos del archivo a generar.

3.5.2.2. Generación del resultado.

Una vez se invoca a la función de conversión *crearXml* con la información obtenida del texto de entrada y de los campos de introducción de datos del formulario, ya no se volverá a procesar el texto de entrada, pues ya se dispone de toda la información de las preguntas en los *ArrayList* y el resto de argumentos proporcionados a dicha función.

Para la creación del archivo XML se ha utilizado la clase *XmlDocument*, que permite añadir elementos individuales al documento, lo que es ideal para implementar el funcionamiento diseñado para el programa.

Las clases *XmlElement*, *XmlAttribute*, *XmlText* y *XmlCDataSection* permiten crear todos los elementos utilizados por los archivos en QTI estudiados.

Esto se hace por medio de los métodos de creación incluidos en la clase *XmlDocument*, *CreateElement*, *CreateAttribute*, *CreateTextNode* y *CreateCDataSection* según el caso.

Teniendo esto en cuenta, para la primera parte de la función de conversión, la generación del código QTI genérico de un examen, a falta de las preguntas, solo hay que tener en cuenta la estructura de los archivos QTI exportados por PoliformaT.

También habrá que incorporar en el código en QTI el nombre del examen, parámetro de entrada obtenido anteriormente.

```
int numeroDePreguntasGeneradas=0;

XmlDocument miXml = new XmlDocument();
XmlDeclaration xmlDeclaration = miXml.CreateXmlDeclaration( "1.0", "UTF-8", null );
XmlElement root = miXml.DocumentElement;
miXml.InsertBefore( xmlDeclaration, root );

XmlElement questestinterop = miXml.CreateElement("questestinterop");
miXml.AppendChild(questestinterop);
XmlElement assessment = miXml.CreateElement("assessment");
XmlAttribute ident = miXml.CreateAttribute("ident");
ident.Value="333283";
assessment.Attributes.Append(ident);
questestinterop.AppendChild(assessment);
XmlAttribute title = miXml.CreateAttribute("title");
title.Value = nombreExamen;
assessment.Attributes.Append(title);
```

Figura 3.5.2.2.F1 Creación del *XmlDocument* y asignación del nombre del examen.

```
▼<questestinterop>
  ▼<assessment ident="333283" title="preguntas">
```

Figura 3.5.2.2.F2 Estructura QTI creada por el código de la figura anterior.

Utilizando los métodos de creación mencionados, se implementa el código necesario para generar la estructura deseada. A continuación se muestran fragmentos de código utilizados para crear distintos objetos de XML.

```
XmlElement qtimetadata= miXml.CreateElement("qtimetadata");
assessment.AppendChild(qtimetadata);

XmlElement qtimetadatafieldauthors= miXml.CreateElement("qtimetadatafield");
qtimetadata.AppendChild(qtimetadatafieldauthors);
XmlElement fieldlabelauthors=miXml.CreateElement("fieldlabel");
XmlText fieldlabelauthorstext= miXml.CreateTextNode("AUTHORS");
fieldlabelauthors.AppendChild(fieldlabelauthorstext);
qtimetadatafieldauthors.AppendChild(fieldlabelauthors);
qtimetadatafieldauthors.AppendChild(miXml.CreateElement("fieldentry"));
```

Figura 3.5.2.2.F3 Un fragmento del código en que se hace uso del método *CreateTextNode*.

```
▼<qtimetadata>
  ▼<qtimetadatafield>
     <fieldlabel>AUTHORS</fieldlabel>
     <fieldentry/>
   </qtimetadatafield>
```

Figura 3.5.2.2.F4 Resultado del código de la figura anterior.

```
XmlElement flowmatpmmaterialmattext = miXml.CreateElement("mattext");
XmlAttribute flowmatpmmaterialmattextcharset = miXml.CreateAttribute("charset");
flowmatpmmaterialmattextcharset.Value = "ascii-us";
flowmatpmmaterialmattext.Attributes.Append(flowmatpmmaterialmattextcharset);
XmlAttribute flowmatpmmaterialmattexttexttype = miXml.CreateAttribute("texttype");
flowmatpmmaterialmattexttexttype.Value = "text/plain";
flowmatpmmaterialmattext.Attributes.Append(flowmatpmmaterialmattexttexttype);
XmlAttribute flowmatpmmaterialmattextxmlspace = miXml.CreateAttribute("xml:space");
flowmatpmmaterialmattextxmlspace.Value = "default";
flowmatpmmaterialmattext.Attributes.Append(flowmatpmmaterialmattextxmlspace);
XmlCDataSection flowmatpmmaterialmattextcdata;
flowmatpmmaterialmattextcdata = miXml.CreateCDataSection(" ");
flowmatpmmaterialmattext.AppendChild(flowmatpmmaterialmattextcdata);
flowmatpmmaterial.AppendChild(flowmatpmmaterialmattext);
```

Figura 3.5.2.2.F5 Un fragmento del código en que se hace uso del método *CreateCDataSection*.

```
▼<mattext charset="ascii-us" texttype="text/plain" xml:space="default">
   <![CDATA[ ]]>
 </mattext>
```

Figura 3.5.2.2.F6 Estructura QTI creada por el código de la anterior figura.

Seguidamente se explicará la implementación de la segunda parte de la función *crearXml*, el bucle que recorre la colección de preguntas y crea el código QTI asociado a cada una.

Antes de la declaración del bucle *for*, se crea una variable de tipo *int* que servirá para ir incrementándose y asignar valores consecutivos al atributo *id* de cada pregunta (*ítem*). En la implementación llevada a cabo su valor inicial es arbitrario.

```
int itemident=9960884;

for(int i=0; i<listado.Count; i++)
{
    itemident++;
    string item_ident=itemident.ToString();

    string preguntayrespuesta=listado[i].ToString();
```

Figura 3.5.2.2.F7 Inicio del bucle de preguntas de la función de conversión.

Antes de proceder a la generación del código QTI asociado a cada pregunta mediante los métodos de creación de elementos XML estudiados, será necesario realizar una serie de comprobaciones en cada pregunta proporcionada ya que aún pueden contener marcas. En cada iteración del bucle se realizarán estas comprobaciones sobre la pregunta antes de crear los elementos XML.

Es el caso de las marcas de puntuación positiva y puntuación negativa asociada a cada pregunta, que no son de obligado uso. Se crearán sendas variables conteniendo su valor por defecto (1 y 0 respectivamente), que posteriormente se modificaran convenientemente en el caso de usarse una de las dos marcas (la de puntuación positiva) o las dos.

Primero se inspecciona la pregunta en busca de la marca de puntuación positiva <M *valor*>, que proporciona la puntuación que se asignará a la respuesta correcta. Dicha marca no es imprescindible, pero hay que comprobar si se ha

utilizado para eliminarla del texto de la pregunta y asignar la puntuación a la variable creada anteriormente para su posterior uso en el código QTI.

Esto se hace mediante comparación de caracteres. El uso correcto de la marca <M *valor*> exige que se ubique al inicio de la pregunta, por lo que la comprobación de caracteres recorrerá la pregunta desde su inicio.

Del mismo modo se detectará la marca <MN *valor*> de puntuación negativa que se asignará a la respuesta incorrecta. Se eliminará la marca del texto de la pregunta y se asignara el valor al atributo correspondiente.

Se utiliza una marca de respuesta, definida por llaves {}, para marcar el lugar en el que aparecerán campos de introducción, que el alumno examinado deberá rellenar con las respuestas correctas. Se utilizan las llaves flanqueando a la respuesta correcta. En preguntas de tipo numérico o rellenar los blancos, la marca de las llaves puede aparecer varias veces, al poder incluirse varias cuestiones en una misma pregunta. Esto obliga a hacer un procesado por fragmentos de la pregunta, que se descompondrá en fragmentos de texto de enunciado y fragmentos de texto de respuesta.

Para la detección de la marca de llaves se utiliza el método *split* de la clase *string*. Se almacenarán en sendos *ArrayList* los fragmentos de texto que contengan el enunciado de la pregunta y los fragmentos de texto que contengan las respuestas correctas.

En el caso de preguntas de tipo selección múltiple (test) no se utiliza la marca de respuesta definida por las llaves ya que el alumno no tiene que introducir ningún texto, por lo que dichos *ArrayList* no se utilizarán en la implementación del código encargado de generar esas preguntas en QTI.

A continuación se muestra el código encargado de detectar las llaves y almacenar por separado el enunciado y las respuestas.

```
string [] partes=preguntayrespuesta.Split(new string[] { "{" },StringSplitOptions.None);

foreach(string parte in partes)
{
    if((!parte.Contains("}"))
           &&
           (!string.IsNullOrEmpty(parte)))
       {
        preguntas.Add(parte);
       }
    else
    {
        string [] partes2=parte.Split(new string[] { "}" },StringSplitOptions.None);
        respuestas.Add(partes2[0]);
        if((partes2.Length>1)
            &&
            (!string.IsNullOrEmpty(partes2[1])))
        {
            preguntas.Add(partes2[1]);
        }
    }
}
```

Figura 3.5.2.2.F8 Código encargado de detectar la marca de respuesta definida por las llaves.

Cuando se usan las llaves justo al inicio de una pregunta, la creación del código QTI asociado a los datos específicos de la pregunta deberá llevarse a cabo de manera especial, como luego se verá. Por ello, se realiza esta comprobación en lugares puntuales del código antes de llegar a ese punto, y se almacena el resultado en la variable *empiezaconblanco*.

```
if((preguntayrespuesta.ToCharArray()[0]=='{')
                &&
    (preguntayrespuesta.Contains("}")))
{
    empiezaconblanco=true;
}
```

Figura 3.5.2.2.F9 Detección de llaves de respuesta al inicio.

Las únicas marcas que quedarían por eliminar del texto de las preguntas serían las marcas específicas de las preguntas de tipo selección múltiple (test): la marca de respuesta (<op>) y la marca de respuesta correcta (<rc>), que se gestionarán mediante el código encargado de generar ese tipo de preguntas. No se incluyen aquí las marcas HTML utilizadas para formatos de texto, subíndices, enlaces, etc. que serán correctamente interpretadas por Sakai y por lo tanto no es necesario eliminar del texto de las preguntas.

El siguiente paso del bucle de preguntas es la creación de los elementos XML asociados a la pregunta que maneja en esa iteración. A continuación se muestra la creación del elemento *item*, que define a las preguntas en QTI, y la asignación de sus atributos principales: *ident*, que define su número de identificación, y *title* para definir su tipo.

Para detectar las preguntas de tipo selección múltiple con una o con varias respuestas correctas, que da lugar a la creación de dos tipos diferenciados de *item*, se contará el número de veces que aparece la marca <rc> en una pregunta, para lo cual se accede a la variable *preguntayrespuesta*, que toma en cada iteración del bucle de preguntas el valor contenido en el correspondiente índice del *ArrayList listado*, que contiene el texto completo de las preguntas.

```
XmlElement item = miXml.CreateElement("item");
XmlAttribute id = miXml.CreateAttribute("ident");
id.Value = item_ident;
item.Attributes.Append(id);
XmlAttribute titl = miXml.CreateAttribute("title");

if(tipoPreguntas[0].ToString()=="N") titl.Value = "Numeric Response";
if(tipoPreguntas[0].ToString()=="F") titl.Value = "Fill in Blank";
if(tipoPreguntas[0].ToString()=="M")
{
    string [] partesrc=preguntayrespuesta.Split(new string[] { "<rc>" },StringSplitOptions.None);
    if (partesrc.Length>2)titl.Value = "Multiple Correct";
    else titl.Value = "Multiple Choice";
}
if(tipoPreguntas.Count>1)tipoPreguntas.RemoveAt(0);

item.Attributes.Append(titl);
section.AppendChild(item);
```

Figura 3.5.2.2.F10 Código encargado de declarar una pregunta y asignarle sus atributos principales.

Lo siguiente que hará el programa es continuar creando el código QTI de la pregunta manejada por el bucle. Primero se crearán los elementos genéricos de ese tipo de pregunta, y seguidamente el código específico a partir de los datos proporcionados de enunciado y respuesta/s.

Para crear los elementos del tipo de pregunta se intercalarán fragmentos de código que cree elementos comunes a todos los tipos de pregunta con fragmentos de código que cree elementos específicos a un tipo. Para hacerlo, se incluirán comparaciones (*if*) antes de los bloques específicos para un tipo.

Se seguirá el orden de elementos marcado por el archivo XML tomado como modelo para ese tipo de pregunta. A continuación se muestra un ejemplo donde se crean elementos QTI solo en el caso de que el tipo de pregunta sea *Multiple Choice* o *Multiple Correct*.

```
if((titl.Value.ToString()=="Multiple Choice")
   || (titl.Value.ToString()=="Multiple Correct"))
{
XmlElement qtimetafieldrandomiz= miXml.CreateElement("qtimetadatafield");
qtimeta.AppendChild(qtimetafieldrandomiz);
XmlElement fieldlabelrandomiz=miXml.CreateElement("fieldlabel");
XmlText fieldlabelrandomiztext= miXml.CreateTextNode("RANDOMIZE");
fieldlabelrandomiz.AppendChild(fieldlabelrandomiztext);
qtimetafieldrandomiz.AppendChild(fieldlabelrandomiz);
XmlElement fieldentryrandomiz=miXml.CreateElement("fieldentry");
XmlText fieldentryrandomiztext= miXml.CreateTextNode("false");
fieldentryrandomiz.AppendChild(fieldentryrandomiztext);
qtimetafieldrandomiz.AppendChild(fieldentryrandomiz);
}
```

Figura 3.5.2.2.F11 Uso de comparación para crear código QTI según el tipo de pregunta.

Una vez creado el código QTI genérico relativo al tipo de pregunta manejado, se procederá a crear el código específico con los datos proporcionados. Primero se creará la variable de tipo *string primerapregunta* para almacenar el primer fragmento del enunciado de la pregunta.

Para asignarle el valor a dicha variable, se consultará la variable *empiezaconblanco* anteriormente obtenida para, en el caso de que la pregunta empiece por llaves de respuesta, asignarle a *primerapregunta* el valor "" (cadena vacía), lo que es necesario para construir correctamente el código QTI asociado a los datos de la pregunta.

```
string primerapregunta;
if(empiezaconblanco)primerapregunta="";
else
{
    primerapregunta=preguntas[0].ToString();
    preguntas.RemoveAt(0);
}
```

Figura 3.5.2.2.F12 La variable *primerapregunta*.

Para crear los datos de la pregunta en QTI, se ejecutará uno de dos bloques de instrucciones distintos: el primero cuando se trate de preguntas de tipo numérico o rellenar los blancos, o el segundo cuando sean preguntas de selección múltiple de una o más respuestas correctas. A continuación se explican ambos casos.

En los archivos QTI los fragmentos de enunciado se almacenan en el bloque *presentation*, y las respuestas en el bloque *resprocessing*, de la correspondiente pregunta (*item*).

Estudiando distintos archivos XML QTI generados por PoliformaT, se identifica el patrón que siguen los elementos del bloque *presentation* en las preguntas de tipos numérico y rellenar los blancos, tales como los bloques *material* (con o sin un elemento *CDATA* con texto del enunciado) y los bloques *response_str* que identifican a las respuestas cuya información habrá de contener el bloque *resprocessing*.

A los elementos *response_str* en necesario dotarlos de un atributo identificador, por lo que se creará la variable *contadoridentificador* para ir incrementando en uno el identificador asignado a los sucesivos *response_str*. También se utilizará dicha variable como contador de respuestas insertadas.

Para la creación de los elementos que contendrán los datos de la pregunta, su texto se procesó previamente por fragmentos, guardando los fragmentos de enunciado y las respuestas en sendos *ArrayList* denominados respectivamente *preguntas* y *respuestas*.

El patrón identificado comienza por un bloque *material* con un elemento *CDATA*, que contendrá el primer fragmento del enunciado de la pregunta o estará "vacío" si la pregunta empieza por un campo de respuesta (el mencionado caso de utilización de llaves de respuesta al inicio).

A continuación se insertará otro elemento *material* con *CDATA* conteniendo el siguiente fragmento de enunciado; de no haber más fragmentos de enunciado, se insertará en su lugar un elemento *material* sin *CDATA*. Después se insertará un bloque *response_str* relativo a la respuesta anterior al último fragmento de enunciado insertado.

Estos dos últimos pasos, insertar *material* con *CDATA* del siguiente fragmento de enunciado (o sin *CDATA* si no hay más enunciados), e insertar *response_str* de la respuesta que precede a ese fragmento de enunciado, se repetirán mientras queden fragmentos de respuesta.

Finalmente, y solo si no se ha incluido, se incluirá un elemento *material* sin *CDATA*.

La generación del código QTI que cumpla este patrón se hace mediante el uso de un bucle *while*.

Se comprobará el estado de la variable anteriormente declarada *empiezaconblanco* (que indica si estamos en el caso de utilización de llaves de respuesta al inicio) pues al haberse insertado un fragmento de enunciado "vacío" al inicio de la pregunta para crear el código QTI, el número de *response_str* creados deberá ser de uno menos que en el caso normal.

También se creará una nueva variable de tipo *bool* que indica si nos encontramos en la última iteración del bucle (*ultimaronda*) y no hay más fragmentos de pregunta que insertar.

Para detectar si se ha incluido un elemento *material* sin *CDATA* se creará la variable de tipo *bool insertadomaterialsincdata*.

Seguidamente se muestra un esquema explicativo con código y pseudocódigo.

```
int contadoridentificador=0;

bool ultimaronda=false;

bool insertadomaterialsincdata = false;

while((preguntas.Count>0)||(!ultimaronda && !empiezaconblanco))
{
if(preguntas.Count==0) ultimaronda=true;

if(contadoridentificador<respuestas.Count)
{
   (insertar material)

   if(!ultimaronda)
   {
     (añadir CData al material)
     preguntas.RemoveAt(0);
   }
   else insertadomaterialsincdata=true;

  (insertar response_str)

   contadoridentificador++;

 } // fin if(contadoridentificador<respuestas.Count)

} //fin while

if(!insertadomaterialsincdata)
{
  (insertar material sin CData)
}
```

Figura 3.5.2.2.F13 Generación del bloque *presentation* en preguntas numéricas y de blancos.

```
<presentation label="FIN">
  <flow class="Block">
    <flow class="Block">
      <material>
        <mattext charset="ascii-us" texttype="text/plain" xml:space="default">
          <![CDATA[ ]]>
        </mattext>
      </material>
      <material>
        <mattext charset="ascii-us" texttype="text/plain" xml:space="default">
          <![CDATA[ +1=2 y 2+ ]]>
        </mattext>
      </material>
      <response_str ident="FIN00" rcardinality="Ordered" rtiming="No">
        <render_fin columns="5" fintype="String" prompt="Box" rows="1"/>
      </response_str>
      <material>
        <mattext charset="ascii-us" texttype="text/plain" xml:space="default">
          <![CDATA[ =4 ]]>
        </mattext>
      </material>
      <response_str ident="FIN01" rcardinality="Ordered" rtiming="No">
        <render_fin columns="5" fintype="String" prompt="Box" rows="1"/>
      </response_str>
      <material>
        <mattext charset="ascii-us" texttype="text/plain" xml:space="default"/>
      </material>
    </flow>
  </flow>
</presentation>
```

Figura 3.5.2.2.F14 Ejemplo del bloque *presentation* en una pregunta de tipo numérico.

El siguiente paso es generar el bloque *resprocessing* de la pregunta de tipo *Numeric Response* o *Fill in Blank*. El bloque constará de un elemento *outcomes* que incluirá la puntuación positiva y la puntuación negativa asociadas a la pregunta (los valores obtenidos mediante las respectivas marcas o bien los valores por defecto) y un número de elementos *respcondition* igual al número de respuestas.

En el caso especial de que la pregunta empiece por un campo de introducción de respuesta, se eliminará la respuesta inicial, que estará “vacía”. Esto es debido al código usado para obtener las respuestas a partir de la marca de llaves, que en este caso concreto introducirá una primera respuesta conteniendo la cadena vacía. La siguiente figura explica la creación del bloque *resprocessing*.

```
(insertar outcomes)
//incluye asignación puntuaciones positiva y negativa

contadoridentificador=0;

if(empiezaconblanco)respuestas.RemoveAt(0);

while(respuestas.Count>0)
{
  (insertar respcondition)
  contadoridentificador++;
  respuestas.RemoveAt(0);
}
```

Figura 3.5.2.2.F15 Generación del bloque *resprocessing* en preguntas numéricas y de blancos.

```
▼<resprocessing>
  ▼<outcomes>
     <decvar defaultval="0" maxvalue="1" minvalue="0" varname="SCORE" vartype="Integer"/>
   </outcomes>
  ▼<respcondition continue="Yes">
    ▼<conditionvar>
      ▼<or>
        ▼<varequal case="No" respident="FIN00">
           <![CDATA[ 2 ]]>
         </varequal>
       </or>
     </conditionvar>
     <setvar action="Add" varname="SCORE">0</setvar>
   </respcondition>
  ▼<respcondition continue="Yes">
    ▼<conditionvar>
      ▼<or>
        ▼<varequal case="No" respident="FIN01">
           <![CDATA[ 4 ]]>
         </varequal>
       </or>
     </conditionvar>
     <setvar action="Add" varname="SCORE">0</setvar>
   </respcondition>
 </resprocessing>
```

Figura 3.5.2.2.F16 Ejemplo del bloque *resprocessing* en una pregunta de tipo numérico.

Sólo quedaría añadir a la pregunta (*item*) dos bloques *itemfeedback* relativos al comportamiento adicional por pregunta correcta o incorrecta respectivamente. El contenido de dichos bloques es genérico por lo que su generación es trivial.

Veamos ahora cómo el programa convertidor genera en QTI los datos específicos de una pregunta de tipo selección múltiple en cualquiera de sus dos modalidades implementadas (una respuesta correcta y varias respuestas correctas).

Primero se lleva a cabo la ordenación de la información necesaria. A partir del *string primerapregunta* se extraerán, por una parte, el enunciado de la pregunta, y por otra, las respuestas. Al identificarse las respuestas por letras mayúsculas, se toma como máximo número de respuestas el número de letras disponibles, 26. Se creará un vector de variables *bool* de longitud 26 en el que se almacenará si la respuesta asociada a ese índice en el *ArrayList* de respuestas es correcta o no. También habrá que eliminar la marca de respuesta correcta. A continuación se muestra el código.

```
string [] partespreguntatest = primerapregunta.Split(new string[] { "<op>" },StringSplitOptions.None);
string enunciadopreguntatest=partespreguntatest[0];

ArrayList respuestastest=new ArrayList();

bool [] soncorrectas=new bool[26];

for(int p=0; p<soncorrectas.Length; p++)
{
    soncorrectas[p]=false;
}

if(partespreguntatest.Length>0)
{
    for(int z=1; z<partespreguntatest.Length; z++)
    {

            respuestastest.Add(partespreguntatest[z].ToString());
    }

    for(int q=0; q<respuestastest.Count; q++)
    {
        string respuestatest=respuestastest[q].ToString();
        if(respuestatest.Contains("<rc>")) soncorrectas[q]=true;
    }

    for(int x=0; x<respuestastest.Count;x++)
    {
        string mirespuestatest=respuestastest[x].ToString();
        if(mirespuestatest.Contains("<rc>"))
            respuestastest[x]=mirespuestatest.Substring(4);
    }
}
```

Figura 3.5.2.2.F17 Código encargado de la ordenación de la información de las preguntas tipo test.

El primer bloque que contiene datos específicos de la pregunta tipo test es el *presentation*. Tras insertar un bloque *material mattext* con un *CData* conteniendo el enunciado de la pregunta, se inserta un bloque *material matimage*.

A continuación habrá que generar un bloque *response_lid* conteniendo los datos de las respuestas. Para ello se crea una variable que almacenará el número de respuestas, *numresptest*, y una variable de tipo *char* que se irá incrementando para asignar sucesivas letras mayúsculas a las respuestas, la variable *letraresp*.

Igual que sucedía con los otros tipos de preguntas, se estudia la estructura de los archivos QTI XML generados por PoliformaT para identificar un patrón, el cual se rige por los siguientes pasos.

Para cada respuesta, se realizará la inserción de un elemento *response_label* conteniendo un elemento *material mattext* (con un *CData* con el texto de la respuesta). Para las cuatro primeras respuestas se añadirá un elemento *material matimage*.

De haber menos de cuatro respuestas, se insertará el número necesario de elementos *response_label* para llegar a cuatro en total (respuestas y añadidos). Cada *response_label* añadido de este modo contendrá un elemento *material mattext* (sin *CData*) y un elemento *material matimage*.

Seguidamente se insertarán cuatro bloques *response_label* sin atributo *ident*, conteniendo cada uno un elemento *material mattext*.

Así se concluiría la construcción del bloque *presentation*. A continuación se muestra un esquema explicativo con código y pseudocódigo.

```
(insertar material mattext con CData enunciado)
(insertar material matimage)

int numresptest=respuestastest.Count;

char letraresp='A';

for(int w=0;w<numresptest; w++)
{
(insertar response_label)
    (añadir material mattext con CData respuesta)
    if(w<4)
    {
      (añadir material matimage)
    }
    respuestastest.RemoveAt(0);
}

int cuatromenosnumresp=4-numresptest;

for (int vv=0; vv<cuatromenosnumresp; vv++)
{
  (insertar response_label)
      (añadir material mattext)
      (añadir material matimage)
}
for(int v=0; v<4; v++)
{
  (insertar response_label sin ident)
      (añadir material mattext)
}
```

Figura 3.5.2.2.F18 Generación del bloque *presentation* en preguntas tipo test.

```
▼<presentation label="Resp003">
  ▼<flow class="Block">
    ▼<material>
      ▼<mattext charset="ascii-us" texttype="text/plain" xml:space="default">
          <![CDATA[ Es un reptil ]]>
        </mattext>
      </material>
    ▶<material>...</material>
    ▼<response_lid ident="MCSC" rcardinality="Single" rtiming="No">
      ▼<render_choice shuffle="No">
        ▼<response_label ident="A" rarea="Ellipse" rrange="Exact" rshuffle="Yes">
          ▼<material>
            ▼<mattext charset="ascii-us" texttype="text/plain" xml:space="default">
                <![CDATA[ ñandú ]]>
              </mattext>
            </material>
          ▶<material>...</material>
          </response_label>
        ▼<response_label ident="B" rarea="Ellipse" rrange="Exact" rshuffle="Yes">
          ▼<material>
            ▼<mattext charset="ascii-us" texttype="text/plain" xml:space="default">
                <![CDATA[ lagartija ]]>
              </mattext>
            </material>
          ▶<material>...</material>
          </response_label>
        ▼<response_label ident="C" rarea="Ellipse" rrange="Exact" rshuffle="Yes">
          ▶<material>...</material>
          ▶<material>...</material>
          </response_label>
        ▼<response_label ident="D" rarea="Ellipse" rrange="Exact" rshuffle="Yes">
          ▶<material>...</material>
          ▶<material>...</material>
          </response_label>
        ▶<response_label rarea="Ellipse" rrange="Exact" rshuffle="Yes">...</response_label>
        ▶<response_label rarea="Ellipse" rrange="Exact" rshuffle="Yes">...</response_label>
        ▶<response_label rarea="Ellipse" rrange="Exact" rshuffle="Yes">...</response_label>
        ▼<response_label rarea="Ellipse" rrange="Exact" rshuffle="Yes">
          ▼<material>
              <mattext charset="ascii-us" texttype="text/plain" xml:space="default"/>
            </material>
          </response_label>
        </render_choice>
      </response_lid>
    </flow>
  </presentation>
```

Figura 3.5.2.2.F19 Ejemplo del bloque *presentation* en una pregunta de tipo test.

A continuación se tratará la generación del bloque *resprocessing* en preguntas de los dos tipos de preguntas test.

Primero se insertará un bloque *outcomes* con la puntuación positiva y la puntuación negativa asociadas a la pregunta.

A continuación se insertarán 26 elementos *respcondition conditionvar*, tantos como el máximo número posible de respuestas, es decir el máximo número de letras mayúsculas distintas. En las preguntas test con una única respuesta correcta el atributo *continue* de los elementos *respcondition* tomará el valor *No* para los cuatro primeros elementos y *Yes* para el resto. En las preguntas test con varias respuestas correctas *continue* siempre tomará el valor *Yes*.

En cada elemento *respcondition conditionvar* se especificarán dos elementos *displayfeedback*. En un primer elemento *displayfeedback* se especificará si la respuesta asociada a esa letra es correcta o incorrecta. Si hay menos de 26 respuestas, el resto de elementos hasta llegar a ese número asignarán el valor *InCorrect* a la respuesta asociada a esa letra.

En un segundo elemento *displayfeedback*, en las preguntas test con una única respuesta correcta se almacenará la letra asociada a esa respuesta seguida de un 1 para los elementos correspondientes a respuestas de la pregunta, y como mínimo para los cuatro primeros elementos. Para el resto el valor será *D1*.

En las preguntas test con varias respuestas correctas ese mismo atributo tomará el valor *AnswerFeedback* en los cuatro primeros elementos y *D1* en el resto, y se añadirá un elemento *CData* (con el valor *null*) en las respuestas propias de la pregunta.

Con esto acabarían las instrucciones necesarias para generar el bloque *resprocessing*. A continuación se muestra un esquema explicativo.

```
(insertar outcomes)
//incluye asignación puntuaciones positiva y negativa

char abcdario = 'A';

for (int abc=0; abc<26; abc++)
{
  (insertar respcondition)

  if(abc>3)itemresprrespccontinue.Value = ("Yes");
  else itemresprrespccontinue.Value = ("No");
  if(titl.Value.ToString()=="Multiple Correct")itemresprrespccontinue.Value = ("Yes");

  (insertar conditionvar)
  (insertar primer displayfeedback)
      if(soncorrectas[abc]==true)itemresprrespcdispfeedlinkrefid.Value="Correct";
      else itemresprrespcdispfeedlinkrefid.Value="InCorrect";
  (insertar segundo displayfeedback)
      if(abc>3)
      {
          char auxcharresp='D';
          itemresprrespcdispfeed2linkrefid.Value = auxcharresp.ToString()+"1";
      }
      else
      {
          itemresprrespcdispfeed2linkrefid.Value=abcdario.ToString()+"1";
          if(titl.Value.ToString()=="Multiple Correct")
              itemresprrespcdispfeed2linkrefid.Value="AnswerFeedback";
      }

      if((titl.Value.ToString()=="Multiple Correct")
         &&(abc<numresptest))
      {
        (insertar CData con el valor null)
      }

  abcdario++;

} //fin for
```

Figura 3.5.2.2.F20 Generación del bloque *resprocessing* en preguntas de los tipos test.

El resto de elementos del código QTI a generar para estos tipos de preguntas test es genérico (no contiene información de los datos específicos de la pregunta), por lo que no plantea dificultad.

Una vez ejecutadas las instrucciones para crear los datos específicos de las preguntas, ya sea el bloque de instrucciones destinado a las de tipo numérico y de rellenar espacios en blanco o el bloque destinado a las de los tipos de selección múltiple, se incrementa la variable *numeroDePreguntasGeneradas*.

Solo quedará usar los parámetros de entrada de la función *crearXml* relativos a la ruta del archivo a generar y al nombre del archivo para salvar todo el *XmlDocument* creado en un archivo. A continuación se muestra es código que ejecuta estas tareas finales.

```
            numeroDePreguntasGeneradas++;

        } //fin for(int i=0; i<listado.Count; i++)

        string miPath=ruta;
        int posicionUltimaBarra = miPath.LastIndexOf('\\');
        miPath=miPath.Substring(0,posicionUltimaBarra);
        miPath=miPath+'\\'+nombreXML+".xml";

        miXml.Save(@miPath);

        return numeroDePreguntasGeneradas;

    } //fin crearXml
```

Figura 3.5.2.2.F21 Código de las últimas instrucciones del método *crearXml*.

3.6. Evaluación y pruebas.

La fase de evaluación y pruebas del programa se ha llevado a cabo durante todo el proceso de desarrollo.

Desde el principio la idea ha sido aproximarse a las herramientas de diseño y desarrollo de Windows Forms y a los métodos de la clase *XmlDocument* buscando primero resultados pequeños pero funcionales, para más tarde añadir nuevas funcionalidades al programa.

Cada nueva funcionalidad se sometió a un proceso de evaluación y pruebas para comprobar su correcto funcionamiento. A continuación se enumeran en orden cronológico las principales mejoras introducidas en el proceso de desarrollo tanto del lenguaje de marcas como del programa.

-Limitar los tipos de preguntas aceptados por el lenguaje de marcas y por el programa a tres: respuesta numérica, rellenar espacios en blanco, y selección múltiple. Esto fue una decisión tomada por la facilidad para crear automáticamente baterías de preguntas de esos tres tipos.

-Dada la dificultad de conseguir modelos fiables de exámenes en formato QTI en los que basarse para generar la conversión, se opta por acceder a las funcionalidades de Sakai para importar, generar y exportar exámenes en este formato, asegurando esto además que los futuros exámenes generados a semejanza serán compatibles con la plataforma.

-Utilización de la marca de respuesta definida por llaves ({}) utilizada por la plataforma Sakai, lo que resultará en mayor comodidad para los usuarios del programa ya familiarizados con esa plataforma. En un principio se utilizaba el signo de igual para detectar la pregunta y la respuesta (el primer tipo de preguntas implementado fue el numérico), lo que no permitía agrupar varias preguntas numéricas en una única estructura de pregunta. Con la marca de respuesta definida por llaves se solventó este problema.

-Mejora de la interfaz para mostrar el número de respuestas generadas por el programa, para lo que se crea una variable para almacenarlo internamente e ir incrementándolo uno a uno según se genere correctamente cada pregunta.

-Mejora de la interfaz de adquisición de datos para incluir la posibilidad de cambiar tanto el nombre del fichero a generar como el nombre del examen a generar, para lo cual se añaden sendos campos de introducción de texto al formulario principal y se crean variables internas para almacenar esos datos.

-Implementación de las marcas de inicio y fin de pregunta. En un principio se utilizaban los saltos de línea para identificar las distintas preguntas, lo que imposibilitaba el uso de saltos de línea en el cuerpo de las preguntas. Con el uso de las marcas de inicio y fin se solucionaba esta cuestión. La marca de inicio se implementó como marca de inicio y de tipo de pregunta, dotándola de esa función con lo que no era necesario disponer de dos marcas separadas.

-Implementación de la marca de puntuación de pregunta. Se definió esta nueva marca y se comprobó su correcto funcionamiento importando a la plataforma PoliformaT exámenes generados con el programa convertidor.

-Implementación de las marcas específicas para las preguntas de tipo selección múltiple (tipo test). Se crean las marcas de respuesta y de respuesta correcta, haciendo innecesario en este tipo de preguntas el uso de la marca de respuesta definida por llaves. Se comprueba que la plataforma acepte y muestre satisfactoriamente las preguntas de este tipo generadas por el programa convertidor.

-Implementación de la marca de puntuación negativa por respuesta incorrecta. Se amplia la funcionalidad conseguida con la implementación de la marca de puntuación añadiendo esta nueva marca de puntuación negativa, siendo necesario modificar el código del programa.

-Conseguir que el programa convertidor acepte acentos y demás caracteres particulares de nuestro idioma. Tras comprobar que la plataforma docente gestionaba correctamente las marcas HTML presentes en el cuerpo de las preguntas, se plantea si existiría un método más sencillo para poder utilizar acentos y caracteres particulares en el archivo de texto tomado como datos de entrada del programa. Tras revisar el código encargado de leer el contenido del archivo y documentarse, se modifica el código para añadir a la función encargada de leer el archivo un parámetro de *encoding*. Al asignarle a dicho parámetro el valor del *encoding* por defecto del sistema, se consigue que la función lea correctamente los caracteres necesarios.

Como se aprecia, desde el inicio del proceso de desarrollo hasta la consecución del mismo el programa ha ido modificándose para adaptarse a las necesidades de funcionamiento.

Para comprobar que los exámenes generados por el programa convertidor se mostrasen correctamente, eran importados a PoliformaT y previsualizados, lo que daba una simulación veraz. Esto ha sido especialmente de gran ayuda para comprobar que la asignación de la puntuación de la pregunta se realizaba correctamente.

No se puede obviar la gran ayuda que supone también el compilador de SharpDevelop para detectar los posibles errores en el código, así como su función para introducir *breakpoints* y ejecutar el programa paso a paso. Realmente ha resultado muy útil y no se ha echado en falta ninguna funcionalidad de Microsoft Visual Studio, utilizado tiempo atrás.

En definitiva, se ha contado con las herramientas necesarias para evaluar y probar tanto los avances en el diseño y la implementación del programa, como los resultados generados por el mismo, destacando la posibilidad de acceder a las funcionalidades de generación e importación de exámenes de Sakai.

Respecto a las pruebas realizadas una vez conseguida una versión estable del programa, se ha constatado que incrementar el número de preguntas del archivo entrante no aumentará el tiempo de respuesta del convertidor excesivamente. Si para un archivo de texto conteniendo unas pocas preguntas el tiempo de conversión es casi instantáneo, para un archivo conteniendo cien preguntas el tiempo es inferior a los dos segundos. Después se ha probado con un archivo conteniendo mil preguntas, siendo igualmente el tiempo de generación del archivo XML resultante inferior a los dos segundos.

Así pues, al aumentar significativamente el tamaño de la batería de preguntas a convertir, el programa no aumenta significativamente su tiempo de respuesta, garantizándose así una experiencia de uso fluida.

4. Estimación de costes

Como dato que forma parte de la recapitulación del trabajo realizado, se incluye aquí una estimación de costes del proceso de desarrollo de *software* del programa.

Esto es interesante para hacerse una idea del tiempo empleado en cada fase del proceso de desarrollo, y de cuánto habría costado (económicamente hablando) el proyecto de haberse realizado en un entorno empresarial estándar.

Si bien a veces es complicado concretar con detalle el tiempo exacto empleado en las distintas fases, se propone un intervalo temporal aproximado, con lo que el hipotético coste económico sería también un intervalo.

A continuación se detalla en una tabla el tiempo aproximado dedicado a cada fase de desarrollo. En estos tiempos se incluye el tiempo empleado en labores de documentación, ya sea acerca del estándar QTI, de las librerías de la plataforma .NET, o del programa SharpDevelop.

Después se calculará con esos datos y una estimación del sueldo medio el coste económico hipotético del proyecto.

Diseño de la interfaz.	25-30 horas
Diseño del funcionamiento interno.	45-50 horas
Implementación de la interfaz.	35-40 horas
Implementación de la lectura y ordenación de la información.	100-110 horas
Implementación de la generación del resultado.	150-165 horas
Evaluación y pruebas (durante todo el proceso de desarrollo).	75-85 horas
Total	430-480 horas

Figura 4.F1 Tiempo aproximado dedicado a cada fase del proceso de desarrollo.

Tomando como sueldo medio de un programador base entre diez y quince euros por hora trabajada, se obtiene el coste hipotético del programa.

430-480 horas a 10-15 euros/hora	4.300-7.200 euros

Figura 4.F2 Coste hipotético del programa.

5. Trabajos futuros

En este apartado se tratará el tema de la reusabilidad del programa, y sus posibilidades de ampliación. En un mundo en el que el *software* libre, las herramientas de distribución gratuita y en general el intercambio libre de información son cada vez más utilizados, es conveniente contemplar la posibilidad de poner a disposición de los posibles interesados el programa de cara a su mejora.

La primera idea de mejora o ampliación que viene a la mente es la posibilidad de incorporar otros tipos de preguntas aceptados por Sakai aparte de los escogidos para el convertidor (respuesta numérica, rellenar los blancos y selección múltiple). No sería difícil modificar el código para aceptar nuevas marcas de tipo de pregunta. Tanto el bloque encargado de leer y organizar la información de las marcas como el responsable de la generación de la pregunta (*item*) en QTI podrían ampliarse, y mediante un condicional que detectase el nuevo tipo de pregunta se podría agrupar el nuevo código. Todo dependerá de la necesidad de los usuarios, de su preferencia por uno u otro tipo de pregunta o su adecuación a la tarea docente que se realice.

Las posibilidades de modificación de los atributos de los exámenes en QTI IMS son muy grandes. Al igual que se ha contemplado la posibilidad de modificar el nombre del examen se podría implementar, bien mediante campos de entrada de datos en el formulario principal o mediante nuevas marcas, la modificación de otros atributos.

Estos atributos podrían ser atributos del examen, atributos de la sección del examen conteniendo las preguntas, o atributos de las preguntas. De nuevo serán las necesidades del usuario final las que dicten hasta qué punto será necesario profundizar en los entresijos del estándar QTI IMS, si bien con un poco de paciencia las estructuras y atributos más típicos serán fácilmente identificados.

Al haberse conseguido implementar una aplicación relativamente sencilla y funcional, no parece descabellado pensar que algunos docentes sabrán apreciar su utilidad práctica. La posibilidad de generar en QTI todas las preguntas de un examen “de golpe” sin necesidad de introducirlas una a una en la plataforma docente es sin duda un avance en lo que respecta al tiempo empleado en realizar la tarea.

Serán pues los docentes quienes demanden o no ampliar las funcionalidades del programa, o incluso se planteen hacerlo ellos mismos, o encargar la tarea a alguien más relacionado con el mundo de la programación.

Sean todos ellos libres de estudiar el programa y someterlo a las pruebas deseadas para ponderar su utilidad, así como de modificarlo y ampliarlo si creen que eso puede ayudarles a lograr una herramienta que les sea útil en su trabajo.

6. Conclusiones

Una vez completado el proceso de desarrollo de la aplicación, se hace necesario recapitular acerca de si se han conseguido los objetivos propuestos.

El método para resolver el problema planteado fue inicialmente utilizar los conocimientos de programación adquiridos para, ayudándose de las tecnologías disponibles, familiarizarse con la generación de archivos XML.

Después, se aplicaría esa experiencia para generar archivos QTI/IMS. Para ello, era necesario disponer de modelos adecuados para comprender la estructura de estos archivos. Aquí Sakai ha jugado un papel importante.

Con esos dos elementos, conocimiento de la generación de archivos XML y conocimiento de las estructuras específicas del estándar QTI/IMS, no ha sido difícil obtener un programa generador que proporcione el resultado deseado.

Posiblemente el método empleado para ello no haya sido el más minucioso, si bien se ha visto que no era necesario estudiar en profundidad el estándar QTI/IMS para implementar el convertidor, sino que bastaba con conocer las estructuras básicas y sobre todo las relativas a los elementos tratados, ciertos datos (no todos) del examen y de las preguntas y respuestas. Esto ha facilitado el proceso de desarrollo, que de otro modo podría haberse complicado en el estudio pormenorizado del estándar, lo que habría supuesto sin duda mucho más tiempo.

Definir correctamente el lenguaje de marcas empleado en la adquisición de datos previa a la generación del resultado también ha sido clave. Era necesario que el lenguaje de marcas fuese sencillo y funcional. La sencillez del lenguaje de marcas empleado será un punto importante a la hora de que los usuarios lleguen a aceptar el programa como una herramienta útil.

El uso de la plataforma Microsoft .NET ha permitido crear una aplicación que funciona en la práctica totalidad de las versiones del sistema operativo Windows, lo que sin duda ayudará a su aceptación. También ha permitido hacer uso de sus librerías y módulos, como *System.Windows.Forms* o *System.Xml*, sin los cuales habría sido imposible llevar a cabo el proyecto en los términos en los que se ha hecho.

La no necesidad de instalación del programa convertidor al tratarse de un archivo ejecutable que no precisa de librerías externas es un añadido más para valorar su sencillez de uso, así como su reducido espacio necesario en disco.

En definitiva se ha logrado implementar un programa convertidor de lenguaje de marcas a formato QTI/IMS que cumple con los objetivos propuestos, existiendo espacio para futuras mejoras en función de las necesidades de los usuarios. Sirva la experiencia para alentar a los docentes a continuar demandando mejoras en sus plataformas docentes, pues con las tecnologías adecuadas y conocimientos de programación no es difícil implementar soluciones adaptadas a sus necesidades.

Anexo I. Manual de usuario

I.1 Contexto y finalidad.

Actualmente la educación *on-line* y la digitalización de los medios docentes están ampliamente extendidas, y surgen herramientas y plataformas que agilizan tareas como publicar en la red exámenes y cuestionarios al alcance de los alumnos.

Con la globalización, la tendencia mundial es crear estándares que permitan compartir e implantar dichas tecnologías en cualquier parte del planeta, posibilitando el intercambio de recursos de enseñanza y la mejora colectiva de las herramientas informáticas relacionadas.

Pero algunos de esos estándares son de una complejidad no al alcance del usuario medio, por lo que aparecen programas intermedios e interfaces para facilitar la tarea de introducir la información en los sistemas. Un ejemplo serían las plataformas utilizadas por las universidades para que los profesores publiquen exámenes en Internet. Dichos asistentes, al estar orientados a la web, no siempre ofrecen al usuario todas las funcionalidades deseadas.

Este programa trata de ser una ayuda para aquellos docentes no familiarizados con los entresijos del estándar QTI (Questions & Test Interoperability) de IMS que en un momento dado necesiten un asistente para generar exámenes más sencillo o funcional que el que tienen disponible. El objetivo del programa es generar un archivo en dicho formato de preguntas a partir de un fichero de texto plano.

Para ello, se utiliza un lenguaje de marcas, esto es unas indicaciones que acompañan a las preguntas y respuestas y permiten al programa identificar su tipo y sus atributos en el acto. Con esto se consigue un lenguaje de menor complejidad que el estándar QTI, significativamente más técnico u opaco para el usuario no familiarizado.

Esto nos permite crear, mediante pequeños programas informáticos, Microsoft Excel, o las funciones de combinar datos de Microsoft Office, colecciones de preguntas sencillas (de tipo numérico, de completar los espacios en blanco, de tipo test) y utilizar el presente programa para convertirlas en su conjunto a formato QTI (versión 1.2), cuando en ocasiones las plataformas docentes obligan a introducir las preguntas una a una, lo que es una desventaja evidente por requerir de más tiempo y esfuerzo.

Otra ventaja que ofrece el programa convertidor es la posibilidad de utilizar marcas ya existentes en el lenguaje HTML, fácilmente reconocibles por las plataformas web, lo que nos permite utilizar saltos de línea, texto en negrita o cursiva, letras griegas, subíndices, o incluso enlaces web. Teniendo en cuenta que algunas de las plataformas docentes más utilizadas no permiten la utilización de estas marcas a la hora de introducir las preguntas, salta a la vista la utilidad práctica de la aplicación. Cabe decir que recientemente la plataforma PoliformaT ha añadido al asistente web de introducción de preguntas un editor de texto que permite estas funciones.

I.2 Funcionamiento básico del programa.

El programa convertidor de lenguaje de marcas a formato QTI/IMS consta de un único fichero ejecutable, sin DLLs ni archivos de soporte, por lo que no necesita instalación. Diseñado e implementado como una aplicación de Windows Forms, su apariencia e interfaz resultarán familiares a los usuarios acostumbrados a los sistemas operativos basados en ventanas.

En lo que respecta a los requisitos del sistema, aunque el desarrollo se ha llevado a cabo bajo Windows 7, se ha comprobado su correcto funcionamiento en otros sistemas operativos, como Windows XP y Windows 8, lo que permite al usuario elegir bajo qué sistema ejecutar la aplicación. El espacio requerido en disco es despreciable al ser el tamaño del ejecutable de menos de 1 MB.

Al hacer doble clic sobre el icono de la aplicación, se muestra el menú principal, que da al usuario la única opción de buscar en su sistema de ficheros un archivo de texto plano.

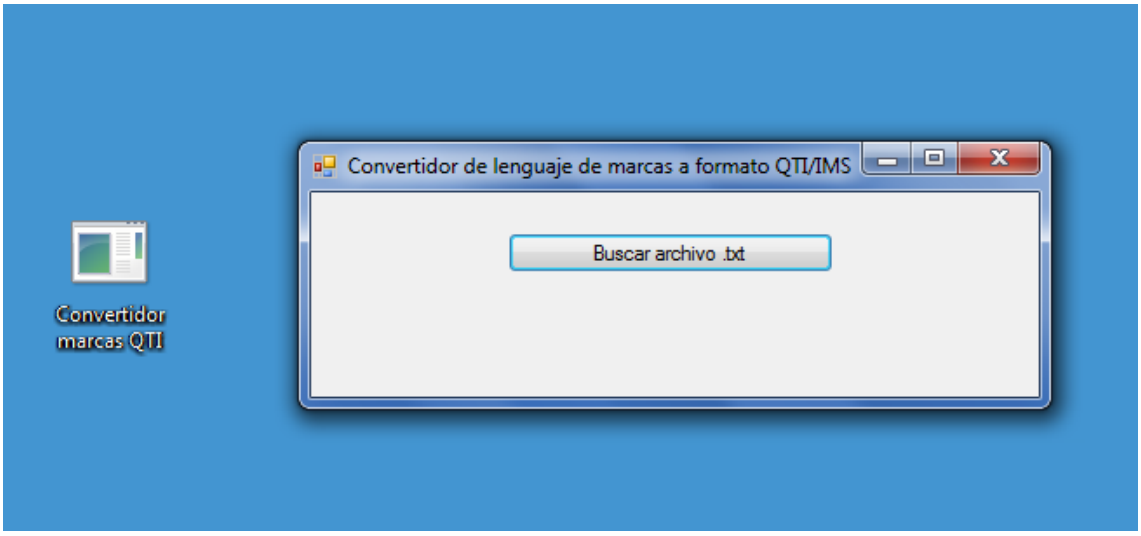

Figura I.2.F1 Icono de la aplicación y menú principal.

Al hacer clic sobre el botón de “Buscar archivo .txt”, se muestra la ventana de exploración de directorios y se debe seleccionar el archivo de texto plano que contenga una o más preguntas (con sus respuestas) que se quiere convertir a formato QTI, que deben adecuarse al lenguaje de marcas que utiliza el programa (ver apartado I.3). Si no se selecciona un archivo con la extensión “.txt”, se muestra un mensaje de error. El usuario podrá entonces volver a hacer clic en el

botón de búsqueda de archivo y seleccionar el archivo de texto adecuado. Si el archivo es válido, aparecen en el menú principal nuevos campos de introducción de datos, donde el usuario podrá si así lo desea modificar el nombre del archivo XML en formato estándar QTI que se creará, y el nombre que en dicho archivo se asignará al examen o colección de preguntas.

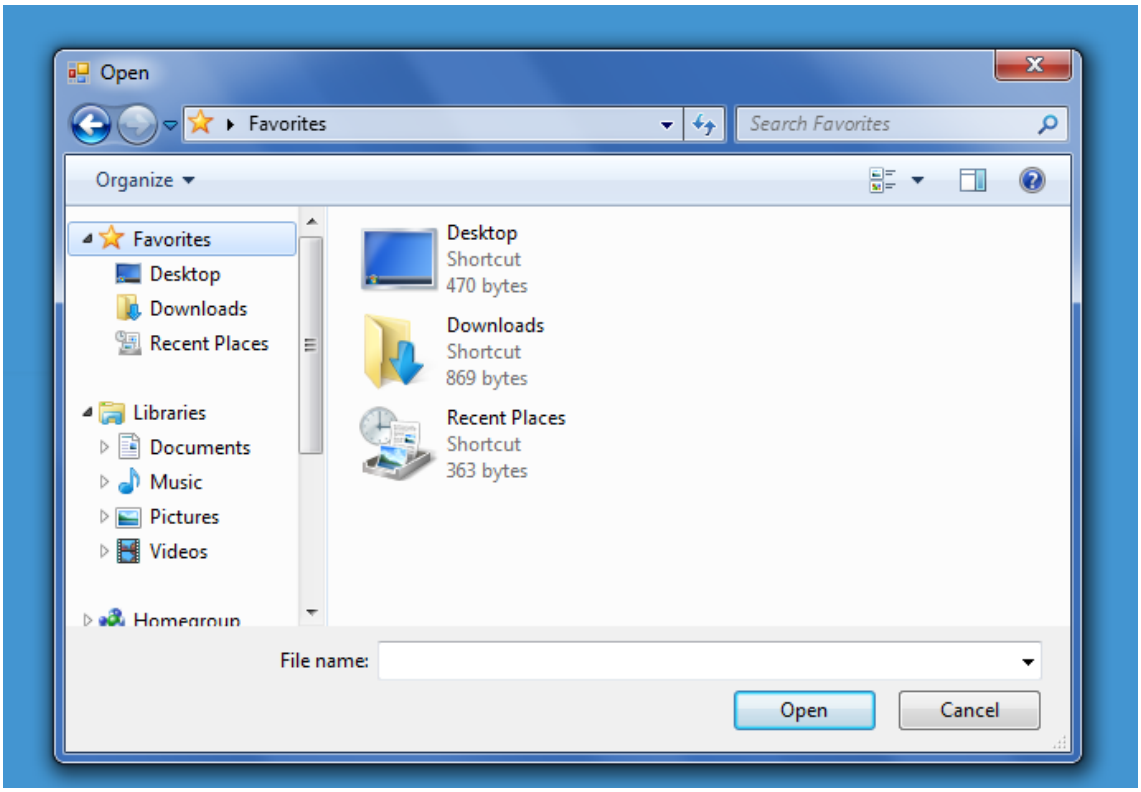

Figura I.2.F2 Ventana de exploración de directorios y selección de archivo.

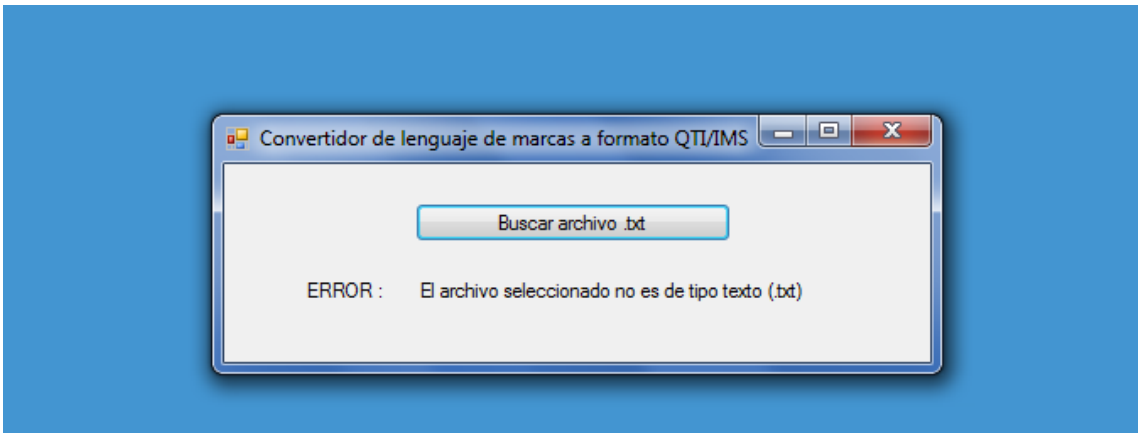

Figura I.2.F3 Mensaje de error mostrado en el menú principal.

El programa también mostrará la ubicación del archivo seleccionado, lo que es útil a la hora de recordarla para acceder posteriormente al fichero. Un mensaje expecifica que el archivo XML en estándar QTI se creará en el mismo directorio, lo que facilita su localización.

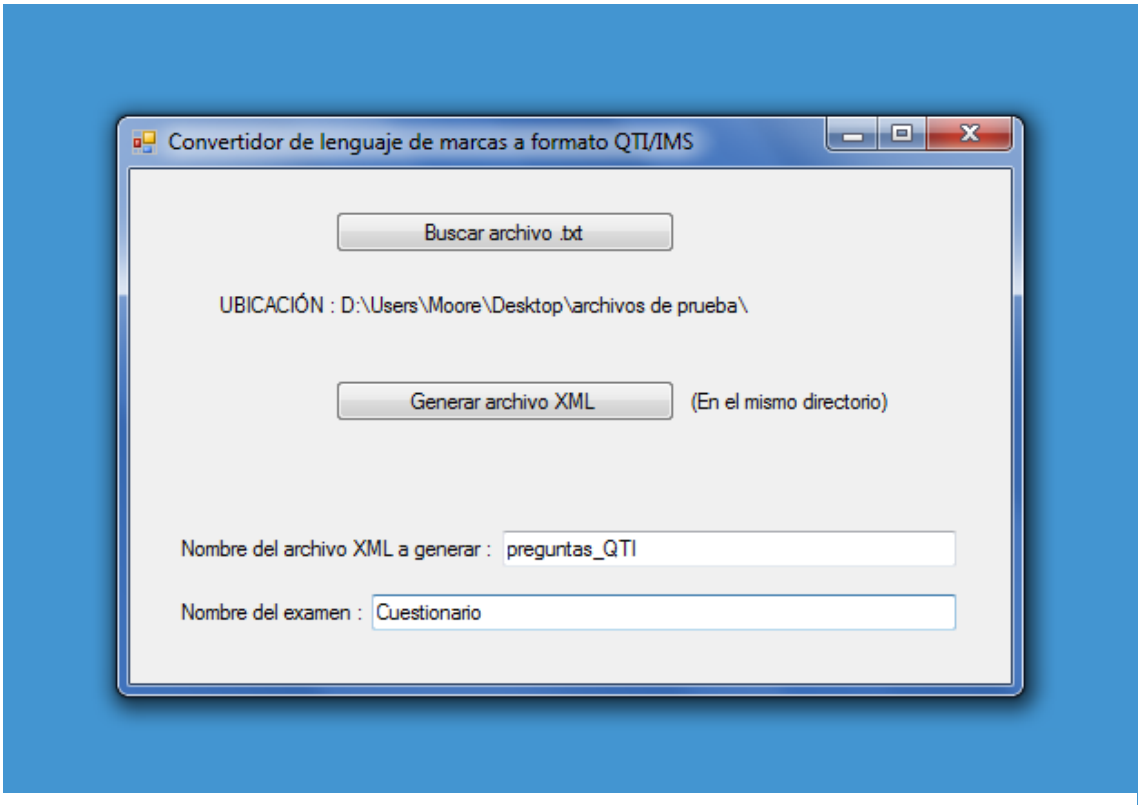

Figura I.2.F4 Menú principal con campos de introducción de datos y ubicación.

El siguiente paso es hacer clic en el botón de “Generar archivo XML”. Con esto el programa leerá línea a línea el archivo de texto proporcionado, y procederá a convertir las preguntas y respuestas, tras lo que se creará el archivo XML y se mostrará un mensaje en el menú indicando el número de preguntas convertidas.

El proceso puede repetirse las veces deseadas, eligiendo un fichero de texto diferente mediante el botón de búsqueda. Se puede editar dicho fichero con cualquier programa que guarde en texto sin formato, como por ejemplo el Notepad (bloc de notas) y volver a clicar el botón de generar archivo, con lo que el programa volverá a leerlo línea a línea generando un resultando distinto. Esto facilita y optimiza la corrección de errores en el fichero fuente.

En el proceso de desarrollo y prueba del programa se han tomado como modelo archivos QTI versión 1.2 (IMS QTI-compliant XML) generados por la plataforma Sakai en su versión 2.9.3 (Kernel 1.3.3). Sakai es una extendida plataforma

educativa *Open Source* utilizada en universidades como la Universitat Politècnica de València.

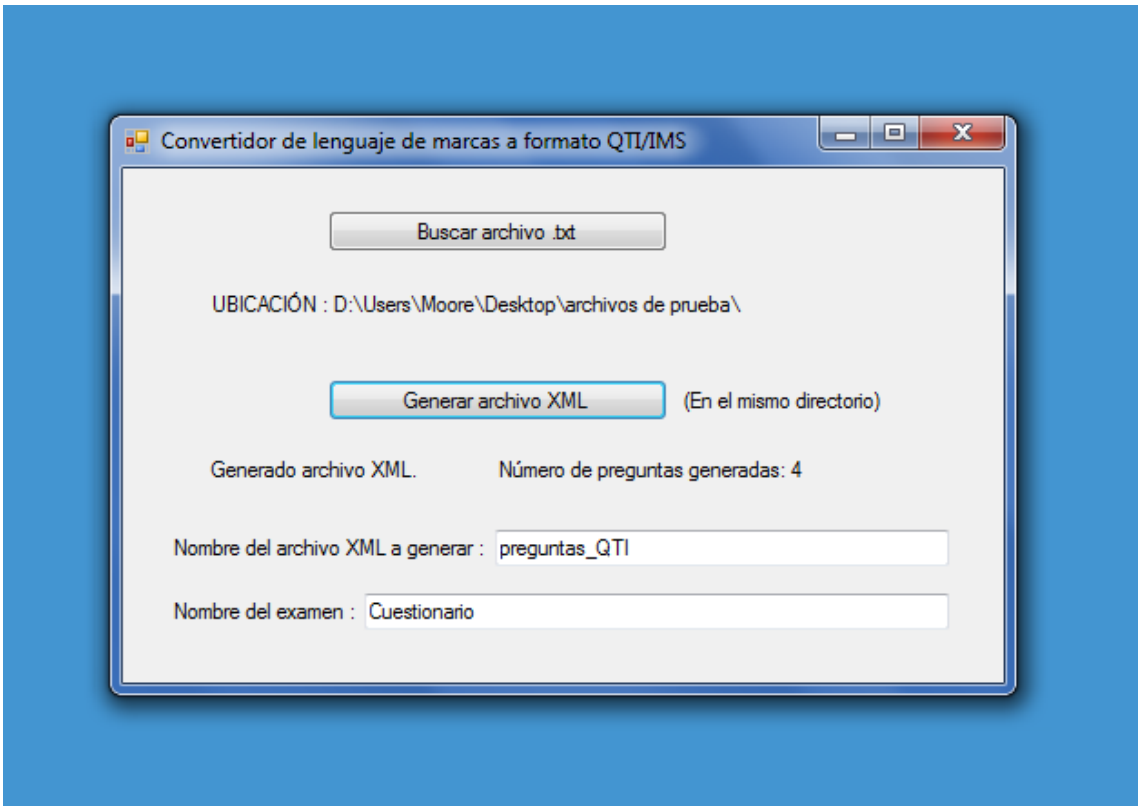

Figura I.2.F5 Menú principal una vez generado el resultado del programa.

I.3 Lenguaje de marcas utilizado.

El lenguaje de marcas utilizado por el programa intenta ser lo más sencillo posible, minimizando el número de marcas necesarias para crear preguntas y respuestas identificables por la aplicación. Las marcas de inicio y fin de pregunta se caracterizan por contener la letra Q, de *question*, pregunta en inglés.

Se utilizan obligatoriamente las marcas <QN>, <QF> y <QM> al comienzo de cada pregunta, según sea respectivamente de tipo numérico, rellenar espacios en blanco (*fill the blanks*) o de selección múltiple.

Se utiliza obligatoriamente la marca </Q> al final de cada pregunta, independientemente del tipo de ésta.

En la elección del tipo de preguntas que el programa acepta, se ha elegido las de respuesta numérica, rellenar espacios en blanco y selección múltiple (tipo test) porque son los que se prestan a crear preguntas de forma automática, utilizando para ello pequeños programas o bien *scripts*, si bien el estándar QTI contempla además otros tipos, los cuales no maneja la versión actual del programa convertidor.

El programa permite añadir opcionalmente la puntuación de la pregunta, utilizando para ello la marca <M*x*>, donde *x* es la puntuación, inmediatamente después de la marca de tipo de pregunta que se haya utilizado (<QN>, <QF> o <QM> según el caso). Si no se incluye esta marca, la puntuación por defecto es de 1 punto. La letra M de esta marca de viene dada por el término *mark*, puntuación en inglés.

Del mismo modo, y solo si se ha incluido la marca de puntuación de pregunta, se puede añadir, si se desea, la puntuación que se restará al fallar la pregunta (coloquialmente conocida como puntuación negativa) incluyendo la marca <MN*y*>, siendo *y* la puntuación que se restará. Esta última funcionalidad dependerá del funcionamiento interno de la plataforma virtual utilizada, habiendo detectado casos en los que no se accedía a dicha información a la hora de mostrar la pregunta.

I.3.1 Respuesta numérica

La sintaxis de las preguntas de este tipo utiliza, además de las marcas de inicio y fin de pregunta, las llaves "{}" para indicar la respuesta que el usuario deberá introducir. Un ejemplo sería "1+1={2}". Al mostrarse la pregunta, en vez de mostrarse las llaves y su contenido se mostraría un cuadro de texto para escribir el resultado. Ejemplos igualmente válidos serían "1+{2}=3" y "2+{3}=4". Tanto en el archivo de texto entrante como en la plataforma virtual, el contenido que se escriba entre llaves deberá ser un número. Para otros tipos de respuestas, como palabras, se usan las preguntas de tipo rellenar espacios en blanco.

Una pregunta de tipo numérico puede estar formada por varias de estas operaciones, incluyéndolas todas entre las marcas de inicio y fin de una misma pregunta.

Se ha optado por el uso de las llaves para identificar las respuestas al ser este el método de introducción utilizado por la plataforma Sakai, por lo que resultará familiar a usuarios que hayan trabajado con ese sistema.

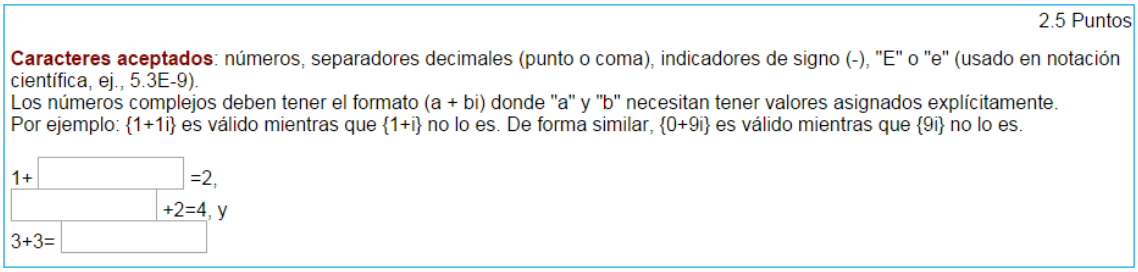

Figura I.3.1.F1 Apariencia de una pregunta de tipo numérico en Sakai y caracteres aceptados.

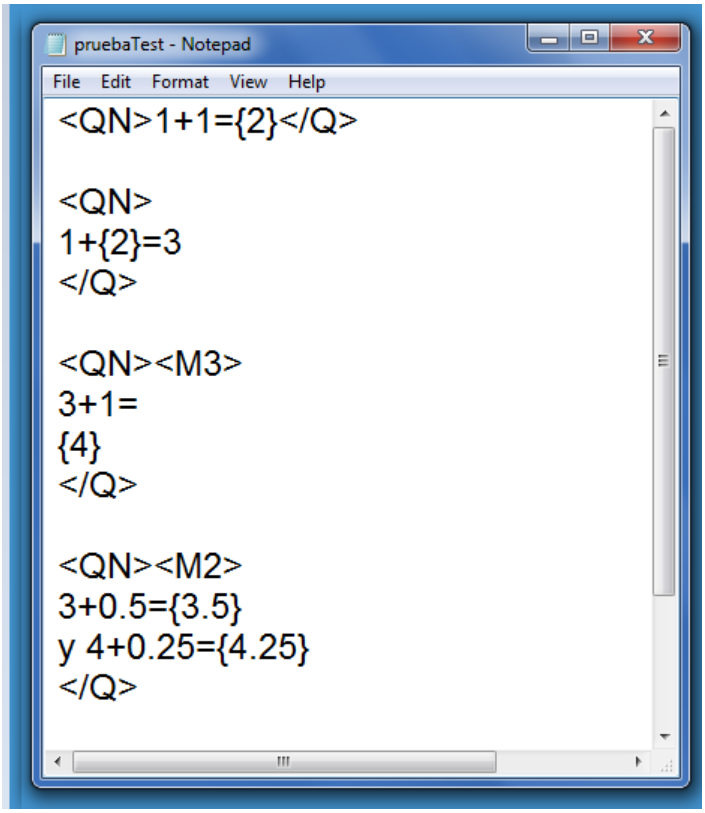

Figura I.3.1.F2 Ejemplo de archivo de texto conteniendo preguntas de tipo numérico válidas.

I.3.2 Rellenar espacios en blanco

Este tipo se utiliza para aquellas preguntas cuya respuesta la forman palabras en vez de números. Se utilizan las llaves del mismo modo que se hace con las preguntas de respuesta numérica. La única diferencia es la marca de inicio de pregunta, que es en este caso <QF>.

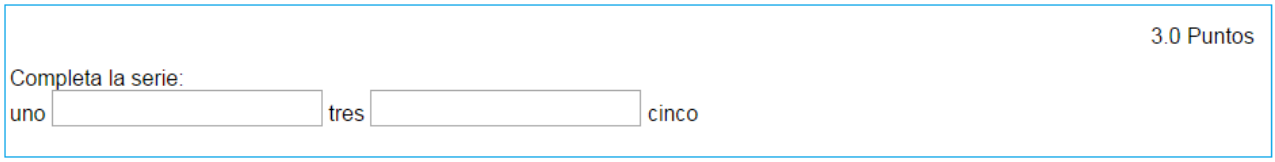

Figura I.3.2.F1 Apariencia de una pregunta de tipo rellenar espacios en blanco en Sakai.

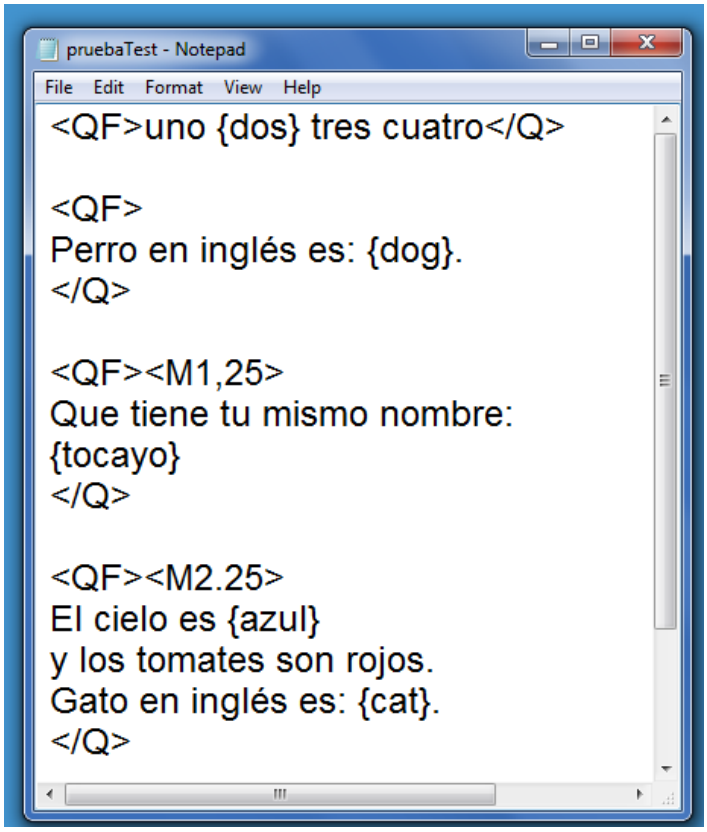

Figura I.3.2.F2 Ejemplo de archivo de texto conteniendo preguntas de rellenar blancos válidas.

I.3.3 Selección múltiple (tipo test)

El tipo de pregunta selección múltiple utiliza una marca distinta de las llaves para indicar las respuestas. Es la marca <op>, de opción, que se añadirá inmediatamente antes de cada respuesta. La respuesta o respuestas correctas deberán añadir la marca <rc> (respuesta correcta) inmediatamente después de la marca <op> para señalarlo, pudiendo ser utilizada en varias respuestas según el

número de respuestas correctas que tenga la pregunta. El final de una opción se delimita por la aparición de otra marca <op> (en el caso de que queden más opciones) o por la aparición de la marca de final de pregunta.

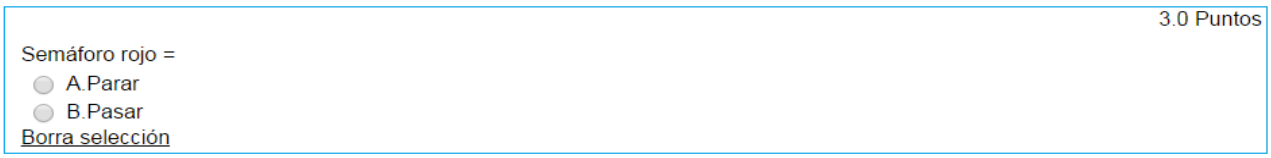

Figura I.3.3.F1 Apariencia de una pregunta de tipo test en Sakai (una única respuesta correcta).

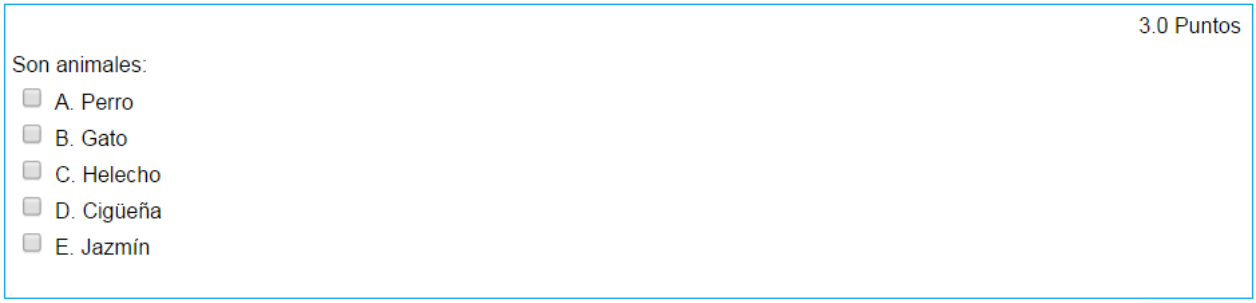

Figura I.3.3.F2 Apariencia de una pregunta de tipo test en Sakai (varias respuestas correctas).

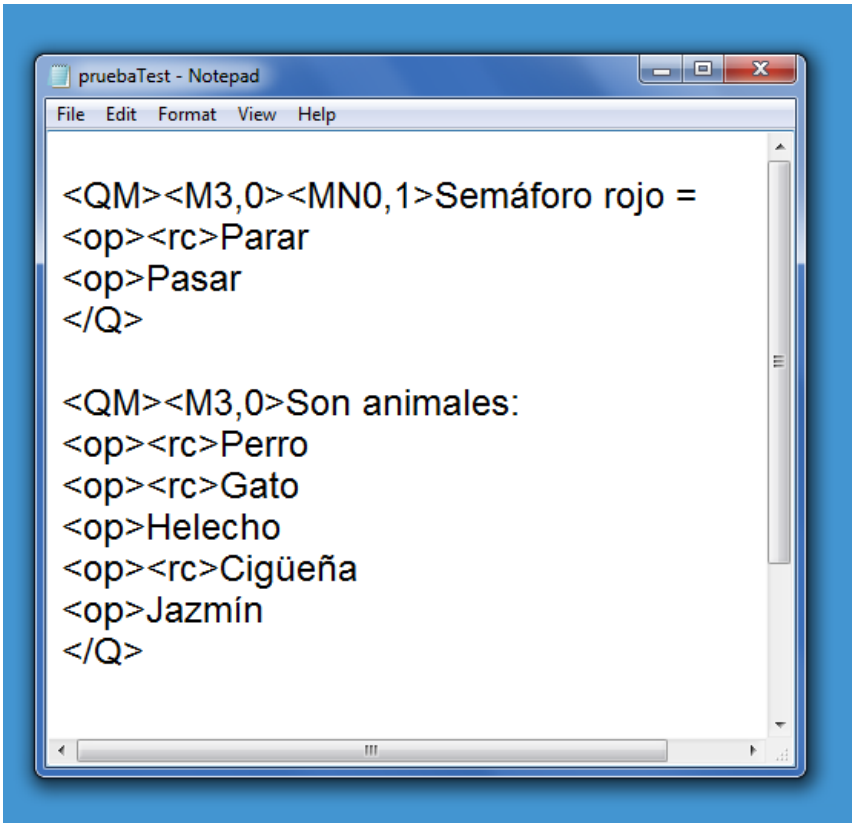

Figura I.3.2.F3 Ejemplo de archivo de texto conteniendo preguntas de tipo test válidas.

I.4 Formatos y símbolos

Además de las marcas correspondientes a los tres tipos de preguntas aceptados, el lenguaje de marcas acepta marcas del lenguaje HTML, que en muchos casos las plataformas web docentes interpretarán correctamente. He aquí las de uso más común:

-Insertar un salto de línea:

-Texto en negrita: <B>*texto*</B>

-Texto en cursiva: <I>*texto*</I>

-Texto subrayado: <U>*texto*</U>

-Subíndice: _{*texto*}

-Superíndice: ^{*texto*}

-Enlaces web: *nombre*

En lo referente al tipo de caracteres aceptados por el programa, éste lee el contenido del archivo de texto proporcionado haciendo uso del *encoding* por defecto usado por nuestro sistema, con lo que leerá todo carácter que cualquier programa de texto sin formato, como el Bloc de notas/Notepad, nos permita incluir en un archivo con la extensión txt. Esto nos permite utilizar acentos, diéresis, la letra ñ, etcétera, pero no otros como las letras griegas. Para estos caracteres especiales, se recomienda usar igualmente las marcas HTML, ya sea en forma de código numérico de carácter (Ω) o en su forma textual (Ω).

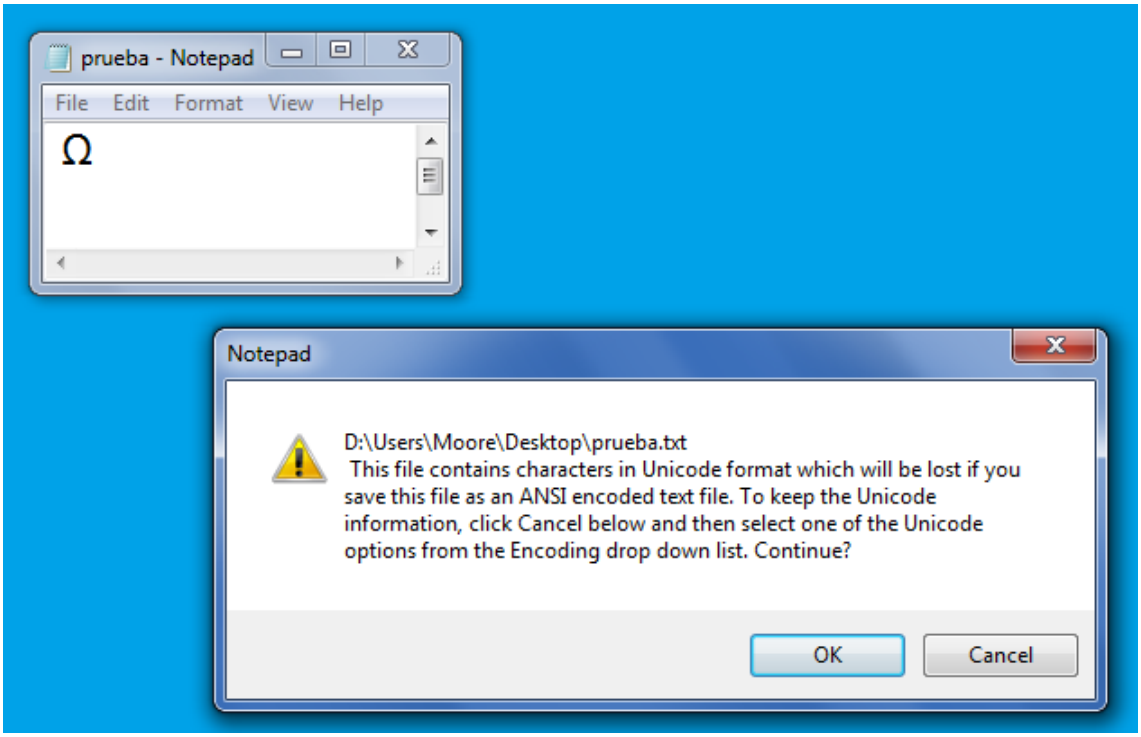

Figura I.4.F1 Mensaje de error mostrado al intentar guardar caracteres especiales en formato txt.

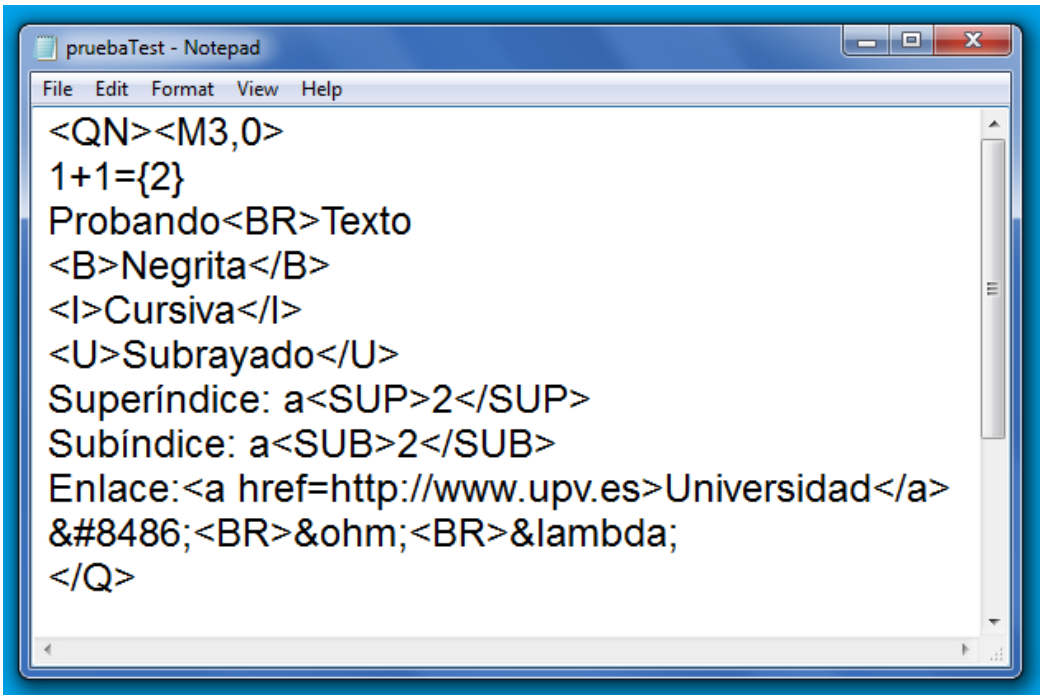

Figura I.4.F2 Archivo txt conteniendo marcas HTML interpretables por Sakai.

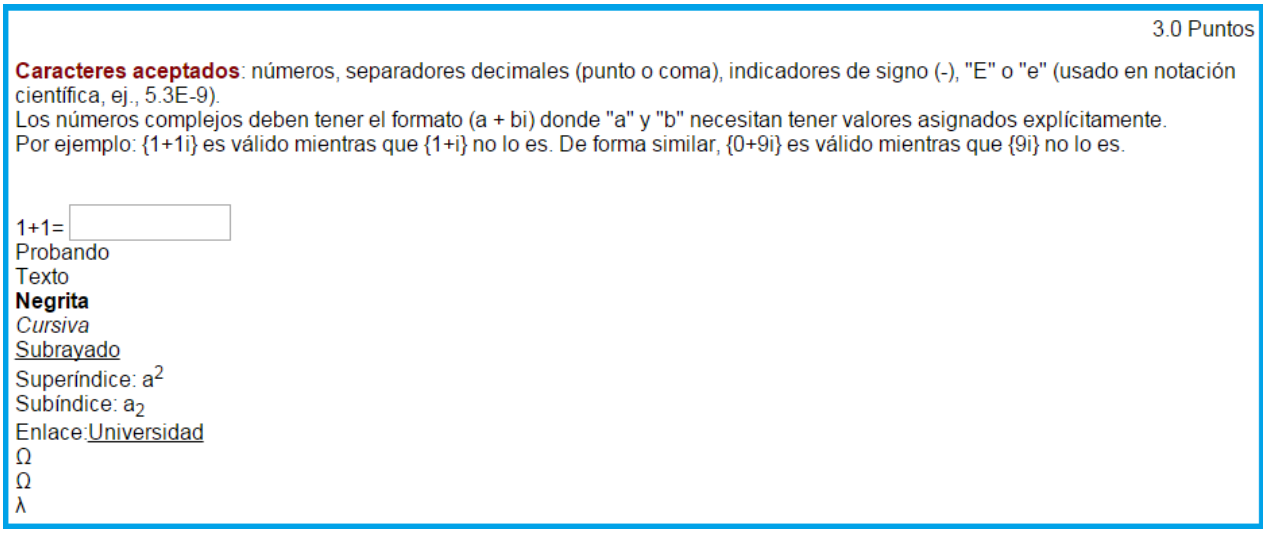

Figura I.4.F3 Resultado de convertir el archivo txt de la figura anterior.

Anexo II. Bibliografía

Gregory M. Horine. *Manual imprescindible de gestión de proyectos*.
Anaya Multimedia, 2015.

Guillem Bou Bauzá, Carme Trinidad Cascudo, Llorenç Huguet Borén. *E-learning*.
Anaya Multimedia, 2003.

David Roldán Martínez, Raúl E. Mengod López, Daniel Merino Echeverría. *Sakai. Administración, configuración y desarrollo de aplicaciones*.
Ra-Ma, 2011.

Jordi Conesa Caralt, Àngels Rius Gavidia, Jordi Ceballos Villach, David Gañán Jiménez. *Introducción a .NET*.
UOC, 2010.

IMS Global Learning Consortium, Inc. *IMS Question & Test Interoperability: ASI Information Model Specification*.
2002.

IMS Global Learning Consortium, Inc. *IMS Question & Test Interoperability: ASI XML Binding Specification*.
2002.