DESARROLLO DE RIAS: CASO DE ESTUDIO EN EL DOMINIO DE LA ANOTACIÓN DE TEXTOS LITERARIOS DIGITALIZADOS

CÉSAR OCTAVIO RUIZ CARICOTE

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA, FACULTAD DE INFORMÁTICA, UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Sistemas Inteligentes

10 de Septiembre de 2012

Director:

José Luis Sierra Rodríguez

Autorización de Difusión

CÉSAR RUIZ CARICOTE

10 de Septiembre de 2012

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: “DESARROLLO DE RIAS: CASO DE ESTUDIO EN EL DOMINIO DE LA ANOTACIÓN DE TEXTOS LITERARIOS DIGITALIZADOS”, realizado durante el curso académico 2011-2012 bajo la dirección de José Luis Sierra Rodríguez en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen

La digitalización masiva de textos literarios abre amplias oportunidades de investigación, antes permitidas a grupos reducidos de investigadores, y limitadas muchas veces por la disponibilidad de los textos, debido a la ubicación geográfica o el deterioro de los mismos.

Esta actividad ha fomentado la aparición de un nuevo campo de investigación conocido como las “Humanidades Digitales”. Esta nueva disciplina está enfocada a la interacción de la computación y las disciplinas humanísticas. Por esta razón, han emergido nuevas herramientas digitales de análisis textual. Sin embargo tales herramientas necesariamente deben ofrecer dinamismo, rapidez y sentido intuitivo para los investigadores humanistas.

Para llevar a cabo tales exigencias, es posible utilizar un nuevo tipo de aplicación web: las Aplicaciones Ricas de Internet (RIA – *Rich Internet Applications*), las cuales utilizan datos que pueden ser procesados por el servidor o cliente, ofreciendo un *look-and-feel* similar a las aplicaciones de escritorio.

Sin embargo la ingeniería de las RIAs es un campo relativamente nuevo en el área de la Ingeniería de Software, área en la que todavía no se ha delimitado formalmente una definición precisa de RIA ni una especificación rigurosa de sus características.

Este proyecto de investigación se encamina precisamente a buscar de forma teórica y práctica el enfoque, metodología, y tecnologías necesarias para el desarrollo de una RIA que sirva de herramienta de anotación de textos literarios digitalizados con propósitos educativos, cumpliendo con las exigencias de análisis de los investigadores humanistas

Palabras clave

Humanidades Digitales, E-learning, Anotación, RIA, AJAX, GWT, GAE

Abstract

The massive digitization of texts grandly opens research opportunities, previously allowed to small groups of researchers, and limited (sometimes by geography or damage of the text) in availability.

This activity has permitted the entrance to a new research field known as "Digital Humanities", focused on the interaction of computing and the humanities. For this reason new digital tools of textual analysis have emerged. However, such tools must necessarily offer dynamism, speed and intuitive sense for scholar humanists.

To carry out these requirements a new type of web application can be used, Rich Internet Applications (RIA), which use data that can be processed by the server or client, offering a look-and-feel similar to desktop applications.

However, RIAs engineering is a relatively new field in the area of software engineering, in which a precise definition of RIA and the specification of its features have not yet been formally defined.

This research project is aimed precisely in focusing on the theoretical and practical approach, methodology, and technologies needed for the development of a RIA to serve as an educational-oriented text annotation tool that meets the demands and requirements of scholar humanists' analysis

Keywords

Digital Humanities, E-learning, Annotation, RIA, AJAX, GWT, GAE

Índice de Contenidos

Agradecimientos

A José Luis Sierra, quien es el principal creador de este proyecto, y quien mediante su guía me ayudó inagotablemente a escribir este trabajo.

A mi familia, que desde la distancia me apoyó para seguir adelante en cumplir este sueño en tierras extranjeras.

Y a mis compañeros y amigos que estuvieron pendiente durante el desarrollo de este trabajo.

A todos ustedes, gracias.

Capítulo 1 - Introducción

Con la digitalización masiva de textos de gran importancia histórica, los más importantes repositorios y bibliotecas electrónicas a gran escala se han ofrecido para alojar estos textos. Tal es el caso de la *Universal Digital Library*[1], el *Gutenberg Project*[2], *Hathi Trust*[3], o el *Google Library Project*[4]. Así por ejemplo, *Hathi Trust* ha llevado a cabo la digitalización de 8 mil toneladas de documentos para un total de más de diez millones de volúmenes. Así mismo, el *Google Library Project* mantiene un acervo de más de 15.5 millones de textos, proporcionados por distintas universidades y organizaciones añadidas al proyecto. En particular, la biblioteca de la Universidad Complutense (BUCM[5]), ha colaborado con esta iniciativa, aportando cientos de libros de su fondo antiguo para su digitalización.

Es lógico pensar que la digitalización masiva de textos abre grandes oportunidades de investigación, antes permitidas únicamente a grupos reducidos de investigadores, debido a limitaciones geográficas, deterioro de los textos, etc. Efectivamente, esta actividad ha fomentado la creación de un nuevo campo de investigación conocido como las "Humanidades Digitales", enfocado a la interacción de la computación y las disciplinas humanísticas. Por esta razón, han aparecido también nuevas herramientas digitales de análisis textual. Como un requisito básico, tales herramientas han de ofrecer a los investigadores humanistas dinamismo, rapidez y sentido intuitivo.

Para llevar a cabo tales exigencias, es posible adoptar un nuevo modelo de aplicación web, las Aplicaciones Ricas de Internet (RIA), las cuales utilizan datos que pueden ser procesados por el servidor o cliente, ofreciendo un *look-and-feel*, similar a las aplicaciones de escritorio, caracterizadas fundamentalmente por una variedad de controles de operación interactiva, la posibilidad de uso online/offline de la aplicación y el uso transparente de conexión entre cliente y servidor, con lo cual dichas aplicaciones pueden superar las limitaciones de las aplicaciones web tradicionales y ofrecer a los usuarios una herramienta rica y de respuesta inmediata a sus eventos, produciendo así, un sentido intuitivo en su uso.

[1] http://www.ulib.org/
[2] http://www.gutenberg.org/
[3] http://www.hathitrust.org/
[4] http://www.google.com/googlebooks/library.html
[5] http://www.ucm.es/BUCM/biblioteca/

Sin embargo la ingeniería de las RIAs es un campo relativamente nuevo en el área de la Ingeniería del Software, en la que aún no se ha delimitado formalmente una definición precisa de RIA, ni tampoco un catálogo preciso con sus características. De esta forma, este proyecto de investigación aborda estas cuestiones a través de un caso de estudio real, de complejidad no trivial en el dominio de las Humanidades Digitales: el desarrollo de una herramienta de anotación de textos digitalizados con fines educativos que cumplan con las exigencias de análisis de los investigadores humanistas.

1.1 Objetivos del Proyecto

De esta forma, el objetivo primario de este proyecto es llevar a cabo una investigación sobre el uso de RIAs en el desarrollo práctico de aplicaciones web para las Humanidades Digitales. En particular, la investigación se focalizará en un caso de estudio desarrollado en el proyecto "Collaborative Annotation of Digitalized Literary Texts", financiado por Google a través de su *Digital Humanities Award Program*. Este caso de estudio se trata de una aplicación para la anotación colaborativa de textos literarios digitalizados desarrollada en el marco de este proyecto, y denominada @note. Este objetivo primario se desgrana, a su vez, en los siguientes objetivos específicos:

- Realizar un breve análisis de los principales requisitos de las aplicaciones de anotación de textos literarios digitalizados.
- Realizar un estudio del estado del arte y estudio comparativo relativo a los métodos y tecnologías utilizadas en el desarrollo de RIAs.
- Describir el diseño y la implementación de @note, el caso práctico de desarrollo de RIAs en el campo de las Humanidades Digitales llevado a cabo en este proyecto.
- Evaluar la utilización de @note desde el punto de vista tecnológico, obteniendo diferentes estadísticas de uso del sistema que permitan corroborar su capacidad y robustez como soporte a diferentes actividades de anotación de textos.

Como resultado práctico de estos objetivos, se recabará experiencia que permita futuras mejoras del diseño e implementación de la aplicación en el seno del proyecto financiado por Google.

1.2 Estructura del Documento

El resto del documento se divide en cuatro capítulos:

- En el capítulo 2 se realiza una revisión del estado de la cuestión, la cual fundamenta esta propuesta, donde se revisan brevemente las principales características de las herramientas de anotación de textos digitalizados, así como los conceptos fundamentales de las RIAs. Se lleva a cabo, así mismo, un estudio de los principales enfoques orientados a modelos para el desarrollo de las RIAs, tales como OOH4RIA, las extensiones de WebML, RUX, o ADRIA. Se estudian, por último, las tecnologías de implementación de RIAs, incidiendo en las tecnologías Google: Google Web Toolkit (GWT), un *framework* AJAX orientado a objetos, el cual posee un compilador especial que transforma el código Java en Javascript, y Google Application Engine (GAE), una infraestructura de servidor basada en el sistema BigTable, un modelo de datos diferente al relacional, con características que lo hacen más escalable, pudiendo crecer las correspondientes bases de datos hasta los petabytes de almacenamiento.
- El capítulo 3 presenta un caso de estudio en el dominio del campo de investigación de las humanidades digitales, explicando la arquitectura, diseño y modelos de datos de la RIA @note desarrollada por el grupo de investigación ILSA (Ingeniería de Lenguajes Software y Aplicaciones) de la Facultad de Informática de la UCM. La implementación de @note se basa en las tecnologías Google, aunque su arquitectura es suficiente versátil como para migrar de manera sencilla a otras tecnologías. El capítulo revisa las principales funcionalidades de la herramienta, su modelo de datos, la organización de la herramienta en el lado del cliente y en el lado del servidor, y los principales aspectos del proceso seguido en su desarrollo.
- En capítulo 4 muestra unas estadísticas de uso del servidor de aplicaciones para la RIA @note, en términos del modelo de datos que ésta utiliza.
- El último capítulo presenta las conclusiones obtenidas con el desarrollo de este proyecto, y esboza algunas posibles líneas de trabajo futuro.

Capítulo 2 - Estado de la Cuestión

2.1 Introducción

En este capítulo se presenta una introducción a los aspectos teóricos y tecnológicos más relevantes que guardan relación directa con este proyecto de máster. Primero se presenta una breve introducción a la anotación de textos literarios digitalizados, tema fundamental para el desarrollo y esbozo de la aplicación tratada en este proyecto, así como la introducción de herramientas de anotación existentes, mostrando en cada caso sus avances y limitaciones. Seguidamente se introduce la base teórica de RIAs, marcando las diferencias y beneficios al desarrollo de las aplicaciones web tradicionales, y remarcando su importancia en el campo del desarrollo de aplicaciones educativas (ya que, en última instancia, el propósito de @note será fundamentalmente educativo). Luego se explica los diferentes métodos utilizados para el desarrollo de las RIAs, así como las extensiones hechas a métodos existentes, necesarios para cubrir los conceptos añadidos que éstas generan respecto a las aplicaciones web tradicionales. Finalmente se presentan las tecnologías utilizadas para su implementación, estableciendo comparaciones entre ellas, y dejando apartada, para una explicación más detallada, el caso de GWT + GAE, por ser ambas las tecnologías utilizadas para la implementación de la aplicación @note en la cual se basa este proyecto.

2.2 Anotaciones de Textos.

2.2.1 Anotación Digital de Textos

La utilización de ordenadores en el campo investigativo de las ciencias humanísticas no es algo nuevo. Uno de los trabajos pioneros más conocido en esta área, es el llevado a cabo por el padre Robert Busa, quién empezó su trabajo con el índice Tomístico (un índice con los trabajos del teólogo medieval Tomás de Aquino) a finales de 1946. Algunos siguieron esta dirección, incluyendo a Antonio Zampolli, quién puede ser considerado un precursor en la aplicación de técnicas computacionales en la investigación literaria y lingüística desde 1960 (Frommholz & Fuhr, 2006). A pesar de esta gran asociación de aplicaciones computacionales con investigación humanística, es sólo en años recientes cuando la utilización de técnicas computacionales se ha hecho extensiva entre estudiantes de humanidades. A este respecto, el uso de anotaciones electrónicas es particularmente relevante. Efectivamente, a lo largo de la historia,

la anotación de textos ha variado su forma según los tiempos, sufriendo adaptaciones pertinentes. De todas estas adaptaciones, las anotaciones electrónicas han supuesto el cambio más significativo, y los humanistas ya son conscientes de que existe una era digital, y no escatiman en ajustar sus métodos científicos de análisis de los textos literarios a las nuevas tecnologías (Sanz & Goicochea de Jorge, 2009).

Una anotación, en un sentido amplio, es una nota externa realizada durante la investigación de cualquier tipo de recurso (documentos, imágenes, audio), sin afectar en ninguna forma al objeto que se anota, pudiendo ser tan simple como subrayar o resaltar pasajes. Su propósito es el de proporcionar al investigador una explicación, además de ofrecer una evaluación de la fuente del recurso.

Para la investigadora de Microsoft, Catherine C. Marshall, las anotaciones sobre textos digitalizados cumplen diversas funciones para el lector: como marca de posiciones (ayuda a memorizar), como actividad interpretativa (ya sea por el profesor o por el alumno) y como un compromiso con el aprendizaje cuando el alumno trabaja con una lectura compleja (Han-Zhen et al., 2008). Distintos grupos de investigación literaria han observado el interés y particular importancia de la anotación de textos digitalizados. Por ejemplo el grupo LEETHI (Literaturas Españolas y Europeas: del Texto al Hipertexto) de la Universidad Complutense de Madrid pone en práctica la utilización de herramientas tecnológicas que permitan realizar anotaciones sobre textos literarios digitalizados. Uno de sus propósitos es el de analizar diversos tipos de textos y transformar sus anotaciones en objetos de aprendizaje; de esta manera, podrán ser utilizadas para compartir conocimientos sobre temas específicos y el profesor podría utilizar las anotaciones realizadas por los alumnos como métodos de evaluación de su asignatura.

2.2.2 *Aplicaciones de anotación de recursos digitalizados*

De esta forma, aparecen distintos sistemas y aplicaciones que facilitan la anotación de textos digitalizados en particular, y de recursos digitales en general. De las propuestas existentes (véase (Tazi S, 2003), (Koivunen, 2005)), los sistemas Collex, Alfalab, Vannotea y MemoNote se consideran especialmente relevantes, y se analizan con más detalle a continuación.

2.2.2.1 Collex

Collex[6] es un sistema de código abierto, desarrollado por la organización académica NINES (Networked Infrastructure for Nineteenth – Century Electronic Scholarship), que reúne colecciones de textos digitalizados e imágenes, y permite añadir etiquetas sobre las cuales otros usuarios pueden buscar y crear discusiones como si se tratara de un aula virtual e-leaning, con el fin de contribuir en el intercambio de textos del siglo XIX. Collex influencia actuales desarrollos de tecnología de la Web semántica para realizar operaciones de minería de datos con el fin de ampliar el descubrimiento de conocimiento (véase (McGann, 2008)).

En Collex, cualquiera puede unirse a la organización y empezar a colaborar. Es sin duda, un excelente aporte para el campo de las humanidades, pero lamentablemente sólo se limita al acervo de libros digitalizados que ellos poseen.

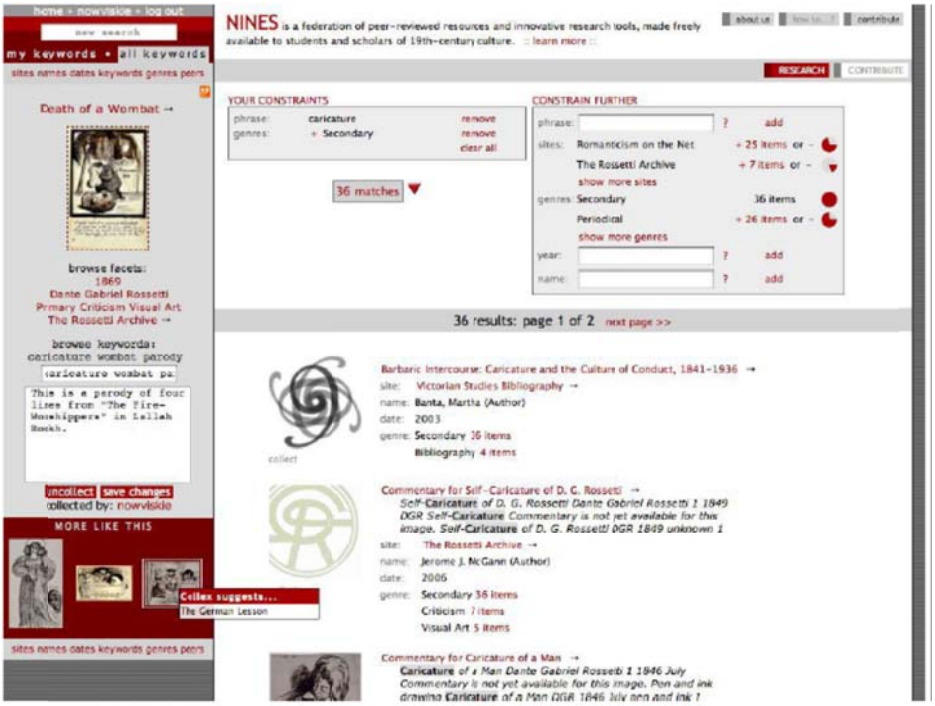

Figura 1 Un conjunto de restricciones y una vista detallada en Collex de un objeto (para etiquetarlo, anotarlo, y descubrir conocimiento) (Imagen tomada de su sitio web *www.collex.og*)

Collex utiliza los principios y tecnologías de la web semántica para explorar el potencial investigativo de la producción académica digital agregada en NINES. Dos conceptos

[6] http://www.collex.org/

fundamentales entran en Collex: "clasificación por facetas" y "folksonomías", las cuales generan una interfaz evolutiva entre los recursos electrónicos revisados e integrados en NINES y las comunidades de usuarios que anotan este contenido a través de interpretación y contextualización crítica. Utiliza RDF (*Resource Description Framework*, infraestructura que permite la codificación, intercambio y reúso de metadatos estructurados (Heath & Bizer, 2011)) para definir "objetos" coleccionables sin limitarlos a su expresión como páginas web. Donde otras herramientas de marcado social se diseñan para permitir colecciones y anotaciones de todas las páginas web, Collex permite a contribuyentes de recursos realizar distinciones más finas, y a usuarios del sistema crear colecciones más acordes a los patrones de los investigadores humanistas. La Figura 1 muestra una búsqueda con restricciones y el despliegue detallado de un objeto en Collex.

2.2.2.2 Alfalab

Alfalab (Zundert, 2009) es un framework colaborativo del KNAW (*Royal Netherlands Academy of Arts and Sciences*), el cual aplica y promueve el uso de herramientas y métodos digitales en la investigación de las humanidades y fomenta la cooperación entre investigadores humanistas tanto a escala local como internacional.

Los grupos de desarrollo e investigación de Alfalab, compuestos por los investigadores humanistas e ingenieros informáticos, trabajan actualmente en el desarrollo de dos importantes aplicaciones, TexLab y SpaceLab:

- TextLab incluye herramientas digitales para la exploración textual, como por ejemplo, un servicio de reconocimiento de entidad para estudios onomásticos, un servicio para transcripción y anotación de recursos textuales y un servicio de apoyo inductivo para la auto-sugerencia de anotaciones de textos.
- Por su parte, SpaceLab ofrecerá herramientas para aplicar información geográfica (GIS) a materiales históricos (todavía en desarrollo, el equipo ha reservado detalles).

De igual forma ayuda a investigadores a transcribir y/o anotar textos al proporcionarles sugerencias, y han incorporado prácticas exitosas llevadas a cabo en redes sociales, como Facebook.

2.2.2.3 Vannotea

Vannotea es un sistema prototipo desarrollado en el centro tecnológico de sistemas distribuidos de la Universidad de Queensland, que permite la indexación, navegación, descripción, anotación, y discusión colaborativa de películas digitales de alta calidad. Utilizando la red de investigación GrangeNet[7], que soporta colaboración de grupo-a-grupo a gran escala y vídeo/audio de alta calidad, los usuarios pueden abrir archivos MPEG-2 y de esta forma segmentar, navegar, describir y anotar de manera colaborativa el vídeo de interés. Aunque existen herramientas de anotación para recursos de audio y vídeo, estos han sido diseñado para uso de entornos *stand-alone*, y aunque las descripciones y anotaciones se pueden compartir al guardarlas en servidores, no están diseñadas para ser compartidas colaborativamente en tiempo real. No obstante, la posibilidad de compartir anotaciones hace que el sistema Vannotea sea de especial interés para las comunidades médicas, científicas, educacionales y de defensa, al permitir discusiones en línea sobre un vídeo particular y la anotación en tiempo real de segmentos, marcos o regiones dentro de los marcos entre grupos distribuidos.

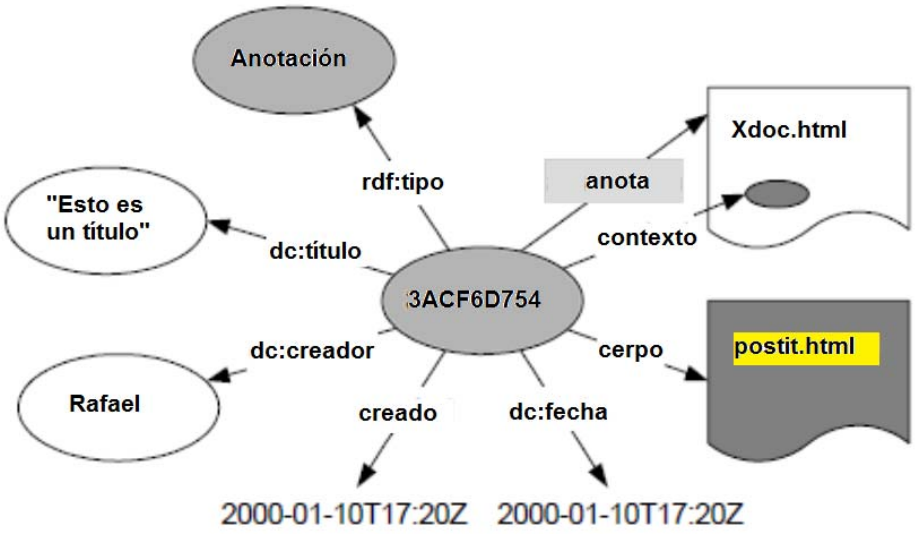

Figura 2 Esquema de anotación básico propuesto por Annotea (Koivunen, 2005).

La base de datos de anotación almacena las anotaciones (las cuales pueden estar asociadas con segmentos, fotogramas, o regiones dentro de algún fotograma), así como la fuente de la anotación (autor, fecha, ubicación). Para este sistema se decidió colocar las anotaciones

[7] http://www.grangenet.net/

sobre Annotea[8], un sistema de código abierto desarrollado por W3C, el cual permite adjuntar anotaciones compartidas a cualquier documento o parte del documento Web. Annotea utiliza un esquema de anotación basado en RDF y XPointer (especificación utilizada para definir anclas en documentos (W3C, 2001)) para enlazar la anotación con el documento. La Figura 2 ilustra el esquema básico de anotación empleado por Annotea.

Vannotea extiende Annotea para soportar la anotación de documentos audiovisuales, utilizando un modelo que se especifica a través de extensiones a XPointer, permitiendo la ubicación de segmentos específicos, fotogramas o regiones. Este enfoque también permite utilizar servidores prototipos de anotación como Zope[9] o W3C perlib (W3C, 1999), que son bases de datos RDF que están por encima de MySQL y proveen su propio lenguaje de consulta: Algae[10].

El cuerpo de la anotación es por lo general texto o HTML, pero esta arquitectura permite generar, adjuntar y almacenar anotaciones audiovisuales, pequeños clip de vídeos o audio capturados durante una discusión en conferencia por ejemplo.

2.2.2.4 MemoNote

MemoNote (Demoulins et al., 2006) es una herramienta de anotación desarrollada en el laboratorio CLIPS (Grenoble). Es una herramienta de anotación semántica dedicada a profesores, que les permite anotar documentos digitalizados con sus propios comentarios, en donde cada anotación tiene una semántica explícita tanto para el profesor como para la máquina. Esto es posible gracias al uso de ontologías, que son particulares para la actividad del profesor. Además de ser semántica, cada anotación es subjetiva, reflejando el punto de vista personal del profesor sobre una determinada circunstancia. Por lo tanto, para cada nueva anotación, MemoNote almacena tres facetas: una cognitiva, una semántica y una personal, tal como se muestra en la Figura 3.

Con el fin de asistir al profesor en la creación de estas anotaciones semánticas, la herramienta les provee de *patrones de anotación*. Estos patrones representan una forma de ancla de la anotación (resaltado, circulo, subrayado, etc) la cual está enlazada a su semántica en un contexto dado. La herramienta utiliza estos patrones para deducir automáticamente la semántica.

[8] http://www.w3.org/2001/Annotea/
[9] http://www.zope.org/Members/Crouton/ZAnnot/
[10] http://www.w3.org/1999/02/26-modules/User/Algae-HOWTO.html

El profesor, durante su actividad, puede cambiar entre diferentes contextos, en los que anota utilizando patrones diferentes y, por consiguiente utiliza un grupo específico de patrones para cada contexto específico. Por ejemplo, mientras el resaltado rojo significa un indicador de error en laboratorios de química, puede significar un importante fragmento a discutir con otros profesores en la preparación de un químico del mismo laboratorio. De esta forma, la aplicación de un patrón específico es determinista únicamente si el contexto ha sido previamente establecido. En consecuencia, MemoNote es sensible al contexto para poder adaptar su comportamiento a cada estado de contexto de cada profesor.

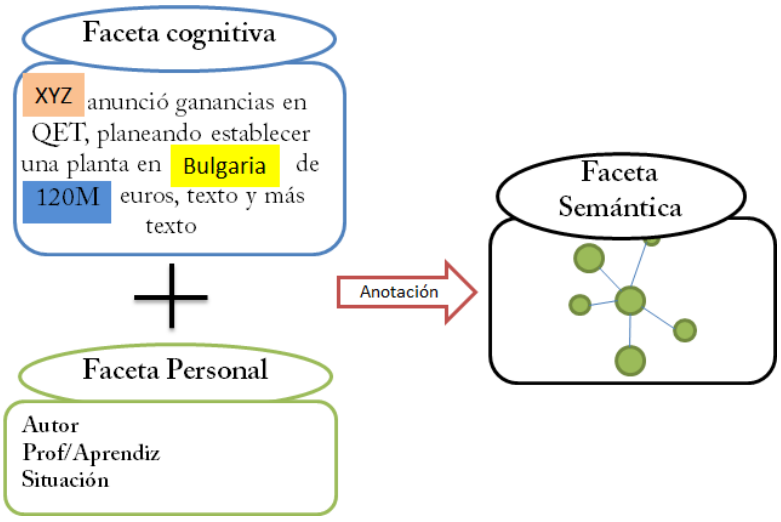

Figura 3 Modelo de tres facetas (Demoulins et al., 2006)

Para representar y almacenar estas anotaciones se utiliza también RDF, y para representar las ontologías MemoNote utiliza OWL (Web Ontology Language).

2.2.3 Esfuerzos de estandarización: OAC

Las herramientas presentadas anteriormente hacen patente la existencia de múltiples formatos y convenios utilizados en la representación de anotaciones. Por tanto, a fin de facilitar la interoperabilidad entre estas herramientas, es necesario introducir algún tipo de estándar.

A este respecto, el grupo internacional OAC[11] (Open Annotation Collaboration) tiene como objetivo fundamental, proporcionar un entorno de anotación interoperable basado en web, que permita a múltiples servidores y clientes, mediante servicios, descubrir y hacer uso de la

[11] http://www.openannotation.org

valiosa información contenida en la anotación. Para este fin adoptaron un modelo de datos basado en datos vinculados RDF. Las directrices de la arquitectura Web y el uso de RDF son los principios fundamentales para el modelo OACM que propone OAC, lo cual resulta en una especificación que puede ser aplicada a cualquier conjunto de recursos Web.

El Modelo OACM proporciona un método interoperable que permite expresar anotaciones de manera que puedan ser compartidas entre plataformas, y es suficientemente expresivo como para satisfacer las necesidades de los investigadores sin dejar de ser lo suficientemente simple para también permitir casos comunes, como la anexión de un fragmento de texto a un recurso web. Las anotaciones se modelan como un conjunto de recursos conectados, incluyendo un *cuerpo* y uno o más objetivos o *targets*, en donde el *cuerpo* hace referencia a esos *targets.*

El modelo OAC, como se muestra en la Figura 4, tiene tres clases principales de recursos:

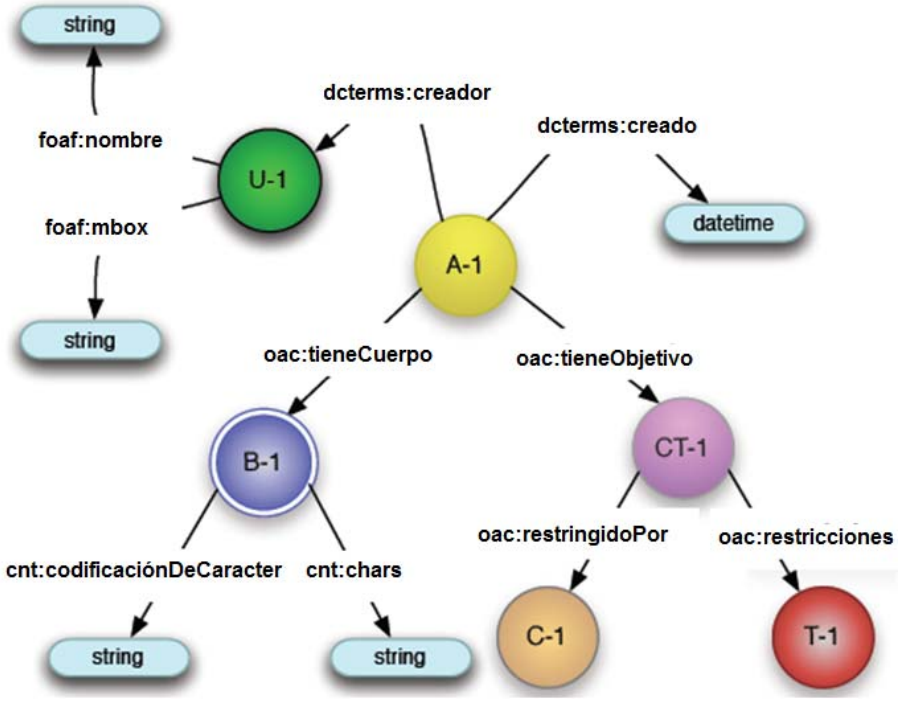

Figura 4 Modelo Básico de OAC (Haslhofer et al., 2011)

- El *oac:Cuerpo:* de la anotación (nodo B-1). Este recurso representa el comentario, los metadatos, u otra información que se cree acerca de otro recurso. Puede ser cualquier recurso Web, de cualquier formato, disponible en cualquier URI. El modelo permite únicamente un cuerpo por anotación.

- El *oac:Target* de la anotación (nodo T-1). Este es el recurso a cual el *Cuerpo* se refiere. Al igual que el *Cuerpo*, puede ser cualquier URI que identifique al recurso. El modelo permite uno o más *Targets* por anotación.
- El *oac:Anotación* (nodo A-1). Este elemento es un documento identificado por un URI HTTP que describe al menos a los recursos del *Cuerpo* y el *Target* involucrados en la anotación, así como también cualquier propiedad y relación adicional (por ejemplo, dcterms:*creator*). El URI HTTP de la anotación devuelve la serialización de la anotación en formato RDF.

La especificación de OAC se encuentra actualmente en la fase Beta. Los trabajos futuros sobre la misma se centran en dar suporte a *cuerpos* estructurados que vayan más allá de referencias a recursos, así como el soporte al procesamiento de las restricciones del modelo.

2.3 RIAs

La sección anterior pone de manifiesto que el despliegue en entornos web de aplicaciones referentes a anotación de recursos debe ofrecer experiencias de usuario similares a las permitidas por las aplicaciones de escritorio. A este respecto, es posible adoptar el modelo RIA como modelo básico de desarrollo.

Las llamadas *aplicaciones ricas de internet* o RIAs (*Rich Internet Application*s) son un nuevo tipo de aplicación web que ha surgido para superar las limitaciones de las aplicaciones web tradicionales en lo referente a la usabilidad e interactividad en las interfaces de los usuarios. Su desarrollo está impulsado por nuevos lenguajes y *frameworks*. Sin embargo la ingeniería de este tipo de aplicaciones es un campo relativamente nuevo en el área de la Ingeniería de Software. Es por esto que este apartado presenta una discusión sobre las definiciones de las RIA, las características y la situación actual respecto a los enfoques de ingeniería de las aplicaciones RIA y las tecnologías utilizadas para su implementación.

2.3.1 Conceptos Básicos

En base a alguna de las definiciones existentes de RIAs (Busch & Koch, 2009), es posible proporcionar una definición unificada, que toma en cuenta las características de una RIA, y la experiencia del usuario. De esta forma, las RIAs son aplicaciones web que utilizan datos que pueden ser procesados tanto por el servidor como por el cliente. Además, el intercambio de datos se lleva a cabo de manera asíncrona con el fin de que el cliente se mantenga

receptivo a sus eventos, mientras que continuamente se recalcula y actualiza partes de la interfaz de usuario. Del lado del cliente, una RIA ofrece un *look-and-feel* similar al de una aplicación de escritorio. De esta forma, las RIAs se caracterizan fundamentalmente por una variedad de controles de operación interactiva, la posibilidad de uso online/offline de la aplicación y el uso transparente de conexión entre cliente y servidor.

En una RIA existe una consistente secuencia de eventos que conllevan a la llamada asíncrona y a la recuperación de datos, frente al modelo transaccional propio de una aplicación web más convencional (Figura 5). De hecho, en una RIA el *Navegador/Cliente* carga la página, ésta se actualiza y refresca dinámicamente, mediante peticiones asíncronas al servidor, y mediante el uso de la información devuelta por dichas peticiones refresca el estado de la interfaz de usuario. De esta forma, las principales características de una RIA son las siguientes:

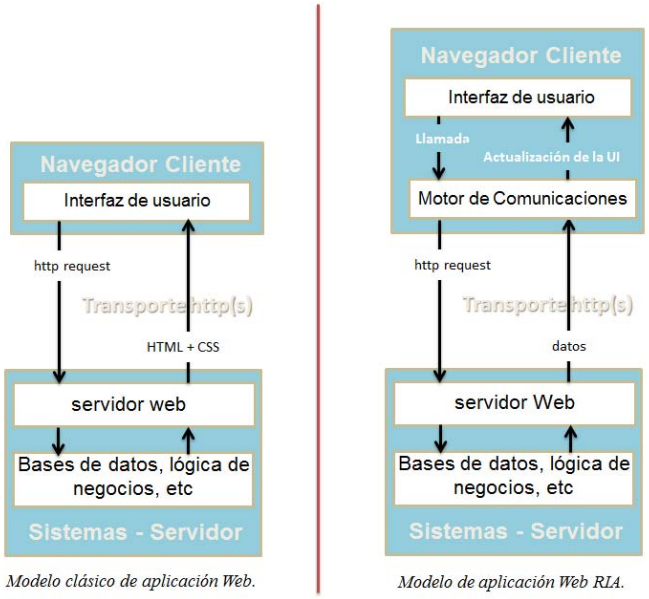

Figura 5 Modelo Tradicional vs Modelo RIA (Jay & Nezlek, 2007)

- *Distribución de datos***.** En las aplicaciones web tradicionales, los datos residen en el servidor. En las aplicaciones RIA los datos pueden ser distribuidos entre el servidor y cliente. El desarrollador puede decidir sobre la distribución e incluso diseñar una aplicación que temporalmente puede emplearse con independencia del servidor. Por lo tanto, una RIA puede usar el contenido persistente y/o volátil del cliente. Los datos pueden ser manipulados en el cliente, y finalmente se envían al servidor una vez que la operación se haya completado. La distribución de datos entre cliente y servidor ofrece como ventajas el potencial uso *off-line* de las aplicaciones, así como la validación y preparación de los datos del lado del cliente. Por su

parte, esta distribución presenta como desventajas diversos problemas bien conocidos en el campo de base de datos: replicación de datos, políticas de alojamiento, consistencia de datos, etc. (Busch & Koch, 2009).

- *Distribución en el* Cómputo *de Página*. En las aplicaciones web tradicionales, sólo hay un controlador en el lado del servidor, que organiza el cálculo de páginas. En cada interacción con el usuario, toda la página se recompone y recarga partiendo desde cero. En las RIAs, se introduce un segundo controlador en el lado del cliente el cual es responsable del cálculo y la actualización de una porción de la página. El procesamiento de datos puede ser ejecutado tanto en el cliente como en el servidor. Es por esto que las RIAs tienen una estructura de navegación diferente a las aplicaciones Web tradicionales. Debido a la capacidad de procesamiento del lado cliente, en un RIA es común ver que el cliente lleva a cabo operaciones complejas.
- *Comunicación cliente-servidor*. En las RIAs, se introducen mecanismos para reducir al mínimo las transferencias de datos que controlan las capas de interacción y presentación del servidor al cliente. Al contrario de las aplicaciones web tradicionales, las RIAs usan tanto comunicaciones síncronas como asíncronas. La distribución de datos y funcionalidad a través de cliente y servidor amplía las características de los eventos producidos, ya que pueden ser originados, detectados, notificados, y procesados en diferentes formas. El reordenamiento y la actualización parcial de la página son las ventajas de este tipo de comunicación.
- *Mejora en el comportamiento de la interfaz de usuario.* Las RIAs ofrecen mejoras en el comportamiento e interacción de la interfaz de usuario (por ejemplo: soporte multimedia, animaciones etc.). Esto permite operar la aplicación en una sola página mediante la carga progresiva de presentación evitando actualizaciones innecesarias de toda la página. Sin embargo entre las desventajas de tener un comportamiento más sofisticado en la interfaz de usuario se encuentran problemas de rendimiento e incompatibilidades con navegadores.

Como consecuencia de estas características, las RIAs son más interactivas y proporcionan mejor respuesta que las aplicaciones web tradicionales. De hecho:

- En lo que se refiere a la interactividad, las RIAs evitan la recarga de página, hacen uso intensivo del *drag-and-drop*, exhiben corto tiempo de respuesta, e incluyen elementos atractivos, como animaciones multimedia. Como consecuencia, es posible ofrecer al usuario de una RIA diferentes funcionalidades como validación de formularios instantánea,

cumplimentación automática de formularios, actualización periódica, editores de texto enriquecido, etc.

- En lo referido al tiempo de respuesta, se evidencia una mejora debido a que las aplicaciones sólo envían datos pertinentes al servidor, y reciben menores volúmenes de información en el intercambio (cliente/servidor). Por tanto, el volumen del tráfico de red que se genera se reduce en gran medida. La aplicación sigue comunicándose con el servidor, pero esta vez transmite páginas fraccionadas en lugar de páginas completas. Este enfoque permite pequeñas actualizaciones de forma incremental para reflejar las acciones del usuario. Así, se puede tener interfaces de una sola página para las aplicaciones, en lugar de la interfaz tradicional multi-página. Esto sugiere que los clientes, incluso con conexiones con poco ancho de banda (por ejemplo, *dial-up*) sean capaces de aprovecharse plenamente de la riqueza de la aplicación (Mullet, 2004)
- Por otra parte, las RIAs, que resultan ser programas más complejos que las aplicaciones web tradicionales, requieren también un proceso de desarrollo más complejo. Los requisitos de ingeniería deben hacer frente a la variedad de características que las RIAs pueden ofrecer a los usuarios. El diseño de estas aplicaciones tiene que considerar diferentes tipos de arquitecturas y comunicación para que las interfaces luzcan como aplicaciones de escritorio. De igual forma, tal y como se indica en (Woolston, 2007), una RIA que sea consistente debe ser lo suficientemente genérica como para ejecutarse en la mayoría de los navegadores, al punto de que si se tuviese que instalar una aplicación de escritorio para acceder a los datos, entonces esta aplicación no debería clasificarse como una RIA (un caso que ejemplifica este supuesto es Google Earth).

2.3.2 Tipos de RIAs

Las RIAs se pueden organizar en tres tipos, en función de cómo se desarrollan y despliegan (Jay & Nezlek, 2007):

- El primer tipo, *basado en plug-in*, implica crear la aplicación en una plataforma dedicada (o en un entorno dedicado) y luego desplegarla, ya sea como una solución integrada o una aplicación *stand-alone* lanzada desde el navegador. Un ejemplo del enfoque lo proporciona la tecnología Flex de Adobe 2 (Kazoun & Lott, 2007), que utiliza un lenguaje de marcado basado en XML llamado MXML para crear declarativamente aplicaciones basadas en Flash.

Estas pueden incluirse en las páginas HTML utilizando etiquetas HTML estándar. Un ejemplo del enfoque de aplicación *stand-alone* es Java Web Start (Jay & Nezlek, 2007) Java Web Start utiliza el Protocolo de Lanzamiento de Red Java (del inglés - JNLP -) con el fin de desplegar una aplicación Java desde el navegador. De esta manera los desarrolladores pueden asegurarse de que la versión más actualizada de la aplicación esté siempre disponible. La mayor ventaja del enfoque *basado en plug-in* es la simplicidad de su desarrollo, además de proporcionar garantía de que la aplicación se ejecute de la misma manera a través de múltiples plataformas, siempre y cuando el *plug-in* correcto esté disponible. De hecho, los applets de Java se encuentran entre las primeras RIAs creadas para la web. Otros ejemplos que dan soporte a herramientas de desarrollo de RIA basadas en *plug-in* incluyen Microsoft Windows Smart Client[12] y Open Laszlo[13].

- El segundo tipo de RIA, RIAs AJAX, es el más popular. Las RIAs creadas con las técnicas de AJAX (Jay & Nezlek, 2007) se conocen como RIAs "basadas en script". Estas aplicaciones utilizan una combinación de tecnologías que, por lo general incluyen XHTML / HTML, CSS, DOM y JavaScript. La idea es utilizar CSS y HTML para dar estilo y presentar la interfaz, y finalmente el uso de JavaScript para realizar solicitudes asincrónicas al servidor. Esta estrategia confiere a la aplicación un tamaño mínimo y no requiere pre-instalar ningún tipo de software. Sin embargo, algunas de las tecnologías tienen bien documentados problemas de compatibilidad, basados en los entornos de la plataforma y navegador. Debido a estas incompatibilidades, los desarrolladores deben actuar con cautela para evitar los casos en donde partes de código se ejecuten diferentemente entre navegadores. La estrategia más exitosa para el desarrollo de aplicaciones estilo AJAX es la de asumir que el navegador del usuario no tiene acceso a JavaScript. Primero, la aplicación se desarrolla utilizando el modelo tradicional de una aplicación de Internet. Luego se utiliza JavaScript para comprobar si éste está habilitado en el cliente para poder modificar los componentes de la página según sea necesario. Este proceso es largo y complejo, y requiere un profundo conocimiento de las varias incompatibilidades que existen entre navegadores. Debido a esta complejidad, se han desarrollado una gran variedad de *frameworks* con el fin de aliviar estos tipos de problemas.

[12] http://msdn.microsoft.com/smartclient/
[13] http://www.openlaszlo.org/

- El último y menos popular tipo de RIA, es el "basado en el navegador". Estos, por lo general, incorporan un lenguaje de interfaz de usuario, basado en XML, que permite a los desarrolladores especificar los elementos necesarios y sus interacciones en un formato declarativo, adaptando, de esta forma, el navegador a las características específicas de la RIA. Un ejemplo es el lenguaje XUL (Feldt, 2009) de la Fundación Mozilla[14]. La principal ventaja de este enfoque es la base que tiene sobre estándares XML apoyados por el W3C como CSS 1 y 2, los niveles de DOM 1 y 2, y JavaScript 1.5 +. XUL está diseñado para ser independiente de la plataforma, lo que permite su portabilidad.

2.3.3 Dominios de Aplicación para las RIAs

A diferencia de las aplicaciones web habituales, las RIAs enriquecen la experiencia del usuario mediante sus interfaces de actuación (similares a las aplicaciones de escritorio), al ser más interactivas y con mayores capacidades gráficas y multimedia, con lo cual el dominio de las RIAs se encuentra en cualquier tipo de aplicación que interseque las aplicaciones de escritorio, con las aplicaciones web, como se muestra en la Figura 6. Alguno de los dominios más populares de aplicación de las RIAs son los siguientes:

Figura 6 Dominio de Aplicación RIA

[14] http://www.mozilla.org

- *Gestoría de* mensajes *de correo electrónico*. Un ejemplo típico es Gmail[15], la aplicación de Google para la gestión del correo electrónico. Esta RIA tiene un interfaz web que permite al usuario efectuar acciones sobre su correo igual que si estuviera utilizando un programa cliente instalado localmente (ver Figura 7).

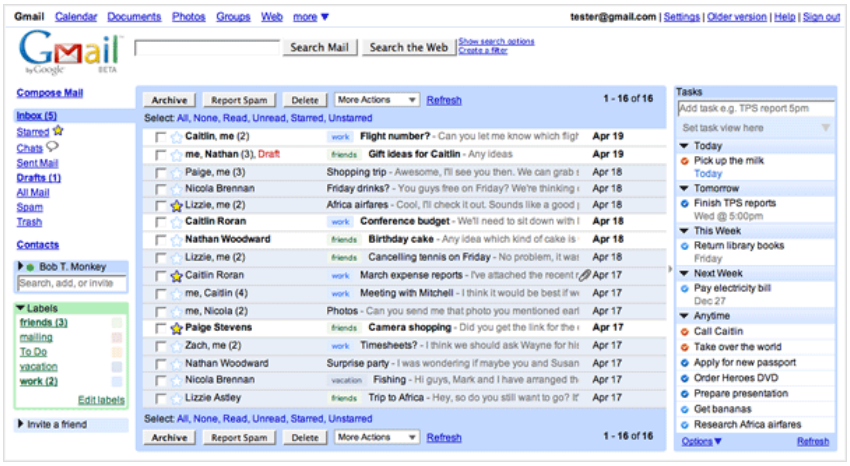

Figura 7 Gmail: Gestor de correos electrónicos

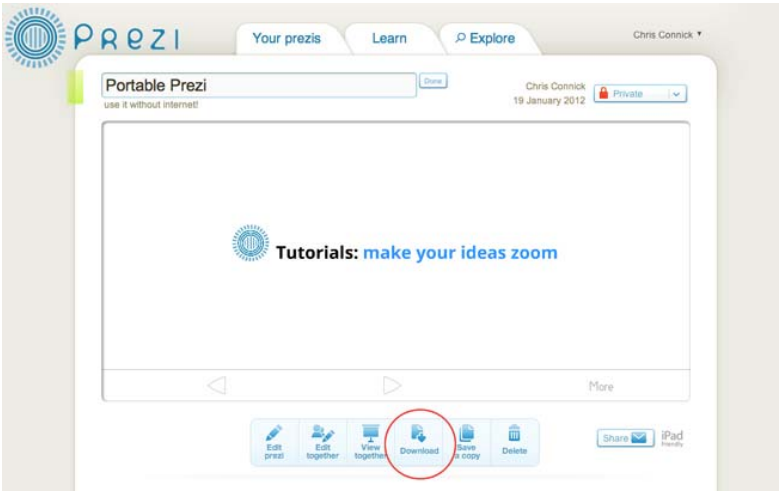

Figura 8 Interfaz de Prezi

- Presentaciones *online*. Otro ejemplo popular es el de las RIAs capaces de crear de forma online presentaciones al estilo *flash*. Tal es el caso de Prezi[16], una aplicación de presentación *online*, que utiliza un solo canvas en vez de diapositivas tradicionales y separadas. Los textos, imágenes, video u otros objetos de presentación se disponen en un lienzo infinito y se presentan ordenadamente en planos. El lienzo permite a los usuarios crear una presentación

[15] http://gmail.com
[16] http://prezi.com

no lineal, donde pueden utilizar acercamientos en un mapa visual. La presentación final se puede desarrollar en una ventana del navegador, como si se tratase de una aplicación de escritorio. La Figura 8 muestra la interfaz de Prezi.

- Gestión *de documentos y hojas de cálculo*. Google Docs & Spreadsheets es un programa gratuito basado en Web para crear documentos en línea con la posibilidad de colaborar en grupo. Incluye un procesador de textos, una hoja de cálculo, un programa de presentación básico y un editor de formularios destinado a encuestas (ver Figura 9). A partir de enero de 2010, Google comenzó a aceptar cualquier archivo en Google Docs, incorporándose al negocio del almacenamiento online con un máximo de 1 GB (con expansiones por costos adicionales).

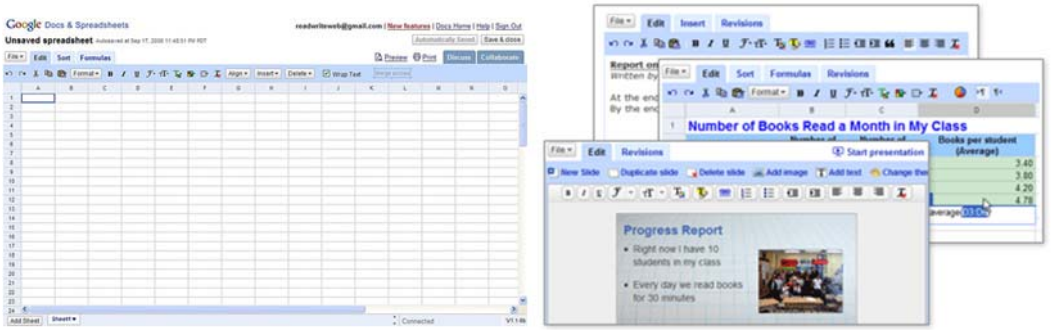

Figura 9 **Google Spreadsheets (izquierda), Google Docs (derecha)**

- *Gestión de Música*. Una RIA de este tipo es Last.fm[17]. Last.fm es una red social, una radio vía Internet y además un sistema de recomendación de música que construye perfiles y estadísticas sobre gustos musicales, basándose en los datos enviados por los usuarios registrados (ver Figura 10). En la radio se puede seleccionar las canciones según las preferencias personales (de acuerdo a un algoritmo y a las estadísticas) o de otros usuarios. Un usuario de Last.fm puede construir un perfil musical usando dos métodos: escuchando su colección musical personal en una aplicación de música con un *plugin* de *Audioscrobbler*, o escuchando el servicio de radio a través de Internet de Last.fm, normalmente con el reproductor de *Last.fm*. Las canciones escuchadas se añaden a un registro desde donde se calcularán los gráficos de barras de los artistas y canciones favoritos del usuario, además de las recomendaciones musicales. La página del usuario también muestra las pistas más

[17] http://last.fm

recientemente escuchadas, y está disponible vía servicios web, permitiendo a los usuarios mostrarlas en blogs o como firmas en foros.

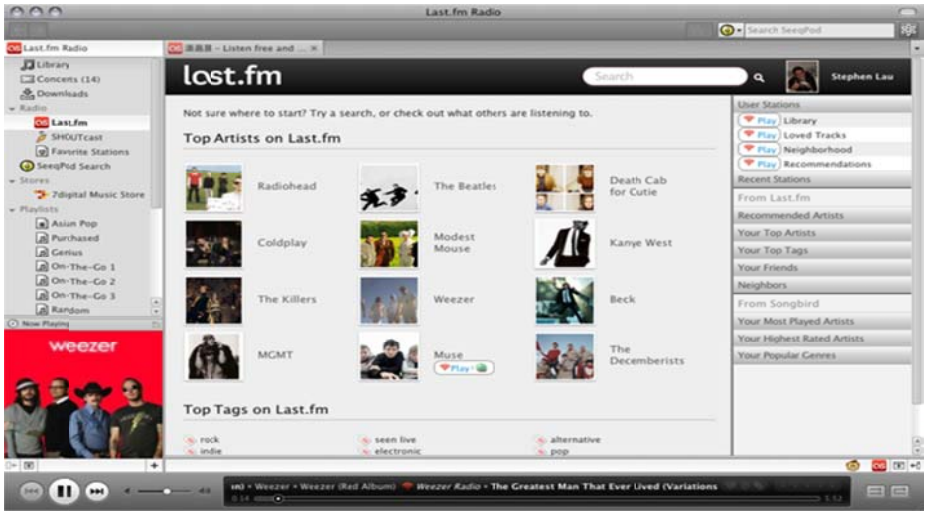

Figura 10 Interfaz de Last.fm

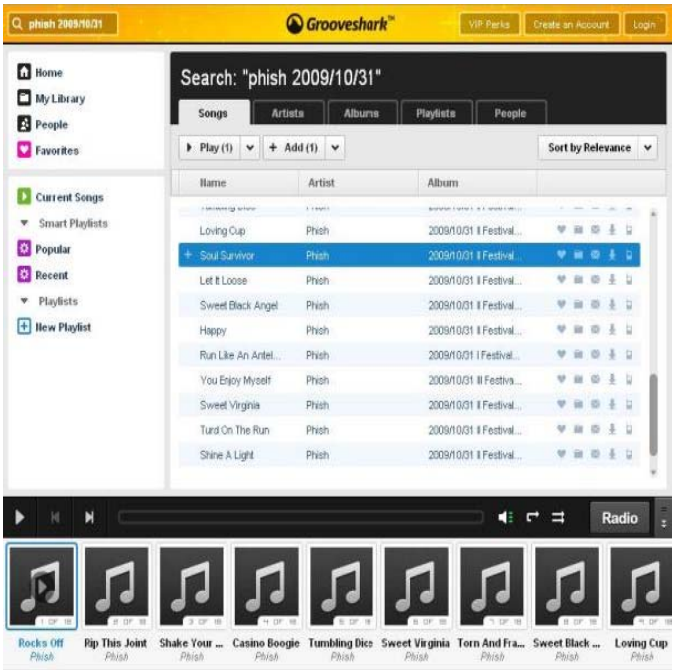

Figura 11 Interfaz de Grooveshark

Otra RIA en este dominio es Grooveshark[18], una organización internacional que tiene como base un amplio motor de búsqueda de música *online* y recomendación de la misma, y que

[18] http://grooveshar.com

permite a los usuarios buscar y subir música de forma libre y gratuita (véase Figura 11). Actualmente tiene un flujo de 50 hasta 60 millones de canciones al mes, además de 10.000.000 de usuarios registrados. Existe la posibilidad de enlazar la cuenta de Grooveshar con terceras cuentas, como por ejemplo, contactos de Gmail y Last.fm.

- RIAs en la *educación*. Un dominio de aplicación típico de las RIAs son los laboratorios remotos o *WebLabs* (García-Zubia et al., 2009). Este tipo de laboratorios están creciendo en importancia y muchas facultades están desarrollando esta tecnología para mejorar su oferta educativa. En este contexto, el trabajo mostrado en (García-Zubia et al., 2007) describe una RIA que permite el control remoto de los parámetros de un regulador PID. En la Universidad de Deusto, por otra parte, existen distintas iniciativas similares (WebLab-DEUSTO), orientadas a materias que requieren acceso a un CPLD o una FPGA (CPLD y FPGA Xilinx XC2S144)[19]. En particular, las tareas de las asignaturas de "Lógica Programable" y "Diseño Electrónico" del tercero y quinto año de la Ingeniería Automática y Electrónica, respectivamente, se llevan a cabo con la ayuda de WebLab-DEUSTO.

  Otro ejemplo de RIA en educación, esta vez en el dominio de la anotación de textos literarios digitalizados, es DLNotes. DLNotes (Roch et al., 2009), es un sistema de anotación para bibliotecas digitales (DLs, del inglés *Digital Libraries*) que permite la creación y el intercambio de conocimientos sobre los contenidos de las DLs enfocadas en actividades de *e-learning*. Mediante el uso DLNotes, instructores y alumnos libremente pueden enriquecer los contenidos de la DL, mediante la asociación de partes relevantes de documentos con anotaciones de texto libre y/o semántico. Las instancias y las relaciones generadas por el proceso de anotación semántica de DLNotes se almacenan y relacionan con bases de conocimientos que pueden ser utilizadas para la navegación semántica así como para la recuperación de información semántica. DLNotes también soporta colaboración gracias a anotaciones públicas y foros de discusión relacionados con las anotaciones. Los hilos de discusión son mecanismos particularmente importantes debido a que permiten la comunicación entre los alumnos e instructores sobre los temas comentados. Por otra parte, mediante el uso de DLNotes, los instructores pueden proponer tareas a sus estudiantes, tales como la construcción colaborativa de un conjunto de anotaciones. La ejecución de estas tareas puede ser guiada y evaluada por los instructores.

[19] http://weblab.deusto.es; http://weblab-pld.deusto.es; http://weblab-fpga.deusto.es

2.3.4 Métodos para el Desarrollo de RIAs.

La necesidad de tener un sustento en la Ingeniería de Software para el desarrollo de RIAs ha sido abordada recientemente mediante varios métodos. Para tal fin, es posible distinguir cuatro tipos de enfoques (Busch & Koch, 2009):

- La extensión de un método ya existente que permita adoptar las características de una RIA para el modelado, transformaciones adicionales y mecanismos específicos de generación, como OOH4RIA, OOHDM, o WebML.
- La combinación de un método para el desarrollo de aplicaciones web con un método para el diseño y generación de interfaces de usuario de RIAs, como por ejemplo, RUX combinado con WebML.
- Nuevos método para el diseño e implementación de RIAs, como, por ejemplo, el enfoque ADRIA (Martínez-Ruiz et al., 2007).
- Enfoques basados en patrones.

A continuación se analizan con más detalle los métodos más característicos de los distintos enfoques.

2.3.4.1 Extensión de Métodos ya Existentes

En esta sección se analizan algunas extensiones para RIA a métodos de desarrollo de aplicaciones web ya existentes. En concreto, la discusión se centra en OOH4RIA, OOHDM y WebML.

2.3.4.1.1 OOH4RIA

OOH4RIA (Object-Oriented Hypermedia For RIA) (Trias et al., 2010), representa una propuesta para el modelado de RIAs y que extiende el método OO-H (Gómez & Cachero, 2003) (que utiliza los mismos elementos y apariencia visual que un diagrama de clases UML, pero define su propio metamodelo para introducir extensiones), el cual adopta un enfoque CRUD (Create, Read, Update and Delete) para definir los modelos de dominio y navegación que permiten generar la parte servidora. OOH4RIA se centra en el aspecto de la UI proponiendo 2 nuevos modelos, el modelo de presentación y el de orquestación, así como transformaciones necesarias que representan la parte cliente RIA:

- El modelo de presentación organiza y da estructura a la UI. Para ello, introduce los conceptos de *page*, *screenshot* y *widget*.
- Por otro lado, el modelo de orquestación describe el comportamiento existente entre los *widgets* que componen la UI y el sistema.

El proceso de construcción de la RIA empieza con la definición del modelo conceptual de datos. A partir de éste se obtiene de forma automática el modelo de navegación. A continuación, se genera el modelo de presentación mediante una transformación M2M (*modelo-a-modelo*), y seguidamente el modelo de orquestación. Finalmente a partir de estos dos últimos modelos se genera el código que implementa la UI. La Figura 12 presenta el proceso completo de modelado de OOH4RIA.

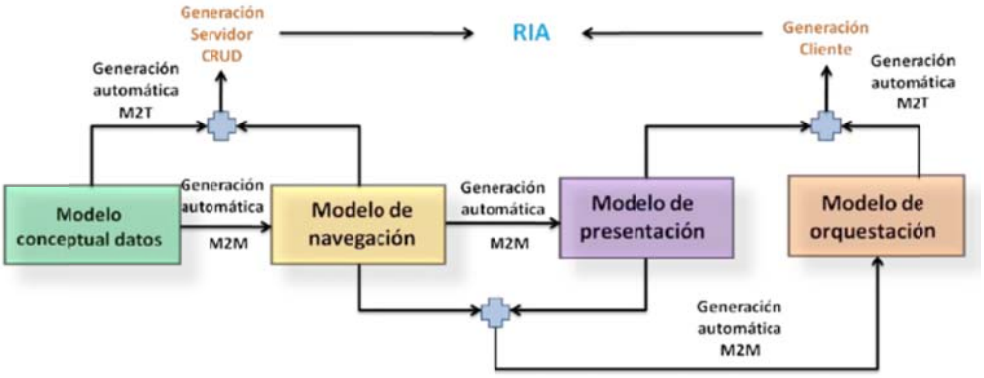

Figura 12 Proceso de modelado de OOH4RIA (Trias et al. 2010)

2.3.4.1.2 OOHDM

El método OOHDM (Schwabe & Rossi, 1998) (Object-Oriented Data Model) también extiende a OOH para dar soporte al desarrollo de las RIAs, y concretamente para el modelado e implementación de la UI. El proceso de modelado empieza con el modelo conceptual de datos, seguido del modelo de navegación. Para el modelado de la UI, se construye el modelo de vista de dato abstracto ADVs (*Abstract Data View*) (Cowan & Lucena, 1995) que permite definir la estructura de la UI y los elementos que la conforman. Para modelar su comportamiento el método propone la utilización del modelo ADV *Chart*, un diagrama de estados que permite modelar los cambios que experimenta la UI causados por los eventos detectados. A partir del modelo de ADVs se obtiene la implementación de la interfaz en una cierta plataforma. La Figura 13 ilustra el proceso de desarrollo propuesto por OOHDM.

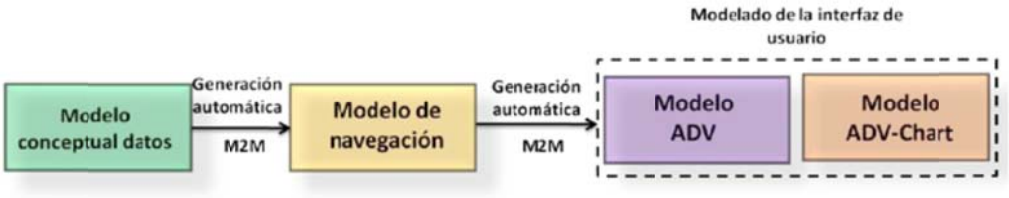

Figura 13 Proceso de Modelado de OOHDM (Trias et al., 2010)

2.3.4.1.3 Extensión WebML.

WebML (Web Modeling Language) es un método dirigido por modelos que también se ha adaptado al desarrollo de RIAs. Se centra en el aspecto que permite al cliente almacenar datos y ejecutar la parte de la lógica de negocio (Toffetti, 2007). El proceso de modelado de WebML empieza con la definición del modelo de procesos de negocio implementado en BPMN (*Business Modelling Language*) (Wohed et al, 2008). A partir de éste se obtiene de manera automática el modelo conceptual de datos y el modelo de navegación (Brambilla et al., 2008), en donde se realizará el reparto de la lógica de negocio. Las funcionalidades serán etiquetadas para definir en qué parte (cliente, servidor) serán ejecutadas. El aspecto de la presentación no se dirige mediante modelos, sino que se codifica de forma manual mediante código XHTML y hojas de estilo XSLT.

En particular WebML se extiende al enriquecer las especificaciones de los datos añadiendo dos dimensiones: 1) localización de los datos y 2) la duración de los datos. La localización puede ser en el cliente o servidor y la duración puede ser persistente o temporal (Preciado et al., 2007).

2.3.4.2 Combinación de métodos de especificación de UIs *de* RIAs *con métodos de desarrollo web*

RUX (Linaje et al., 2007) es un método dirigido por modelos que cubre el aspecto relativo al diseño de interfaz de las RIA basado en el *framework* de referencia Cameleon (Calvar et al., 2003), de acuerdo al cuál una UI se puede descomponer en cuatro niveles: *conceptos y tareas*, *interfaz abstracta*, *interfaz concreta*, e *interfaz final*.

El proceso de desarrollo en el método de RUX se resume en la Figura 14. Dicho desarrollo parte de un *modelo web subyacente*, y, a través de una transformación (marcada como CR en la Figura 14), genera una representación abstracta de la interfaz. Dicha representación es

transformada, a su vez, en una interfaz concreta (transformación TR1 en la Figura 14), y ésta, a su vez, en una interfaz final (transformación TR2). De esta forma, en el método RUX cada etapa se alimenta de la anterior, utilizando para ello las posibilidades de transformación proporcionadas por la *Component Library* del método de RUX (también en la Figura 14).

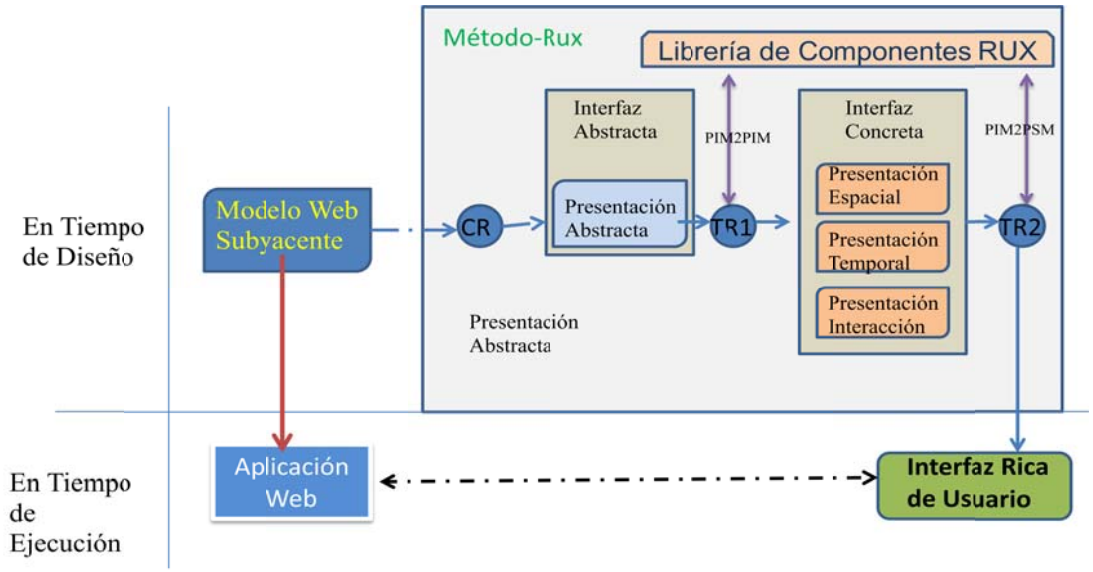

Figura 14 Visión general del método RUX (Linaje et al., 2007)

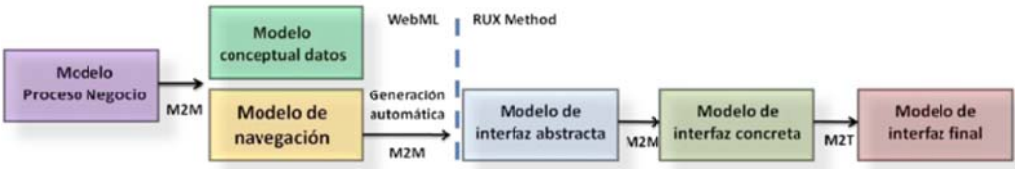

Figura 15 Proceso de desarrollo de la combinación entre WebML y RUX (Trias et al., 2010).

Mientras que RUX especifica cómo desarrollar la UI de una RIA a través de un proceso transformacional, no define la naturaleza de los modelos web subyacentes. Tales modelos pueden ser desarrollados utilizando, por tanto, otros enfoques. En particular, en (Preciado et al., 2007b) se propone combinar RUX con WebML como método completo de desarrollo de RIAs. De esta forma, a partir del modelo de navegación WebML se genera el modelo de interfaz abstracta de RUX, y, a partir de aquí, empieza el desarrollo RUX de la UI. La Figura 15 muestra el proceso de desarrollo de esta combinación.

2.3.4.3 Enfoques específicos para el desarrollo de RIAs.

Un ejemplo de método específicamente orientado al desarrollo completo de RIAs es ADRIA (*Abstract Design of* RIAs) (Dolog & Stage, 2007).

ADRIA divide el diseño de una aplicación web en tres actividades, que se ocupan, respectivamente, de la estructura, comportamiento y el aspecto de la aplicación, tal y como se ilustra en la Figura 16.

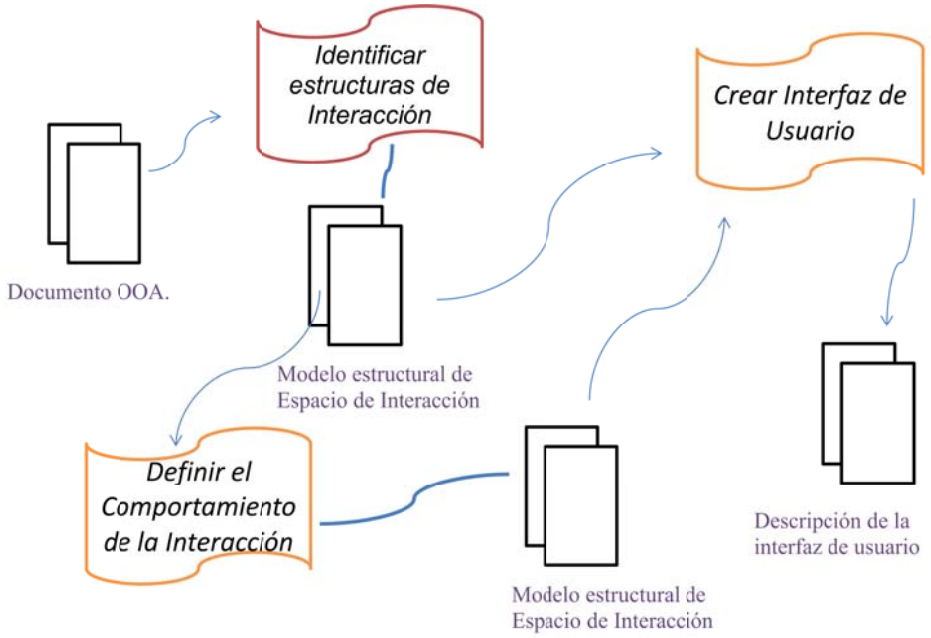

Figura 16 Actividades de ADRIA (Dolog & Stage, 2007)

El concepto clave en las tres actividades de ADRIA es el de "espacio de interacción", que se define como un elemento identificable del sistema que soporta una parte específica de la interacción del usuario. Un ejemplo típico de un espacio de interacción es una ventana, pero también puede ser una parte más pequeña de una ventana, como un panel con un menú o algo completamente diferente, como un teclado que se utiliza para introducir datos específicos. La idea de tratar con los espacios de interacción en el diseño es aplazar las decisiones sobre el aspecto concreto de los elementos de la interfaz de usuario hasta que la interacción con el usuario haya sido diseñada en un nivel global.

2.3.4.4 Enfoques basados en patrones.

Estos enfoques parten de las recomendaciones de diseño basadas en buenas prácticas provenientes de las formas de interacción que van surgiendo, por ejemplo, en aplicaciones AJAX (Mahemoff, 2007), y estas son plasmadas en patrones UML específicos para el desarrollo web, utilizando, por ejemplo, perfiles de aplicación como UWE (UML-based Web Engineering) (Koch et al., 2008), con el fin de lograr mayores niveles de abstracción.

2.3.5 Tecnologías para la Implementación de RIAs.

En esta sección se revisan algunas de las tecnologías más relevantes empleadas en la implementación de RIAs: AJAX, bibliotecas JavaScript específicas, ICEfaces, Flex, Silverlight y HTML5. Esta sección excluye las tecnologías de Google, que se tratarán en la siguiente sección debido a su relevancia en el contexto de este trabajo.

2.3.5.1 AJAX

AJAX es un acrónimo de *Asynchronous JavaScript + XML*, que se puede traducir como "JavaScript asíncrono + XML", introducido por primera vez en (Garret, 2007). Tal y como se indica en dicho trabajo, AJAX no es una tecnología en sí mismo, sino una combinación de tecnologías existentes, que, unidas, facilitan el desarrollo de RIAs. Más concretamente, AJAX integra las siguientes tecnologías (véase Figura 17)

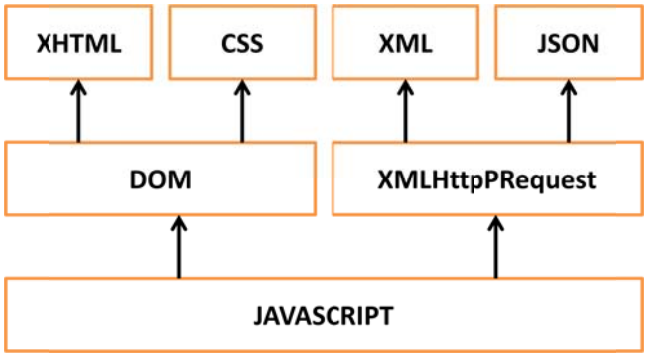

Figura 17 Tecnologías agrupadas bajo el concepto de AJAX (Garret, 2007)

- XHTML y CSS, para crear presentaciones basadas en estándares.
- DOM, para permitir la interacción y la manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- XMLHttpRequest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.

De esta forma, las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.

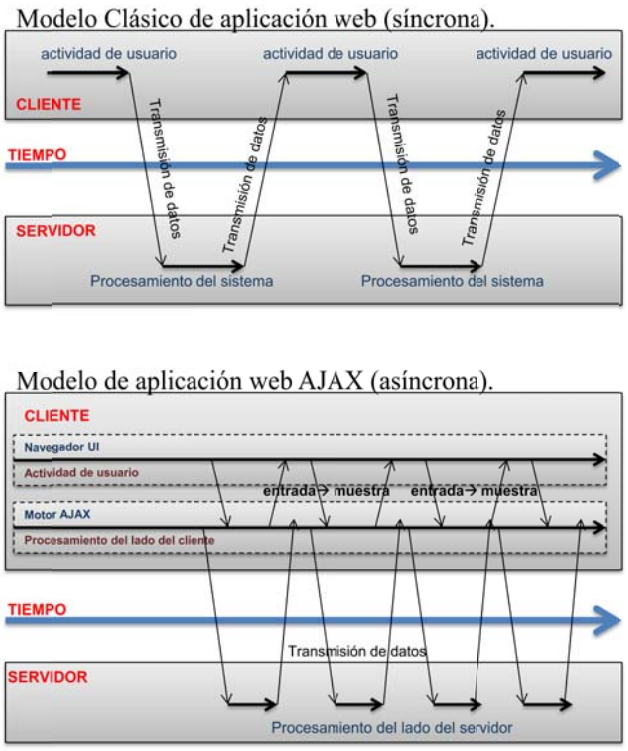

Figura 18 Comparación entre las comunicaciones síncronas de las aplicaciones web tradicionales y las comunicaciones asíncronas de las aplicaciones AJAX (Garret, 2007)

La Figura 18 muestra la diferencia más importante entre una aplicación web tradicional y una aplicación web creada con AJAX. La imagen superior muestra la interacción síncrona propia de las aplicaciones web tradicionales. La imagen inferior muestra la comunicación asíncrona de las aplicaciones creadas con AJAX, que es, tal y como se ha indicado ya, el patrón de comunicación típico en una aplicación RIA.

En AJAX, las peticiones al servidor se realizan a través de invocaciones a servicios JavaScript proporcionados por el elemento encargado de gestionar la comunicación AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona, mediante HTTP, obteniéndose, como resultado, un cuerpo de datos (codificado, por

ejemplo, en XML o en JSON), que se utiliza para actualizar alguna porción de la página. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor.

Desde su aparición, se han creado cientos de aplicaciones web basadas en AJAX. En la mayoría de casos, AJAX puede sustituir completamente a otras técnicas como Flash. Además, en el caso de las aplicaciones web más avanzadas, pueden llegar a sustituir a las aplicaciones de escritorio.

2.3.5.2 Librerías JavaScript para el Desarrollo de Aplicaciones Web.

Las librerías JavaScript fueron una de las primeras tecnologías que ayudaron realmente a desplegar aplicaciones web ricas e interactivas. Proporcionan un marco para una RIA que utilizan componentes del lado del cliente para manejar las funciones de *front-end* de la interfaz. Son básicamente ficheros JavaScripts que forman parte de grandes colección de controles. Tales librerías pueden combinarse también con AJAX, así como gestionar las interacciones comunes de usuario, como ocultar y mostrar el contenido basado en un evento realizado por el usuario.

Algunas de las librerías JavaScript más populares hoy en día son jQuery, MooTools, YUI (Yahoo! User Interface library), y ExtJS. Todas estas librerías incluyen componentes de RIA como mallas, gráficos y elementos complejos para formularios, así como funcionalidades para manejar AJAX. Cabe decir que la mayoría de estas librerías son de código abierto.

Algunos sitios que utilizan este tipo de librerías de JavaScript son Google, Digg, Yahoo, Amazon, Microsoft, Twitter, y Best Buy.

2.3.5.3 ICEfaces

ICEfaces es un *framework* de código abierto para el desarrollo de RIAs, y constituye un marco de desarrollo basado en JavaServer Faces (JSF) 2 estándar. ICEfaces extiende a JSF para simplificar el desarrollo y mejorar el conjunto de funcionalidades estándares de JSF, al mismo tiempo facilitando el desarrollo y ampliando el espectro de las capacidades de RIA que se pueden incluir en cualquier aplicación web basada en JSF.

Las RIAs se construyen a partir de componentes ricos. ICEfaces cuenta con 3 tipos de componentes adaptables al tipo de sistema:

- Componentes avanzados ICEfaces (ACE), dirigidos a navegadores modernos.
- Componentes ICEfaces del ICE, aptos para navegadores antiguos.

- Componentes ICEfaces corporativos (EE), que constituyen un conjunto de componentes comerciales orientados a empresa.

El desarrollo de aplicaciones con ICEfaces es esencialmente un desarrollo en JSF - Java EE, el cual promueve una arquitectura basada en componentes que utiliza definiciones ordinarias de UI basadas en tags, así como un enlace de datos dinámicos en el servidor de aplicaciones. ICEfaces proporciona a los desarrolladores de aplicaciones un modelo de desarrollo familiar al desarrollador Java Enterprise, permitiendo ocultar totalmente el desarrollo de JavaScript. En otras palabras, ICEfaces maneja todo el código JavaScript / AJAX de aplicación Web a través de una API de Java. Esto simplifica enormemente la tarea de crear aplicaciones RIA mediante la eliminación de algunas de las complejidades introducidas por la construcción de funciones JavaScript.

Algunos sitios que utilizan ICEfaces son Boeing, NASA, Union Pacific, T-Mobile, y Bank of America.

2.3.5.4 Flex

Flex permite el desarrollo programático de componentes ejecutables en un *Flash player*. Funcionalmente, las aplicaciones Flex son similares a las aplicaciones AJAX, ya que ambos son capaces de actualizaciones dinámicas en la interfaz de usuario y ambos incluyen la capacidad para enviar y cargar datos de manera transparente. Sin embargo, Flex va más allá al proporcionar un conjunto más rico de controles y componentes de visualización de datos, con el fin de permitir a los desarrolladores construir y desplegar RIAs en la Plataforma Flash. De esta forma, actualmente Flex se basa en varias tecnologías clave:

- ActionScript 3.0, un potente lenguaje de programación orientado a objetos, diseñado para la construcción rápida de RIAs. Aunque las versiones anteriores de ActionScript ofrecían la potencia y flexibilidad necesaria para crear atractivas experiencias web, ActionScript 3.0 evoluciona el lenguaje, mejorando el rendimiento y la facilidad de desarrollo de complejas aplicaciones que manejan grandes conjuntos de datos. Así mismo, promueve el desarrollo orientado a objetos y el desarrollo de código reutilizable.
- MXML, una sintaxis XML declarativa que permite la creación y mantenimiento simplificado de interfaces de usuario. MXML proporciona una manera fácil de visualizar e implementar la naturaleza jerárquica de una interfaz de usuario compleja sin la necesidad de escribir y

mantener directamente código ActionScript. MXML se compila automáticamente en ActionScript durante el proceso de construcción de una aplicación Flex (Labriola & Tapper, 2011)

Algunos sitios que utilizan Flash para sus aplicaciones web son Mint.com, Flickr, y Hyundai.

2.3.5.5 Silverlight

Silverlight es un *framework* para RIAs desarrollado por Microsoft. Sus tecnologías incluyen:

- Silverlight: Un marco para la creación de RIAs, escrito en un subconjunto de. NET.
- XAML: Un lenguaje basado en XML en el que se declara interfaces de usuario. XAML es una propuesta análoga al lenguaje MXML de Flex, discutido anteriormente.
- *Microsoft Expression Studi*o, una herramienta de diseño profesional destinada a crear aplicaciones de escritorio para clientes Windows, así como aplicaciones web específicas destinadas al reproductor de Silverlight.

Silverlight está comenzando a ofrecer una plataforma atractiva, especialmente para los desarrolladores. NET que pueden utilizar herramientas como *Microsoft Visual Studio* y aprovechar el conocimiento existente para crear rápidamente nuevas aplicaciones. La única limitación que Microsoft parece tener en este momento es la distribución del entorno de ejecución de Silverlight, que es todavía limitada, incluso para Windows (y, desde luego, para plataformas Mac y Linux). De acuerdo con `riastats.com`, el 27 por ciento de todos los navegadores no tienen un reproductor de Silverlight instalado, y casi tres cuartas partes de todas las máquinas que ejecutan Linux no tienen Silverlight (Anderson, 2010).

Un ejemplo web que utiliza Silverlight es Netflix.

2.3.5.6 HTML5

HTML5 es uno de los últimos desarrollos para RIAs, y es, en esencia, el resultado de la fusión de lo mejor de HTML 4, JavaScript, CSS, y las librerías de JavaScript, en una única especificación. HTML5 es una tecnología abierta, lo que significa que no va a ser un único órgano gobernante como Adobe para Flash o Microsoft para Silverlight, por ejemplo, el que controle la evolución de dicha tecnología.

Una mejora clave en HTML5 es el soporte para reproducir archivos de vídeo y audio de forma nativa en el navegador (es decir, sin necesidad de plug-ins). Así mismo, uno de los mayores cambios en HTML5 frente a versiones anteriores de HTML es la adición del elemento *canvas*, que proporciona una superficie de dibujo de propósito general que se puede utilizar para llevar a cabo tareas que, hasta ahora, se solían realizar mediante Adobe flash (Freeman, 2011).

2.4 Tecnologías Google

En esta sección se revisan las tecnologías de Google para el desarrollo de RIAs: GWT (*Google Web Toolkit*) y GAE (*Google Application Engine*). Así mismo, aunque no es estrictamente una tecnología para la construcción de RIAs, se describe también brevemente el API de Google Books, debido al uso que del mismo se hace en este trabajo.

2.4.1 Google Web Toolkit

Google Web Toolkit (GWT) es una herramienta de desarrollo para la creación, optimización y mantenimiento de RIAs. Su meta es la de habilitar el desarrollo productivo de aplicaciones web de alto rendimiento sin que el desarrollador tenga que ser un experto en HTML, JavaScript, AJAX y otras tecnologías web básicas.

GWT proporciona dos herramientas fundamentales para ayudar en el desarrollo de una aplicación, que constituyen la columna vertebral del marco de GWT:

- El *compilador* GWT. Este compilador convierte código Java a JavaScript, transformando la aplicación Java en una aplicación equivalente en JavaScript.
- El *navegador web* incorporado. El marco de GWT ofrece un navegador web incorporado que se ejecuta y depura sus aplicaciones en modo “host”. El navegador ejecuta el código de bytes de Java directamente en su propia máquina virtual de Java, sin necesidad de convertirlo a su equivalente JavaScript. Esto requiere que el navegador tenga una interacción directa con la JVM, que se lleva a cabo por los controles especiales embebidos en el navegador web (Internet Explorer en Windows y Gecko / Mozilla en Linux), teniendo estos controles conectores para trabajar con la máquina virtual de Java.

Así mismo, GWT proporciona dos bibliotecas de clases básicas, que pueden utilizarse en el desarrollo de aplicaciones:

- *La biblioteca de emulación del JRE*. Esta biblioteca soporta un subconjunto de las clases incluidas en la distribución Java estándar. Más concretamente, esta biblioteca es compatible con las clases más importantes de los paquetes `java.lang`, `java.io` y `java.util`. El compilador GWT emitirá mensajes de error si se tratan de utilizar otras clases no soportadas, o métodos no soportados de las clases consideradas.
- La *biblioteca de widgets*. El marco de GWT proporciona una serie de widgets para el desarrollo de la interfaz de usuario, análogos a los proporcionados por marcos como Swing. Los widgets de GWT se dividen en *widgets* contenedores y *widgets* de interfaz de usuario, y representan clases autocontenidas que permiten utilizar controles y componentes como paneles, formularios, botones etc. en las aplicaciones web construidas.

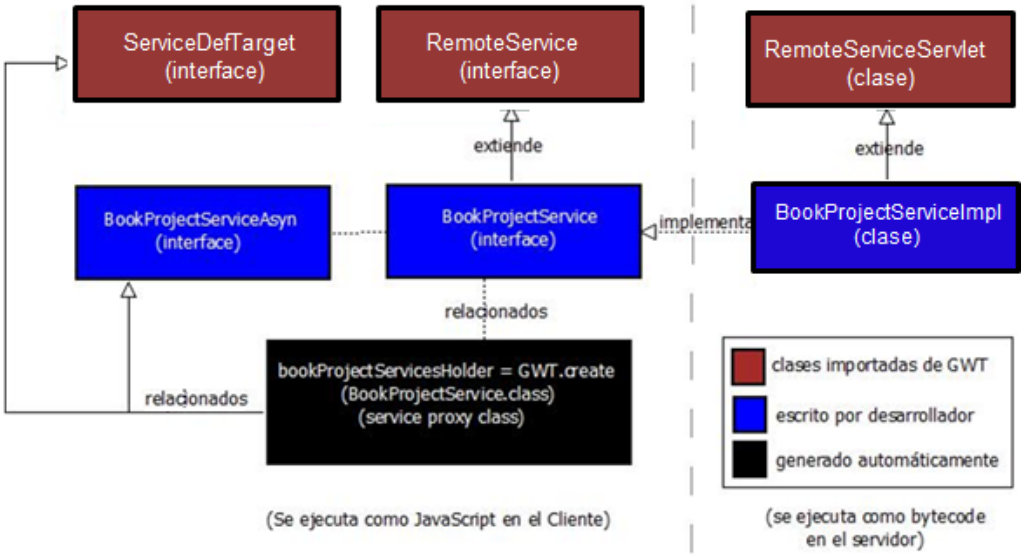

Figura 19 Arquitectura GWT – RPC (Google, 2011)

Como se ha indicado anteriormente, una diferencia fundamental entre aplicaciones AJAX y aplicaciones tradicionales HTML, es que con AJAX, no hay necesidad de "solicitar" una nueva página al servidor para lograr actualizar la interfaz del usuario, sino que dicha actualización puede llevarse a cabo incrementalmente, mediante la realización de peticiones asíncronas al servidor, y de recepción de datos desde el mismo que se utilizan para llevar a cabo las actualizaciones. Sin embargo, como todas las aplicaciones cliente/servidor, las aplicaciones AJAX sí necesitan buscar datos del servidor mientras se ejecutan. Para ello, pueden utilizar RPC (*Remote Procedure Call*) como mecanismo de comunicación básico con el servidor. GWT

soporta RPC y facilita al cliente y servidor pasar objetos Java de un lado a otro a través de HTTP, así como hacer llamadas a *servlets* Java.

La Figura 19 muestra la arquitectura de GWT – RPC, en la cual se puede apreciar una división clara de cliente y servidor.

El lado del cliente utiliza el código que el compilador GWT ha transformado en JavaScript, en donde se definen las cabeceras de los servicios que implementará el servidor, el cual solo tendrá código Java, y devolverá de manera asíncrona al cliente (a través de la interfaz asíncrona que utiliza) el resultado de la ejecución de código del servicio.

Cuando se usa apropiadamente, RPC da la oportunidad de mover la lógica de interfaz de usuario al cliente, resultando en un mejor rendimiento, reduciendo el coste de ancho de banda, así como la carga en el servidor y una mejor experiencia al usuario (Google, 2011).

2.4.2 Google App Engine

Google App Engine (GAE) es un motor completo de desarrollo que utiliza tecnologías familiares para crear y servir aplicaciones web. De acuerdo con la filosofía del GAE, se codifica la aplicación (por ejemplo, utilizando GWT y Java), se prueba en la máquina local y luego se sube a la nube de Google. A continuación se revisan las características más relevantes del GAE.

2.4.2.1 Ejecución de aplicaciones en el GAE

El GAE ejecuta la aplicación web Java a través del JVM Java 6. El punto de entrada de la aplicación en sí es un *servlet*, y ésta se ejecuta en un entorno seguro de ejecución que restringe, entre otros, la escritura en archivos, el uso de *sockets*, la expansión de subprocesos, o la realización de llamadas al sistema. En su lugar, la aplicación debe utilizar un conjunto de servicios proporcionados por el GAE, tales como un servicio de almacenamiento de datos, un servicio de correo, servicio de acceso HTTP y HTTPS, etc.

A fin de invocar adecuadamente el *servlet* de cada aplicación, el GAE examina la URL utilizada en la solicitud. De esta forma, una solicitud cuyo nombre de dominio sea *aplicación-id*.appspot.com se dirige a la aplicación cuyo ID es *aplicación-id*. Todas las aplicaciones obtienen un nombre de dominio *appspot.com* de forma gratuita. Las solicitudes de estas URLs se dirigen todas a la versión de la aplicación que se haya seleccionado como versión predeterminada en la consola del administrador del GAE. Cada versión de la aplicación también incluye su propia URL, de modo que es posible implementar y probar una nueva versión antes de

establecerla como versión predeterminada. La URL específica utiliza el identificador de versiones del archivo de configuración de la aplicación, además del nombre de dominio `appspot.com`, siguiendo el patrón *version-id.aplicación-id*.appspot.com. También se pueden utilizar subdominios con la URL específica de la versión: *subdomain.version-id.aplicación-id*.appspot.com.

El modelo de ejecución en sí es un modelo de ejecución *en la nube*. De esta forma, GAE utiliza varios servidores web para ejecutar una aplicación y ajusta automáticamente el número de servidores en uso para procesar las solicitudes de una forma segura. Una determinada solicitud se puede enviar a cualquier servidor, que no tiene que ser el mismo servidor que procesó una solicitud anterior procedente del mismo usuario.

2.4.2.2 *Servicios ofrecidos por el* GAE

A continuación se revisan brevemente los servicios más relevantes ofrecidos por el GAE a las aplicaciones.

2.4.2.2.1 Almacenamiento de Datos

El almacén de datos del GAE ofrece un servicio de almacenamiento sólido y escalable para las aplicaciones web, con especial atención al rendimiento de las consultas y de las operaciones de lectura. El almacén incluye soporte para dos APIs estándar diferentes para el almacenamiento de los datos: Java Data Objects (JDO) y Java Persistence API (JPA) (Yang, 2010). Tales servicios son proporcionados por la plataforma de acceso *DataNucleus*, una implementación de código abierto de varios estándares de persistencia Java, que incluye un adaptador para la *Datastore* de GAE (Google, 2011).

Es importante destacar que el Datastore de Google se basa en el sistema *BigTable*. Este modelo es más escalable que las bases de datos relacionales, pero no es tan versátil como aquellas, ni soporta ciertas características usuales en sistemas de bases de datos más convencionales, tales como relaciones *many-to-many*, relaciones *unowned*, sentencias con ‘JOIN, GROUP BY, HAVING, SUM, AVG, MAX, MIN’, ni sentencias polimórficas.

2.4.2.2.2 Almacenamiento de Blobs

Este servicio permite a las aplicaciones mostrar objetos de datos de un tamaño de hasta 2 gigabytes. Estos objetos se llaman valores del almacén de *blob*, o *blobs*. Los *blobs* resultan útiles

para mostrar archivos de gran tamaño, como archivos de vídeo o de imagen, y para permitir a los usuarios subir archivos de datos grandes.

Las aplicaciones no crean datos de *blob* directamente, sino que los *blobs* se crean de forma indirecta a través del envío de un formulario web u otra solicitud POST HTTP. Los valores del almacén de *blob* pueden mostrarse al usuario, o la aplicación puede acceder a ellos en una transmisión parecida a un archivo realizada mediante el API del almacén de *blob*.

Los blobs no pueden modificarse después de su creación, aunque sí pueden eliminarse. Cada uno tiene su registro de información correspondiente, guardado en el almacén de datos, que proporciona información detallada sobre el *blob* (por ejemplo, su hora de creación y tipo de contenido).

2.4.2.2.3 Acceso a URLs

Una aplicación puede acceder al contenido de URLs a través del protocolo HTTP (normal) o HTTPS (seguro). Tales URLs estarán restringidas a puertos en los siguientes intervalos: 80-90, 440-450, 1024-65535. Si el puerto no se menciona en la URL, se utiliza 80 para HTTP y 443 para HTTPS. Por su parte, el acceso puede utilizar los métodos HTTP GET, PUT, HEAD y DELETE, y pueden incluir cabeceras de solicitud HTTP y una carga útil (cuerpo de la solicitud HTTP).

El servicio de acceso a URLs es compatible con solicitudes síncronas y asíncronas:

- Con una solicitud síncrona, la llamada al API para acceder a una URL espera hasta que el *host* remoto devuelve un resultado y, luego, devuelve el control a la aplicación. La aplicación puede especificar la cantidad máxima de tiempo que esperará cuando realiza la llamada. Si se excede el tiempo de espera máximo, la llamada genera una excepción.
- Una solicitud asíncrona al servicio de acceso a URLs inicia la solicitud y, a continuación, devuelve de inmediato un objeto. La aplicación puede realizar otras tareas mientras se extrae la URL. Cuando la aplicación necesita los resultados, invoca un método en el objeto, que espera que finalice la solicitud si es necesario y, a continuación, devuelve el resultado.

La aplicación puede tener hasta 10 llamadas de extracción de URL asíncronas simultáneas. Si hay solicitudes de extracción de URL pendientes cuando se cierra el controlador de solicitudes, el servidor de aplicaciones espera a que las solicitudes restantes se devuelvan o bien alcancen su tiempo límite antes de devolver una respuesta al usuario.

2.4.2.2.4 Usuarios

Las aplicaciones GAE pueden autenticar usuarios mediante uno de estos tres métodos: Google Accounts, cuentas de los propios dominios de Google Apps o identificadores OpenID. Una aplicación puede detectar si el usuario actual ha iniciado sesión y puede redirigirlo a la página de acceso correspondiente para que acceda a su cuenta o, si la aplicación utiliza la autenticación de Google Accounts, para que cree una cuenta nueva. Mientras un usuario se encuentre conectado a la aplicación, ésta puede acceder a la dirección de correo electrónico (o identificador OpenID en caso de que la aplicación utilice OpenID). La aplicación también puede detectar si el usuario actual es un administrador, lo que facilitará la implementación de áreas exclusivas de administradores en la aplicación.

2.4.3 Google Books Search API.

Aparte de los servicios ofrecidos por el GAE, las aplicaciones pueden hacer uso de otros servicios ofrecidos por Google, o bien servicios externos. Tales servicios se exponen, normalmente, mediante APIs de servicios *web*, invocables desde las propias aplicaciones GWT. Un ejemplo de tales APIs es la API *Google Book Search*.

La API *Google Book Search* permite a aplicaciones cliente ver y buscar libros en Google Books, así como consultar información estándar del libro. Cada libro incluye una página denominada “Acerca de este libro” con información bibliográfica básica como: título, autor, fecha de publicación, tamaño y tema. Para algunos libros también se puede ver información adicional como: términos clave y frases, referencias del libro por parte de publicaciones académicas u otros libros, títulos de capítulos y una lista de libros relacionados. Los libros que son utilizados, al carecer de derechos de autor, serán entregados para ser vistos completamente, lo cual permite ver cada página del libro; al ser de dominio público, el lector puede descargar también una versión PDF.

Es importante destacar, no obstante, que el funcionamiento de la versión actual de esta API es bastante deficiente, por lo que es necesario, en ciertos casos, usar JavaScript nativo para obtener el comportamiento deseado.

2.5 A Modo de Conclusión

En este capítulo se ha descrito una visión general de las RIAs, en donde se nota un evidente interés por la creación de aplicaciones web que imiten el comportamiento de las

aplicaciones de escritorio convencionales. Cada día son más los desarrollos que apuntan a tener este tipo de comportamientos en sus aplicaciones, y su ámbito de utilización escala diferentes tipos de áreas (medicina, electrónica, entretenimiento, literatura entre otras). En concreto, el campo de las Humanidades Digitales, y el dominio de la anotación de textos literarios digitalizados en particular, son especialmente susceptibles de aprovechar las ventajas brindadas por este tipo de aplicaciones.

Al ser un campo relativamente nuevo en el ámbito de la Ingeniería de Software, han surgido nuevas investigaciones y enfoques para modelar este tipo de aplicaciones. Principalmente se evidencia que la mayoría de grupos de investigación dedicados al modelado de aplicaciones han optado en extender modelos existentes así como unir varios tipos de métodos de modelado con el fin de que la RIA sea lo suficientemente descriptiva. De la misma forma otros autores, para salvar estas limitaciones, han decidido crear nuevos métodos de modelado (ADRIA, por ejemplo). Por otra parte se evidencia la importancia que compañías de software privadas observan en las RIAS, al competir en el desarrollo de tecnologías que permitan crear de manera sencilla aplicaciones ricas (Microsoft, Adobe, Oracle, Google). Se describió en este capítulo de manera detallada las tecnologías que ofrece Google, debido a que fueron las utilizadas en el desarrollo de la aplicación que se describe en este trabajo: @note. Se muestra el gran soporte que ofrece Google para la creación amplia de RIAs, de fácil uso y gran escalabilidad, ofreciendo servicios adicionales que permiten ampliamente desarrollar, sin prácticamente requerir ningún otro tipo tecnología externa, este tipo de aplicaciones. Este hecho se hace patente en el siguiente capítulo, en el que se describe con detalle el desarrollo de @note.

Capítulo 3 - Desarrollo de una aplicación RIA para la anotación de textos literarios digitalizados

3.1 Introducción

En esta sección se presenta un caso de estudio de desarrollo de aplicación RIA en los dominios de las Humanidades Digitales y de la Educación, que permite llevar a cabo una evaluación cualitativa sobre el desarrollo de RIAs en casos reales relativos a dichos dominios. Más específicamente, el objetivo de este caso de estudio es explicar la arquitectura y desarrollo de la RIA @note, desarrollada por el grupo ILSA de la Facultad de Informática de la UCM.

Este caso de estudio, por su naturaleza colaborativa y manejo de conceptos humanísticos complejos, así como por la necesidad de ofrecer alta escalabilidad, presenta varios retos, que pondrán en evidencia las ventajas y ayudará a desvelar las posibles desventajas que puedan surgir en la utilización de la propuesta realizada en este trabajo de investigación.

La estructura del capítulo es como sigue. La sección 3.2 describe brevemente la RIA @note, desde el punto de vista de su funcionalidad y uso por parte de los investigadores y estudiantes en Literatura. Seguidamente en la sección 3.3 se muestra el modelo de datos utilizado. La sección 3.4 describe la estructura de la parte de servidor de @note, detallando los servicios de llamadas asíncronas que se producen entre cliente y servidor, así como la interfaz basada en servicios web. La sección 3.5 muestra la estructura de la interfaz de usuario en la parte cliente de la aplicación. Finalmente, la sección 3.6 describe el desarrollo de la aplicación.

3.2 @note: Sistema colaborativo para la anotación de textos digitales.

En este apartado se presenta brevemente el sistema @note desde un punto de vista de usuario (para una descripción más detallada puede consultarse (Gayoso, 2012)).

@note es una aplicación desarrollada por el equipo de investigación ILSA y representa una herramienta capaz de realizar anotaciones semánticas sobre libros digitalizados, particularmente, sobre el corpus de libros antiguos de la UCM digitalizados por Google. Estas anotaciones serán realizadas por un grupo de estudiantes investigadores en letras, organizados en grupos de trabajos para los cuales el tutor habrá asignado un libro (objeto de análisis), el idioma de la interfaz, una taxonomía de conceptos clasificados por categorías y tags, los cuales permiten abstraer la semántica de la anotación realizada por el alumno, y una folksonomía (que al

principio de la actividad estará vacía). El procedimiento comienza con la autenticación del usuario en la aplicación. Una vez autorizada su entrada, el usuario, en caso de ser administrador encontrará el menú de administración (véase Figura 20). Dicho menú ofrece las siguientes funciones:

- *Catalogue*, que permite al administrador crear las taxonomías.
- *Inteface Languages*, que facilita la gestión de idiomas de la interfaz de la aplicación para los alumnos.
- *Export Templates*, que facilita la gestión de creación y edición de plantillas, en las cuales se introducirán anotaciones que serán exportadas a RTF, con el fin de permitir a los alumnos el escribir ensayos de manera *offline*.

Figura 20 Menú de Administración

- *Activity*, que representa la sección donde el usuario podrá crear una actividad reuniendo a un grupo de trabajo para anotar un texto
- *Groups*, *Users*, *Administrator*, que permiten la gestión de grupos usuarios y administradores respectivamente.
- *Get A book*, que permite buscar un texto en bibliotecas digitales (actualmente la biblioteca disponible es Google Books).
- *Upload a Book*, que permite al administrador subir textos personales que no encuentre en la librería digital

- *My Books y My Activities*, que representan la biblioteca con los libros buscados por el administrador y las actividades de las cuales él es el tutor respectivamente.

El usuario estudiante por otro lado, luego de pasar el permiso de autentificación de la aplicación, encontrará el conjunto de actividades asignadas por los diversos tutores (Véase Figura 21).

Figura 21 Actividades asignadas por un tutor

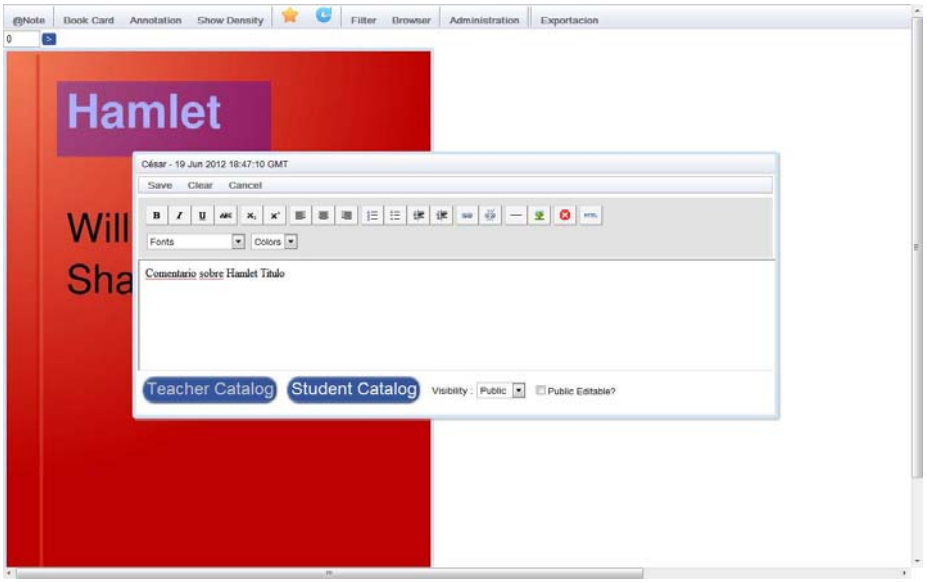

Figura 22 Gestión de Anotaciones

Una vez seleccionada la actividad que se va a realizar, se desplegará la imagen del texto. Es aquí donde el usuario cuenta con todas las herramientas necesarias para anotar el libro. Las anotaciones tendrán anclas que se colocarán haciendo uso de eventos de tipo *onMousePress* y *onMouseRelease*, como se muestra en la Figura 22. Al colocar el ancla se desplegará de manera automática una ventana que permitirá colocar la anotación, así como asignarle *tags* obtenidos de

la Taxonomía o Folksonomía, como se muestra en la Figura 23. El usuario podrá añadir cuantos *tags* considere conveniente; una vez escogidos, pasará a salvar la anotación.

Los *tags* añadidos a las anotaciones permiten a los usuarios filtrar y buscar anotaciones basados en estas etiquetas. Así mismo, se podrá utilizar un filtrado avanzado, en donde se podrá establecer restricciones lógicas (cómo por ejemplo, anotaciones de un usuario "A" que no haya utilizado la etiqueta "T" etc). La sección de exportación permite escoger y clasificar anotaciones para luego transformarlas a RTF.

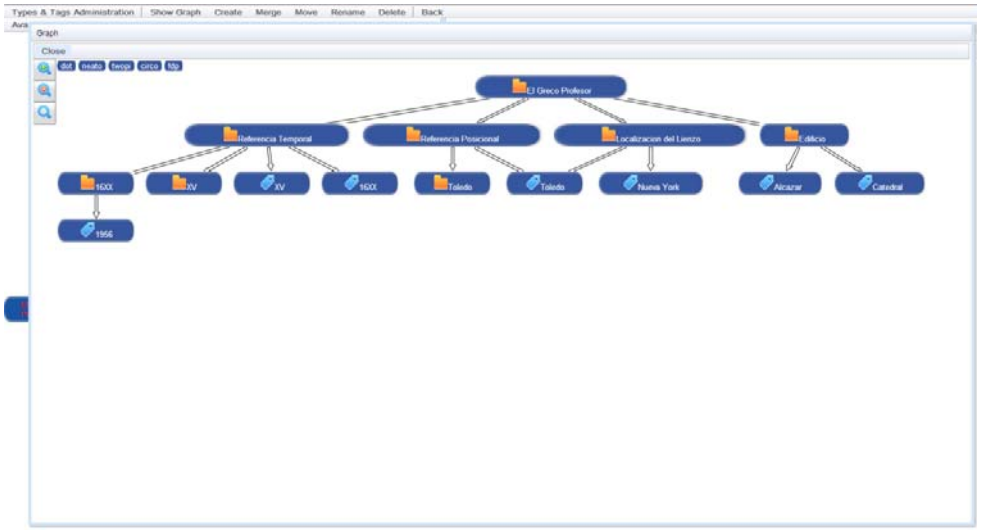

Figura 23 Taxonomía para una actividad

Dentro de la aplicación el usuario podrá ver anotaciones realizadas por otros usuarios activando el botón correspondiente, mostrándose el texto sobre el cual el usuario hubo realizado su anotación, teniendo la posibilidad de responder a esta anotación, creando así hilos o foros de comentarios.

La aplicación también ofrece un servicio RESTful para exportación de anotaciones junto con los *tags* pertenecientes a la taxonomía o folksonomía, así como algunos metadatos, como por ejemplo correo electrónico del usuario anotador.

3.3 El modelo de datos

El sistema de datos de la aplicación @note se basa en un modelo de datos orientado a objetos, solución que se alinea adecuadamente con la tecnología de desarrollo utilizada: GWT en el lado del cliente, y el marco de persistencia JPA en el servidor, el cual aporta, además,

independencia del servidor de aplicaciones concreto utilizado (actualmente la aplicación se aloja en el GAE, aunque se está trabajando ya en su migración a un servidor propio del grupo ILSA).

La Figura 24 muestra el diagrama de clases utilizado en la aplicación. Todas las entidades de este modelo, representan POJOs (*Plain Old Java Objects*), por lo que los métodos utilizados son únicamente *getters* y *setters*, los cuales fueron omitidos del diagrama por simplicidad en el dibujo.

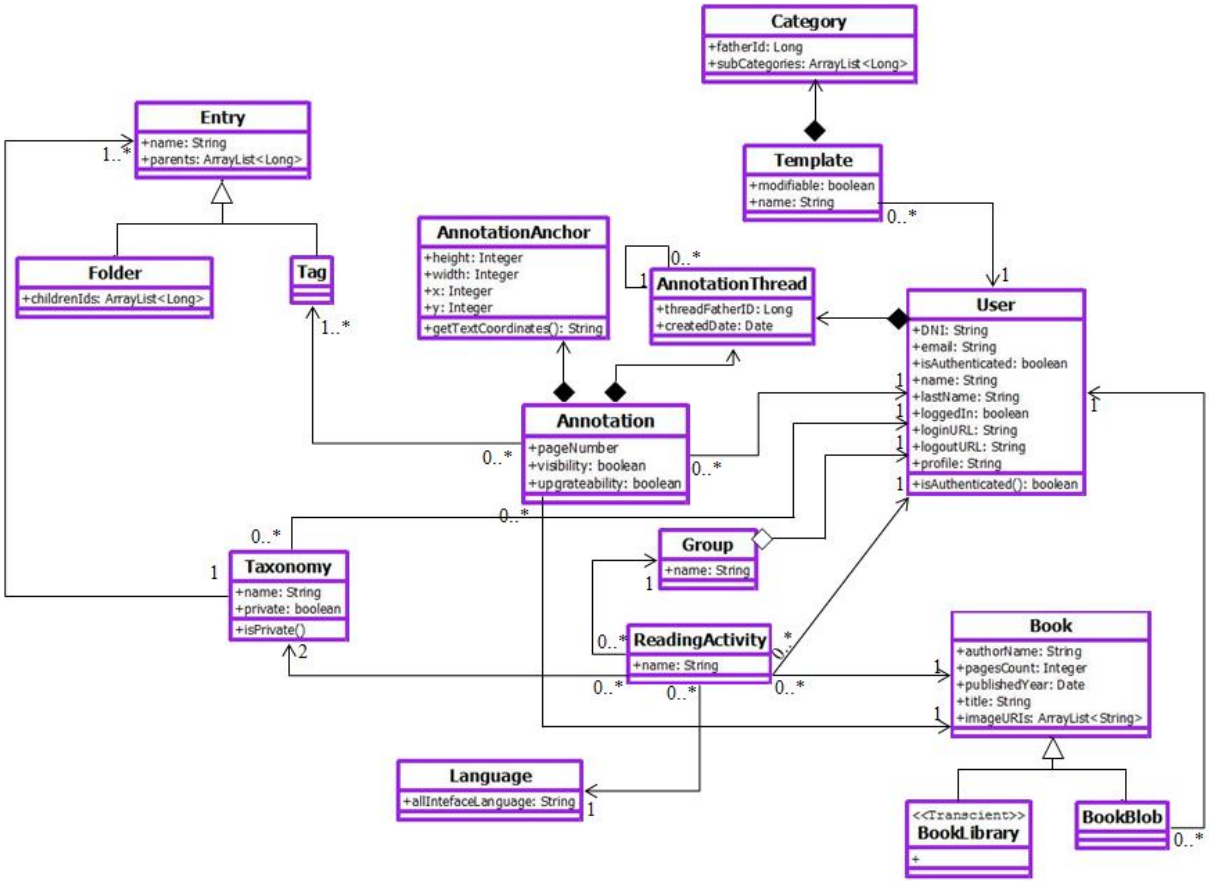

Figura 24 Diagrama de Clases de @note

Las entidades consideradas en este diagrama, representan únicamente a las clases persistentes de la aplicación, que serán todas aquellas clases de naturaleza serializable para la transferencia de los datos entre cliente y servidor. Sin embargo existen otras clases de naturaleza no persistente y serializables. En el modelo sólo se muestran las que se consideran conceptualmente importantes para la aplicación (en concreto, *BookLibrary*), y no las que se utilizaron por facilidades de desarrollo. De esta forma:

- La entidad *User:* describe al usuario que anota. Sus atributos representan los datos básicos de cualquier usuario (nombre, DNI, correo, etc), y su tipo de perfil, esto es, si es profesor o estudiante (dependiendo de este perfil existirán permisos habilitados a ciertas secciones de la aplicación). El usuario puede o no pertenecer a uno o varios grupos de lectura (entidad *Group*). De igual forma el profesor puede crear entidades *Template.*
- La entidad *Group:* está compuesta por un grupo de usuarios destinados a anotar el texto.
- Las entidades *Category* y *Template*, representan clases ajenas a la actividad, y su función es la de servir como plantillas para la clasificación de anotaciones, que serán exportadas a otros formatos, con el fin de hacer análisis textuales de forma off-line, tal y como se explica en la sección 2.2.
- La entidad *Book:* está compuesta por los meta datos básicos de un libro (ISBN, número de páginas, año de publicación, etc). Existen dos formas de este objeto: el *BookLibrary*, el cual no se guardará en la base de datos (volátil), debido a que es directamente recuperado de Google Books mediante su id, y el *BookBlob,* que representa un dato blob (las imágenes y archivos que puedan ocupar más de 2MB se guardan en el servicio de Blobstore, y su entrada se llama, como ya se indicó en el capítulo anterior, blob) del GAE, y es un libro (imágenes) que el profesor sube a la nube. Esta forma de la entidad *Book* sí es persistida en la base de datos.
- La entidad *Annotation*, es sin duda la clase conceptualmente más importante de toda la aplicación. Cuenta con una lista de anclas (*AnnotationAnchor*) con las cuales mantienen una relación fuerte (sólo existen mientras exista *Annotation*), y son las que representan el texto seleccionado en donde se realiza el comentario. Así mismo esta entidad cuenta con una lista de entidades *AnnotationThreads* (de igual forma mediante una relación fuerte), las cuales representan los hilos de comentarios (de profundidad infinita) sobre la anotación. El significado de la anotación se abstrae mediante la asignación de una lista de entidades *Tag*, que aportarán naturaleza semántica a ésta. Dichas listas son proporcionadas por la Taxonomía y Folsksonomía asignadas a la actividad. Por último, y naturalmente, cuenta con la referencia al libro (número de página) sobre el cual se ancla, así como al usuario creador.
- La entidad *ReadingActivity* es conceptualmente importante, porque representa el esqueleto de la aplicación, al ser la que compone las clases necesarias para recrear las anotaciones de textos. Esta entidad se conforma por: *un grupo*, que representa el conjunto de usuarios que

estarán destinados a realizar las anotaciones, *dos catálogos*, uno representando una taxonomía de naturaleza inmodificable y creada por los expertos, y otro representando una folksonomía que, en principio, se asigna vacía y que será generada y desarrollada por los colaboradores del grupo, *un idioma de interfaz* que será asignado por el creador de la actividad, dependiendo del idioma del texto a analizar, y por último, *un texto* (objeto de análisis), perteneciente a la biblioteca del usuario creador de la actividad, bien sea generado de manera personal (*BookBlob*, - el cual es persistente en la base de datos de la aplicación-) , o tomado de la librería digital que lo contenga (este objeto se declara como no persistente debido a que se busca en bases de datos ajenas a la aplicación, pero sí serializable para permitir transmitir esa búsqueda al cliente).

- La entidad *Taxonomy*: representa un grafo de conceptos, destinados a aportar etiquetas que permitan abstraer la semántica de las anotaciones. Esta clase se compone por una o muchas entidades *Entry*. La clase *Entry* es la encargada de generar el grafo, al permitir una herencia de sus clases hijas (*Folder* y *Tag*), la primera conteniendo una lista de identificadores de clase *Entry*, y la segunda, la clase *Tag*, utilizada por los usuarios para clasificar su comentario.
- La entidad *Language:* la cual representa, por último, el idioma de la interfaz de la aplicación. Contiene como atributos cada uno de los textos y títulos estáticos que puede tener la aplicación.

3.4 El Lado del Servidor de @note

La Figura 25 esquematiza la arquitectura utilizada en la implementación del lado del servidor de la aplicación. La arquitectura seguida corresponde a la tradicional arquitectura cliente/servidor. Es de hacer notar que no fue factible implementar arquitecturas más elegantes que permitieran aún más la separación de la lógica de negocios con el modelo de datos, debido a que en los inicios de desarrollo de la aplicación no existía ningún marco que ofreciera la implementación de arquitecturas MVC o MVP para GWT.

A fin de explicar esta arquitectura, se utilizarán los tres tipos de entidades *Anotación*, *Plantilla* y *Actividad* introducidos en la sección anterior. De esta forma, existen 5 gestores de servicios utilizados por el cliente para la comunicación con el servidor.

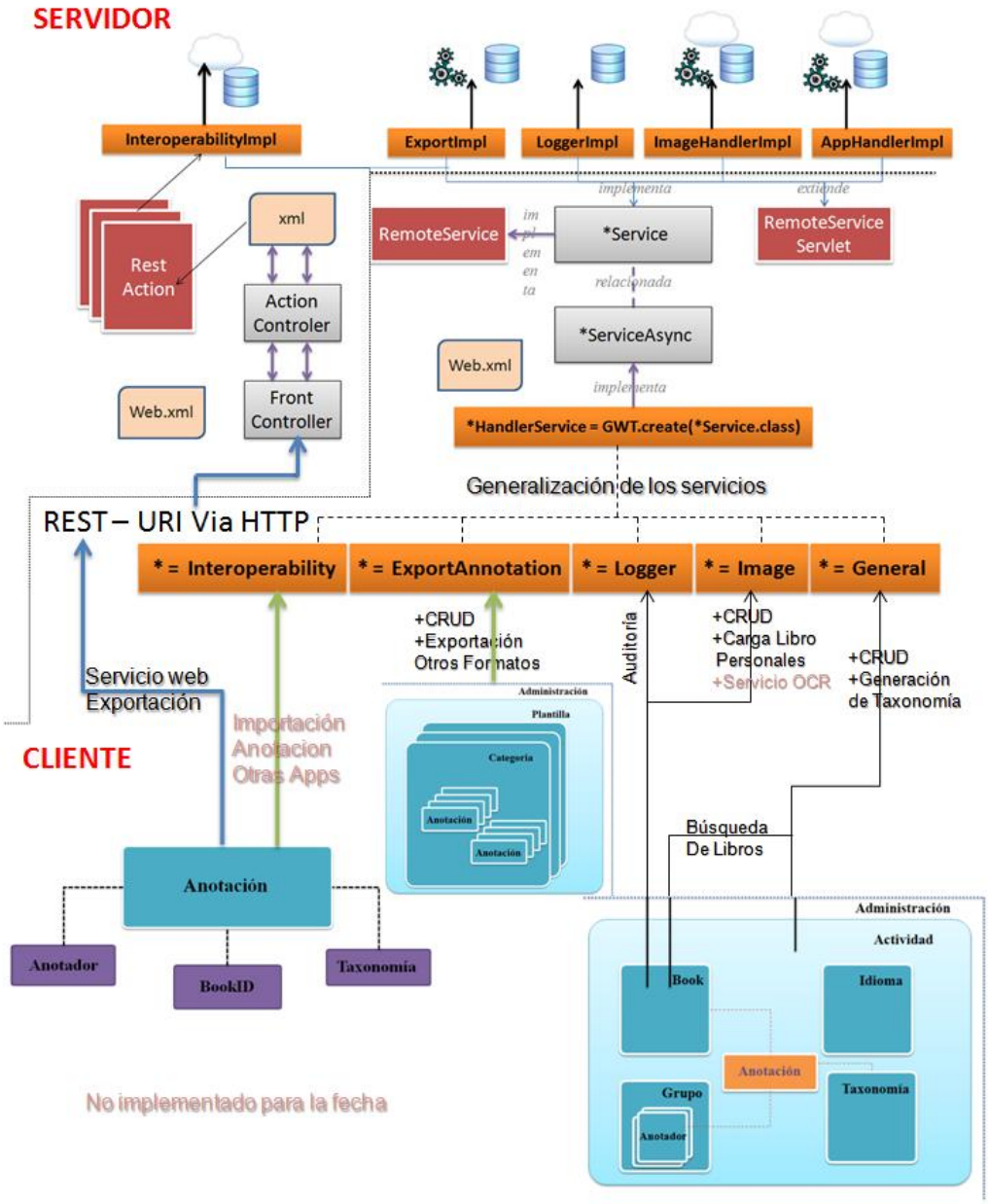

Figura 25 Arquitectura de @Note

Uno de los gestores se encarga de ofrecer un servicio REST que exporta las anotaciones requeridas en formato JSON o XML. Este servicio utiliza la estructura de servicio web RESTful. Los otros cuatro gestores establecen con el servidor llamadas a métodos de forma asíncrona. Estos métodos son los encargados de ejecutar la lógica de negocio, funciones CRUD, y llamadas a otros servicios pertenecientes a otros sistemas. Dependiendo de sus funciones, los gestores se denominan *InteroperabilityHandlerService*, *ExportAnnotationHandlerService*,

LoggerHandlerService, *ImageHandlerService* y *GeneralHandlerService*, y se detallan a continuación:

- El gestor *ExportAnnotationHandlerService* se utiliza para la exportación de plantillas de anotaciones en otros formatos (.doc, pdf, …), y ofrece funciones CRUD sobre estas entidades[20]. Actualmente se soporta la exportación en HTML y en RTF.
- El gestor *LoggerHandlerService* se utiliza como auditor, en secciones sensibles de la aplicación, como por ejemplo eliminar/actualizar usuarios, actividades, taxonomías etc.
- El gestor *ImageHandlerService* se invoca a fin de posibilitar la carga de libros personales (*BookBlobs*) para ser incorporados a la biblioteca del usuario, así como la manipulación mediante funciones CRUD de estos libros. Se tiene, así mismo, previsto que el servicio sea capaz de realizar OCR sobre las imágenes anotadas.
- El gestor *GeneralHandlerService* es una fachada a la lógica de negocio más amplia de la aplicación, pues aquí se ejecuta los métodos más comunes, y funciones CRUD de las entidades faltantes: generación de las taxonomías y carga los libros pertenecientes a librerías externas, creación de anotaciones, etc.
- El gestor *InteroperabilityHandlerService* se utiliza como adaptador de recuperación o importación de anotaciones de otros sistemas web. En la fecha de escritura de este trabajo, aún no se ha abordado la implementación de este servicio, estando prevista su implementación en la siguiente fase del proyecto.

La comunicación asíncrona establecida entre el cliente y estos gestores se realiza mediante la estructura de comunicación RPC de GWT detallada en la Figura 19.

La entidad *Anotación*, junto con el anotador, el libro anotado y la taxonomía puede ser exportada mediante un servicio REST. Este servicio se implementa mediante una arquitectura simple REST, la cual cuenta con un servlet (*front controller*) encargado de gestionar la petición HTTP. Dependiendo de la petición (GET, POST), la implementación del servlet (*action controller*) gestionará la exportación en formato JSON.

Este mismo objeto Anotación podría ser importado desde otra aplicación, mediante la utilización de un adaptador que procese la solicitud de petición del objeto Anotación externo, y

[20] Es importante recordar, que los métodos CRUD son realizados mediante la plataforma de persistencia Datanucleus, a través del API JPA.

lo convierta al objeto Anotación de @note. Para la fecha de escritura de este trabajo, dicho adaptador no ha sido desarrollado.

Las Plantillas, destinadas a la exportación de anotaciones clasificadas a formato RDF, se realizan mediante el gestor *ExportAnnotationHandlerService*. Este gestor realiza una llamada a un servicio externo que lleva a cabo la transformación y devuelve el resultado. Esto es debido a que el GAE no soporta la escritura de archivos, lo que imposibilita la creación de un archivo y escribir en éste las anotaciones. Sin embargo, el servicio externo es controlado por los desarrolladores de @note, al incluir un código php que sobrescribe la petición *al vuelo* y la devuelve en formato RDF.

Por último, la actividad de lectura, conformada por, Book, Group, Taxonomies e Idioma, es manipulada por tres gestores. Las acciones de Delete sobre un libro, o una taxonomía son auditadas mediante el *LoggerHandlerService*, el cual genera un log sobre estas acciones directamente en el servidor. La generación de la taxonomía y los métodos básicos de CRUD de la Actividad son gestionados por el *GeneralHandlerService*.

3.5 La UI de @note

Esta sección se centra, por último, en el diseño de la UI de la aplicación. Primeramente se describe la notación utilizada en la presentación del diseño (notación inspirada por los diversos enfoques al diseño de RIAs descritos en el capítulo anterior). Seguidamente se describe el diseño de la UI en sí.

3.5.1 Notación

Para la explicación de la navegación de la aplicación se utiliza la notación descrita en la Figura 26. En dicha notación:

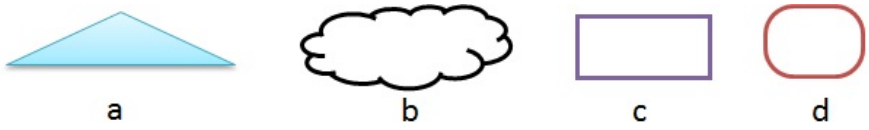

Figura 26 Notación utilizada

- (a) representa una sección nueva de la aplicación, es decir, como si se tratara de una nueva página en las aplicaciones web tradicionales.
- (b) equivale a una llamada al servidor, a través de los gestores explicados en la sección 3.4
- (c) representa el evento *click*, que produce una acción de cambio de página, o refrescamiento.

- (d) Representa la generación de un formulario con datos, esto es, cargar el formulario con datos existentes (para los casos en donde se esté editando un objeto, o vacío para objetos nuevos)

3.5.2 El diseño de la UI

En GWT aquellas clases que implementen la interfaz *EntryPoint* son aquellas que implementan el método de JavaScript *onLoad*, que es el método invocado en primer lugar tras la carga de la página. Por lo tanto estas clases definen las secciones de navegación de la aplicación. De esta forma:

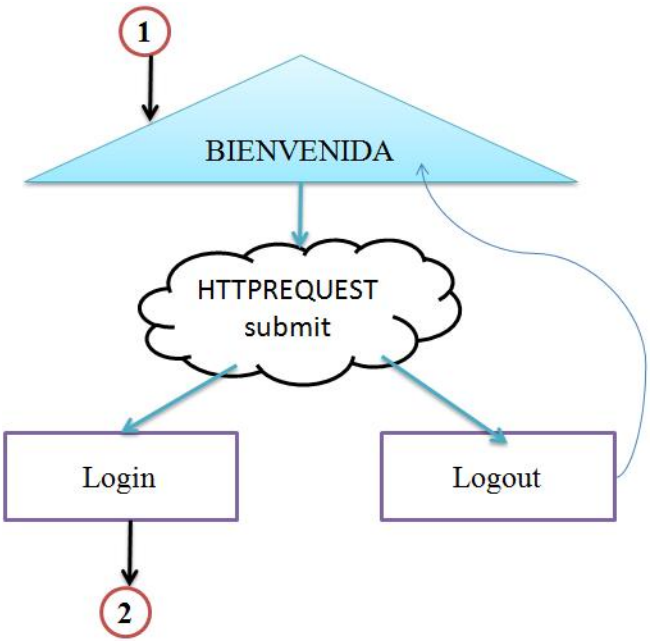

Figura 27 Sección de Bienvenida

- Cuando la aplicación arranca, la primera sección en ejecutarse es la de bienvenida (véase la Figura 27). Aquí, de manera transparente, se generan los URLs de *login* o *logout* utilizados para la autentificación de una cuenta en Google.
- Una vez superada la autentificación con Google, el método devuelve la referencia a una clase que sirve como controlador de todas las clases que implementen *EntryPoint*. Esta clase controladora representa la sección de administración de la aplicación, y es particularmente importante a nivel de implementación, porque ayuda a agrupar y separar las clases del cliente, y las clases compartidas entre cliente y servidor (Figura 28).

- Las secciones de Taxonomías, Actividades y Grupos comparten la misma estructura de navegación. Por simplicidad se mostrará sólo una a modo de ejemplo (véase la Figura 29). Esta sección en su carga, realiza una llamada asíncrona para listar las actividades existentes en el sistema, cada una de las cuales pueden ser editadas o eliminadas. De igual forma permite la creación de una nueva actividad mediante un formulario. Este formulario necesita datos, que el usuario pueda escoger con el fin de crear su actividad. Para esto se realiza otra llamada al servicio (en este caso particular cargará, Grupos, Libros, Idiomas y Taxonomías, los cuales son los necesarios para la creación de una actividad). Como se ha indicado, esta misma estructura de navegación es seguida en la gestión de Taxonomías y Grupos.

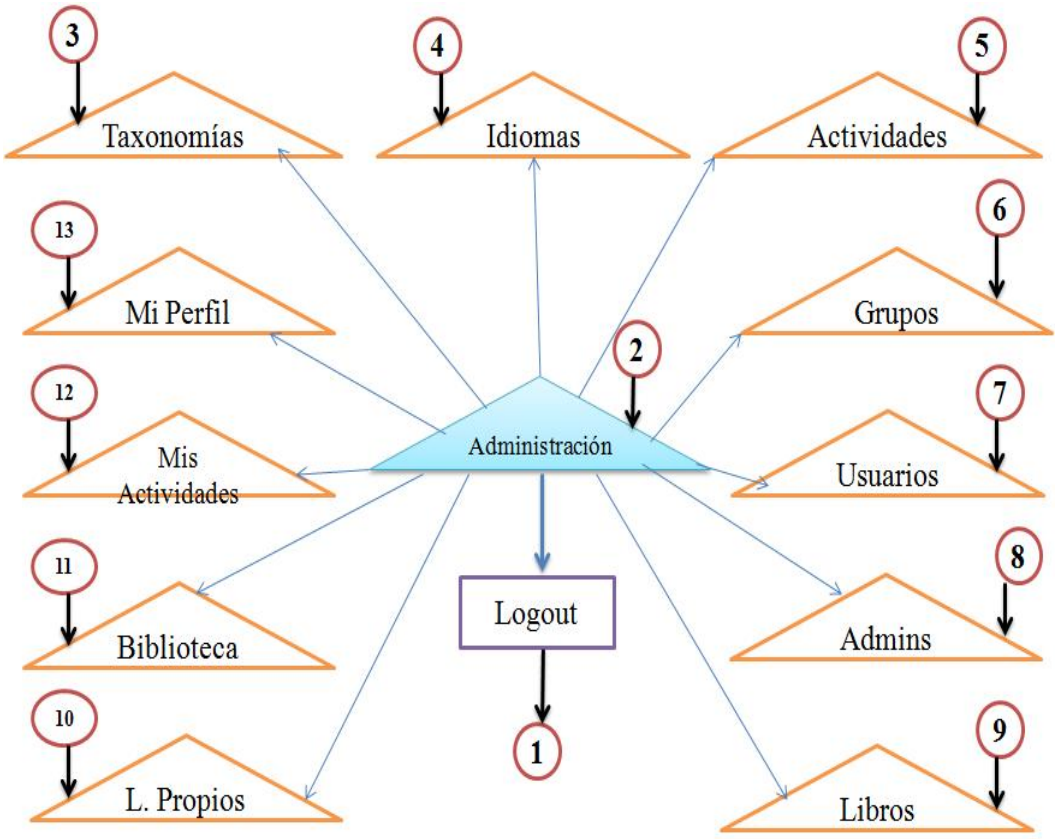

Figura 28 Sección de Administración

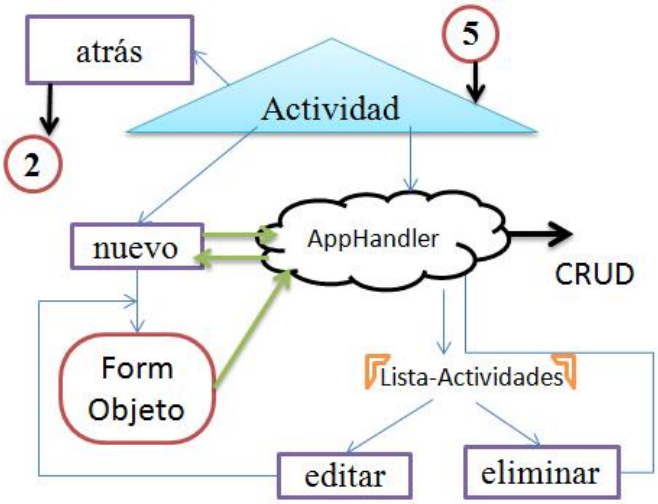

Figura 29 Sección Taxonomía, Actividad, Grupo

- De igual forma las secciones de Idiomas, Usuarios, Administradores y Libros personales (Figura 30) comparten la misma estructura de navegación. Además esta estructura de navegación es prácticamente igual a la utilizada en las Actividades (vista anteriormente) con la diferencia de que no existe ninguna llamada para cargar los objetos necesarios en la creación de estas entidades.

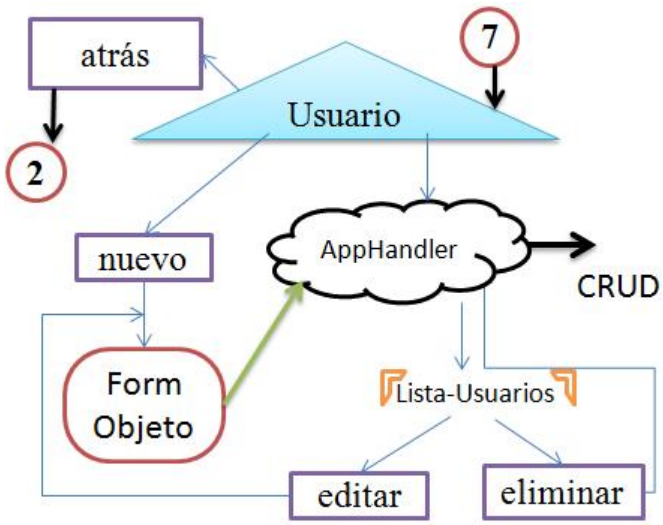

Figura 30 Sección de Idiomas, Usuarios, Administradores, Libros Personales

- La Figura 31 muestra la sección de administración de libros. Cuando se carga esta sección se llama a la lista de libros con los que cuenta el usuario en el sistema, permitiéndole la opción de eliminar alguno.

- Para la carga de un nuevo libro, se ejecuta otra clase *EntryPoint* que se encargará de la búsqueda en librerías externas. Como resultado de la búsqueda se genera una lista de libros para los cuales el usuario escogerá uno, el cual podrá visualizar. Si es el libro deseado se guardará a su biblioteca personal; de lo contrario, el usuario podrá volver a la búsqueda.

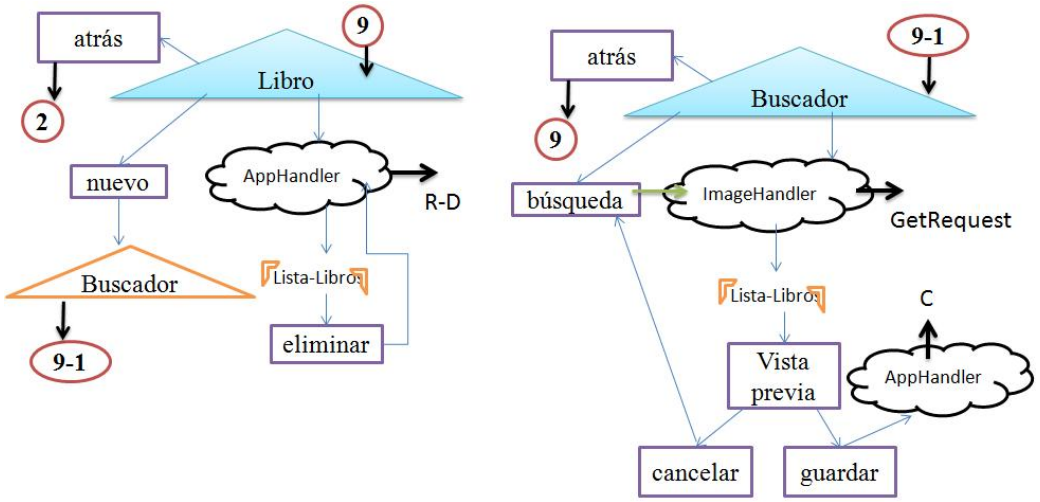

Figura 31 Sección Administración de Libro

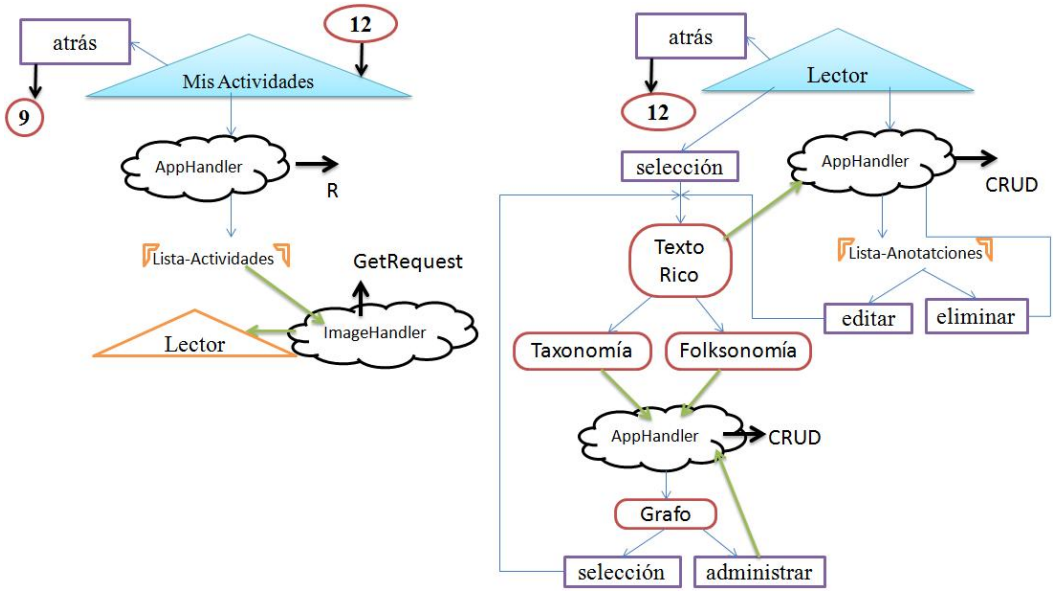

Figura 32 Sección Mis Actividades

- La gestión de Mis Actividades representa la única sección permitida por los usuarios comunes (no administradores), y se muestra en la Figura 32. Aquí el sistema carga las

actividades asignadas por el administrador al usuario, el cual podrá elegir con cual desea trabajar, de manera transparente se llama al servicio de carga de libro y arranca la clase *EntryPoint* conceptualmente más importante: el Lector. Aquí es donde se permite la creación de anotaciones, mediante un anclaje (selección), un comentario (texto Rico) y tags (seleccionados en la Taxonomía o Folksonomía). De igual forma al igual que en otras secciones la eliminación o la edición de anotaciones es intuitiva.

3.6 Desarrollo de @note

El desarrollo de @note surge de la propuesta creada por los grupos de investigación LEETHI e ILSA al programa organizado por Google: *Digital Humanities Award Program*, del cual dicho proyecto fue uno de los premiados, ofreciendo así, la financiación para su desarrollo. El modelo de proceso seguido fue RAD (Rapid Application Development), debido a los tiempos de entrega relativamente cortos para el desarrollo de secciones que había que entregar a Google con el fin de justificar la financiación.

Las siguientes secciones describen las etapas de desarrollo de la aplicación, conceptualmente pensadas por el grupo de investigación de humanistas e informáticos, y desarrollada por ILSA, así como los problemas encontrados durante el desarrollo y las lecciones aprendidas.

3.6.1 Desarrollo Colaborativo en un Equipo Multidisciplinar

El grupo de investigación LEETHI, orientado al estudio del paso de la lectura tradicional a otras formas de lecturas digitalizadas, describe sus inquietudes sobre la aplicación al jefe de proyecto en ILSA en sesiones de reuniones semanales. La Tabla 1 muestra los tiempo y etapas de desarrollo de @note. El tiempo está dado en meses y resulta una aproximación en donde cada etapa cuenta con sesiones de pruebas.

Al ser los equipos involucrados en el desarrollo de áreas de investigación tan diferentes (filólogos e informáticos), surgieron algunos problemas estructurales durante el desarrollo de la aplicación. En la Tabla 1 se reflejan todos los cambios que hubo que realizar sobre la aplicación (ya sea en el modelo de datos o en términos de usabilidad), y que los expertos filólogos de LEETHI consideraba necesarios, en una sola etapa, “Restructuración de la Aplicación”, aunque estos cambios fueron realizados de manera periódica entre etapas

Tiempo	Etapa	Detalle
1	Documentación y Configuración	Lectura de la documentación y especificación de GWT y GAE, configuración y puesta en marcha de estas tecnologías
1	Modelado de datos	Se modelan los datos Annotation, Text Selector, y se prueban desde el punto de vista del cliente (GWT), sin persistirlos.
1	Modelado de la arquitectura	Se desarrolla la comunicación entre cliente y servidor, mediante un gestor de comunicación (específicamente GeneralHandlerService)
1	CRUD del modelo de datos	Las entidades Annotation y AnnotationAnchor se hacen persistentes.
1.5	Introducción a las Taxonomías	Se modela la entidad Taxonomy y se desarrolla para su persistencia y comunicación con el cliente (CRUD)
1	Recuperación de Libros en Google	Se establece la comunicación para la entrada de libros de Google Books a la aplicación
1	Administración de usuarios	Se desarrollan los perfiles de usuarios, privilegios y auditorías de secciones en la aplicación
1.5	Actividades de Lectura	Se modela y se desarrolla la sección de actividades de lecturas (Grupos, Idiomas, Taxonomías, libro de lectura)
1.5	Filtro de anotaciones	Generación de navegación y filtro para la búsqueda de anotaciones
0.5	Hilos de comentario	Se crea los foros críticos de lectura, comentarios que realizan usuarios sobre anotaciones realizadas por otros usuarios
1	Carga de libros propios	Se desarrolla la posibilidad para el administrador de cargar sus propios textos
2.5	Restructuración de la aplicación	Cambios en el modelo y en la usabilidad de la aplicación, mejoras en la interfaz de usuario
1.5	Pruebas generales y depuración	Se prueba la aplicación general y se depuran errores encontrados, se despliega a producción.
1.5	Pruebas por el cliente	Seguimiento a pruebas realizadas por alumnos de humanidades, depuración de errores.

Tabla 1 Etapas de desarrollo

Su consecución aproximadamente llega a dos meses y medio, como refleja la tabla. De igual forma es importante aclarar que @note cuenta con servicios web para la exportación de anotaciones. Los tiempos involucrados en dicho desarrollo, al no pertenecer estrictamente al aspecto RIA, no se consideran relevantes para este trabajo.

3.6.2 Problemas encontrados y limitaciones de GWT y GAE

A medida que se iba desarrollando la aplicación, como es normal, iban surgiendo las limitaciones de las tecnologías utilizadas, y era necesario buscar *workarounds* que evitaran estas limitaciones. Por ejemplo, a nivel del GAE se descubrieron las siguientes limitaciones:

- La naturaleza de GAE no permite relaciones *many-to-many*. Esto hace que en el modelo de datos, clases que requieran este tipo de relación pierdan la potencia de ser estructuradas como verdaderas clases Java. De esta forma, en lugar de tener como atributos listas de objetos de otras clases, se tienen listas de identificadores de objetos a otras clases, lo que hace que la referencia entre éstas sea mediante identificadores y no mediante objetos. Esta limitación hace que el desarrollo sea más lento y menos robusto, y dificulta su mantenimiento, debido a que para los métodos CRUD de los objetos que posean este tipo de relación será necesario cargar de la base de datos, cuando sea necesario, la listas de objetos referenciados mediante sus identificadores.
- Otra limitación encontrada fue la imposibilidad de gestionar directamente, en el GAE, la base de datos. Las modificaciones que se realicen a la base de datos deben ser realizadas programáticamente, con lo cual si es necesario cambiar algún dato *en frío*, habrá que escribir algún método que lo realice y seguidamente desplegar la aplicación correspondiente. Al ver que este tipo de gestión es tedioso y poco útil, surge la necesidad de crear un gestor simple de base de datos que incluya cada una de las entidades involucradas en la aplicación.
- GAE no soporta la reescritura de archivos, así que cuando se desarrolló la sección de exportación de anotaciones en formato RDF, se afrontó un problema, ya que lo común es crear un archivo programáticamente y escribir sobre él. El w*orkaround* utilizado fue la utilización de otro servidor php (se utilizó php, porque existen servidores gratuitos para alojar servicios simples), que realizara esta transformación y la devolviera al GAE.
- Por último, en las pruebas realizadas en producción con los alumnos de humanidades, nos dimos cuenta que la aplicación dejaba de funcionar pasado un tiempo, y esto era por la limitación en transacciones del GAE, en su versión gratuita. Para esta limitación no existe *workaround*: simplemente se ajustó la tasa a una versión *de pago*, que permitiera más transacciones.

Por su parte, desde el punto de vista de GWT se encontró una gran limitación: la inserción de Javascript nativo en GWT es ciertamente complejo y de comportamiento variable

(de hecho, la propia documentación de GWT lo especifica), lo que hace que su implementación no sea eficaz y se trate de evitar en todo momento. Un caso particular fue la generación gráfica del grafo para las taxonomías / folksonomías: utilizar librerías de JQuary y gestiones de Javascript nativo era realmente frustrante, lo que nos llevó a abortar esta aproximación y utilizar un servicio externo que entregara el *layout* del grafo, para así luego pintar en los nodos los botones correspondientes.

3.6.3 Lecciones aprendidas

El disponer ya de una primera versión estable de la aplicación, la experiencia previa de programación de aplicaciones, junto con las limitaciones de las tecnologías utilizadas, nos han ayudado a comprender mejores formas de construcción de RIAs. Por ejemplo la decisión de desacoplar el sistema, al tener una capa de persistencia independiente de las gestiones que realiza el cliente fue una excelente idea, al permitir realizar migraciones (en caso que se requiera) a otros servidores de una manera más fácil.

El desarrollo de un sistema complejo de llamadas asíncronas mediante la utilización de *servlets* fue sin duda una de las mejores lecciones aprendidas durante el desarrollo de esta aplicación, debido a que nos ayudó a separar la lógica de negocios del sistema, dependiendo de las funciones que se realicen (los gestores explicados en la sección anterior).

Sin embargo por la novedad de GWT para el desarrollo de este trabajo, no existían marcos que permitiesen arquitecturas formales (MVC, MVP, etc) para la aplicación, por lo cual utilizamos la arquitectura tradicional cliente/servidor; esto hace que el sistema sea menos flexible, de depuración más lenta y mantenimiento mucho más complejo. Para la fecha ya han aparecido marcos para arquitecturas MVC, que aunque no son desarrolladas por Google, Google en sí si recomienda su uso.

Somos conscientes de que debido, en gran parte, a las limitaciones de las tecnologías involucradas, ciertas partes necesitan ser refactorizadas: por ejemplo, la generación del *layout* del grafo no puede pertenecer a un servicio externo experimental como el que estamos usando actualmente, debido a que se arriesga a que el servicio pueda cambiar, y consecuentemente la aplicación se vea afectada; entonces resulta necesario que el *layout* sea controlado por la lógica de negocios de la aplicación. Como consecuencia podemos afirmar que una RIA debe evitar en

lo posible, al menos a nivel visual y de respuesta, la utilización de servicios externos: mientras más auto contenida sea, más similar a una aplicación de escritorio será.

3.7 A Modo de Conclusión

Este capítulo se ha dedicado a la descripción de @note, una RIA para la anotación de textos digitalizados, en donde de manera detallada se explica el modelo de los datos utilizados y la arquitectura empleada, destacando que por cuestiones de novedad de GWT durante la primera fase de desarrollo no existía un marco formal para tener una arquitectura más flexible a la utilizada (la tradicional arquitectura cliente/servidor), lo que conlleva tener problemas principalmente de mantenimiento. Sin embargo el sistema complejo de gestores utilizados para las llamadas a métodos asíncronos mediante *servlets* separados, reduce en gran manera el acoplamiento de esta arquitectura, permitiendo que el mantenimiento se lleve a cabo de manera más fácil, al dividir la lógica de negocio en 5 gestores (pudiéndose ver como controladores en arquitecturas MVC). Así mismo, resulta interesante observar el modelo de proceso llevado a cabo por el equipo multidisciplinar que participa en el desarrollo de @note (LEETHI, ILSA), evidenciando retrasos de tiempo por refactorizaciones conceptuales sobre la aplicación.

Del mismo modo se describe las buenas prácticas utilizadas para el desarrollo. La más destacada resulta ser el desacoplamiento que existe entre la persistencia de los datos y la aplicación. De hecho, tener una capa de persistencia independiente de las gestiones de la aplicación resulta eficaz en el caso que se decida realizar una migración, así como en el mantenimiento de la misma.

De igual forma es evidente que existen limitaciones en las tecnologías. Aquí se describió las pertenecientes al GAE y a GWT, y las soluciones utilizadas para esquivar dichas limitaciones, algunas resultando ser bastante delicadas, como la de contar con servicios externos para la generación del *layout* del grafo, y la persistencia de datos de forma no transaccional en relaciones *many-to-many*.

Capítulo 4 - Estadísticas de uso de @note

4.1 Introducción

Una de las principales ventajas del GAE es que el almacén de datos contiene estadísticas detalladas sobre los datos almacenados de una aplicación. Esto permite extraer fácilmente conclusiones sobre el uso de la misma. En este capítulo se analizan estas estadísticas para @note durante su uso en diferentes experiencias con alumnos de distintas facultades de Humanidades de la UCM.

La sección 4.2 describe el método de obtención de datos. Por su parte, la sección 4.3 describe el uso de las principales entidades involucradas en la aplicación.

4.2 Método de Obtención de Datos

El GAE permite obtener información relativa a, por ejemplo, la cantidad de entidades de un determinado tipo o el espacio que utilizan los valores de propiedad de un tipo en concreto. Estas facilidades son especialmente relevantes de cara a analizar el uso de una RIA. A estos valores se accede programáticamente al realizar una consulta de entidades específicas mediante el API del almacén de datos. Las entidades con estadísticas incluyen las propiedades siguientes:

- *Count*: el número de elementos que determina la estadística.
- *Bytes:* el tamaño total de los elementos de esta estadística
- *Timestamp:* la última hora en que se ha modificado la estadísitica

Estas estadísticas pueden hacer referencia a los tipos de datos "`Blob, Boolean, Date/Time, Float, String, Key, Integer, Text, NULL`".

Con la ayuda de la API introducida en este capítulo se logró determinar la proporción de los datos de la aplicación en el servidor para las distintas entidades manejadas en @note. En particular, se recogieron datos para las entidades que se consideran más importantes y de mayor peso: Annotation, BookBlob, Tag, Activity, Anchor y User.

4.3 Uso de las Entidades en @note

Esta sección describe el uso, en términos de tamaño que ocupan en memoria en el *Datastore* de GAE, cada una de las entidades involucradas en la aplicación. Se muestran, por

cada entidad, una tabla que incluye el número de objetos almacenados y el tamaño que ocupan en el almacén de datos. De igual forma la tabla incluye un gráfico que refleja el porcentaje que ocupa cada uno de los atributos de la clase, separados por tipo de dato, así como la información adicional necesaria para persistir los objetos correspondientes.

4.3.1 La entidad Annotation

Esta clase representa en la aplicación, la que ocupa mayor cantidad de espacio en memoria en el almacén de datos de Google, y es la segunda con mayor número de objetos generados (la primera la tiene el ancla de la anotación -Anchor-; esto es debido a que una anotación puede tener uno o más anclas). La Tabla 2 muestra el gráfico del espacio ocupado por los datos de cada uno de los tipos de datos que conforman los atributos de la entidad:

- Los datos de tipo *Integer*, con 2%, están representados por las propiedades: 'número de página', 'identificador de la actividad de lectura', 'identificador del usuario', y la lista de 'identificadores de tags', que sin duda es la que aporta más tamaño por el hecho de ser una lista de elementos.
- Los datos de tipo *String***,** con 1%, lo conforman los atributos: 'identificador del libro' y 'el nombre del usuario' (este atributo se creó por agilizar la búsqueda de una anotación, sin tener la necesidad de recuperar el objeto *User*).

	Entidades
Tamaño total	**979 KBytes**
número	**547**
Promedio	**2 KBytes**

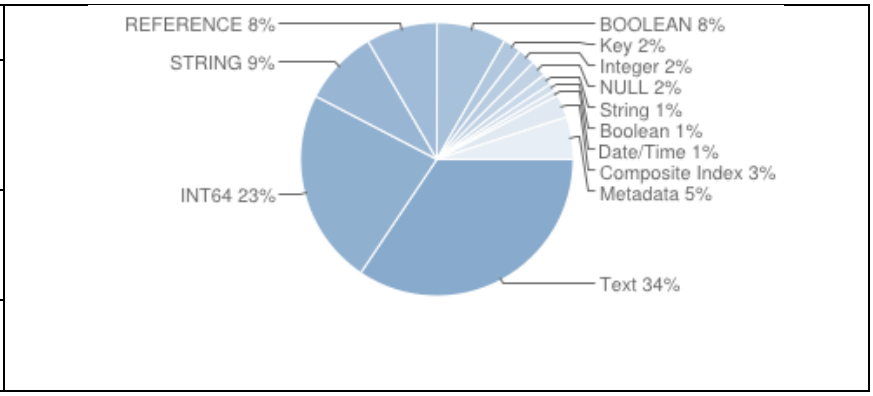

Tabla 2 Estadística de datos de la Entidad Annotation

- Los datos de tipo *Boolean*, con 1%, lo forman las propiedades de 'visibilidad' y 'editabilidad' de la anotación, que son las encargadas de decidir si una anotación es pública o privada, y si es editable o no editable.
- Los datos de tipo *Date* con 1%, se refieren a la propiedad de 'fecha' de creación de la anotación.

- Por su parte, los datos de tipo *Text* pertenecen a un tipo reservado de GAE, que permite almacenar cadenas de caracteres de más de 42 elementos (en GAE no se puede almacenar *Strings* de más de 42 caracteres). Estos datos son los que ocupan mayor espacio de la entidad, 34%, lo cual es lógico, debido a que representan los 'comentarios' de las anotaciones.
- Los datos de tipo *REFERENCE*, con un 8%, se atribuyen a la referencia que hace la anotación a la lista de objetos *Anchor*, donde se anclan dichas anotaciones[21].
- Por último, los datos de tipo NULL, con un 2%, se corresponden con atributos cuyos valores se han omitido durante el proceso de creación de las anotaciones.

Obsérvese que, aparte de estos datos, existen también entradas adicionales orientadas a facilitar el proceso de persistencia y almacenamiento. En particular:

- Entradas para datos de tipo BOOLEAN, STRING e INT64. Estos datos se corresponden con índices añadidos por el GAE para agilizar las búsquedas.
- Entrada para *Composite Index*. Esta entrada, con un 3%, representa los índices de búsquedas por los criterios 'identificador de libro' y 'número de página'. En este caso, dicho índice se añadió programáticamente, ya que involucra búsquedas que afectan a más de un atributo.
- Entrada para Metadata, que se refiere a datos que describen a la entidad (clave -Identificador-, tipo, y propiedades)
- Entrada para Key que se refiere a la presencia del tipo Key (una clase reservada del GAE para asignarle un identificador único a cada objeto) como atributo de la clase.

4.3.2 La entidad BookBlob

La clase Bookblob sólo referencia a las propiedades de los libros subidos por los profesores (más no los libros en sí, estos se guardan en otro servicio de Google -El Blobstore-). Su tamaño total en la base de datos ocupa 50 KB (véase la Tabla 3). Entre sus datos se tienen:

- Datos de tipo *Integer*, que son lo suficientemente pequeños como para haberse redondeado a un 0%. Se refieren al atributo 'identificador de usuario': quién ha subido el libro.

[21] Obsérvese que, en otros casos, las referencias se realizan utilizando listas de identificadores, a fin de representar relaciones *many-to-many*, no soportadas por el GAE.

- Datos de tipo *String***,** con 22%, que se deben a los atributos: ‘autor, número de páginas, año de publicación, título del libro, identificadores de las imágenes’.

	Entidades
Tamaño total	**25 KBytes**
número	**25**
Promedio	**1KByte**

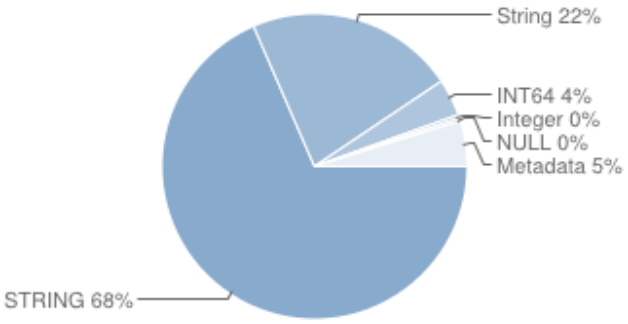

Tabla 3 Estadística de datos de la Entidad BookBlob

El resto de entradas se refieren a atributos no completados, así como a información relativa a la persistencia y almacenamiento (índices, etc.) añadida por el GAE.

Obsérvese que, según la tabla, el número de libros subidos por los profesores ha sido 25. Sin embargo analizando internamente la base de datos se nota que ha habido libros repetidos. Esto se debe quizás a la inexperiencia del usuario al principio de la introducción de la sección de la carga de libros. Entre los libros repetidos por los mismos usuarios se han encontrado 8, con lo cual quedaría un total de 17 libros. Este número de libros subidos, aunque a primera vista parezca pequeño, para un grupo de aproximadamente 5 profesores, implica que cada profesor ha subido aproximadamente 3 libros cada uno, lo que evidencia la importancia que le dan los profesores a esta funcionalidad.

4.3.3 La entidad Tag

Esta entidad cuenta con 394 elementos, distribuidos en varias taxonomías, y representan estructuras que describen el grafo de conceptos. Entre sus atributos se encuentra: ‘la lista de anotaciones que contienen el Tag’ (utilizado para la búsqueda y filtro de anotaciones), ‘el identificador de la taxonomía al cual pertenece’ y ‘el conjunto de padres de los cuales proviene en su taxonomía’. Estos tres atributos conforman el 7% del tipo de dato *Integer*, como se observa en la figura. De igual forma el atributo “nombre” de la entidad ocupa el 4% del tipo *String*. El resto de las entradas se corresponde con información no completada (nula), o añadida por el GAE. El tamaño total de la entidad es de 101 KBytes, resultando ser la tercera entidad que ocupa

mayor tamaño en el almacén de datos, en gran medida debido a la cantidad de elementos que contiene.

	Entidades
Tamaño total	**101 KBytes**
número	**394**
Promedio	**262 Bytes**

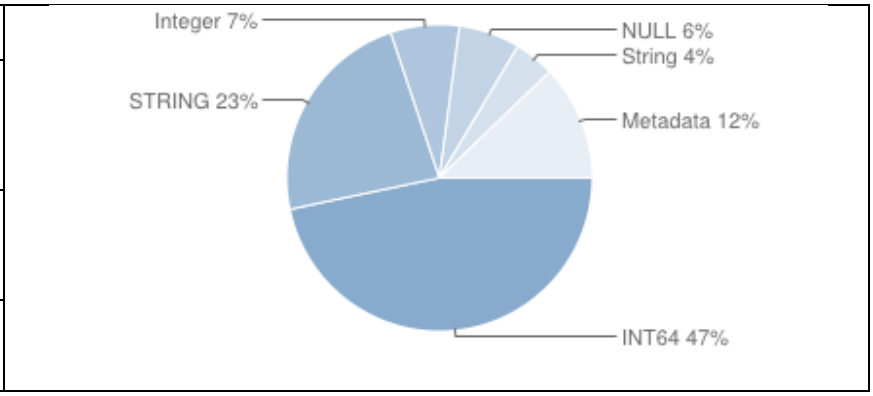

Tabla 4 Estadística de datos de la Entidad Tag

4.3.4 La entidad Activity

Esta clase cuenta con 28 elementos distribuidos entre 6 profesores, lo que equivale en promedio a 4,6 actividades por profesor. Este resultado, según las pruebas realizadas en producción, no coincide, y lo que sucede es que existen actividades creadas por parte de los profesores, inconclusas (falta completar elementos de la actividad) o no utilizadas pero si generadas (actividades que no tienen ningún tipo de anotación, evidencia de su desuso). El número de elementos encontrados que quedaron inconclusos o no utilizados resulta ser 13, con lo cual quedan elementos efectivos 15 a un promedio de 2,5 actividades por profesor.

	Entidades
Tamaño total	**9 KBytes**
número	**28**
Promedio	**323 Bytes**

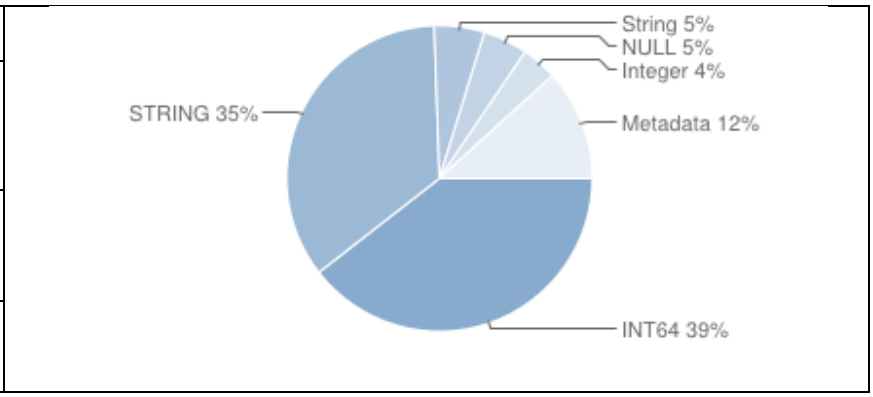

Tabla 5 Estadística de datos de la Entidad Activity

Esta entidad destina a sus propiedades de tipo *String* un 5 %, conformada por los atributos ‘identificador del libro’, ‘idioma a ser utilizado en la interfaz’ y el ‘nombre de la actividad’. El tipo *Integer* con 4%, agrupa a las propiedades ‘identificador de la taxonomía’, ‘identificador de la folksonomía’, ‘identificador del grupo’ y ‘el identificador del profesor’. El resto es

información nula o añadida por el GAE. El tamaño total es de 9 KBytes, lo que resulta pequeño comparado con las entidades vistas hasta el momento. Esto es debido al corto número de elementos con que cuenta la clase.

4.3.5 La entidad Anchor

El ancla de la anotación (*Anchor*) es la segunda clase que ocupa mayor tamaño en el almacén de datos (122 KB), y esto es debido a la cantidad de elementos que tiene: 671. Esto nos permite deducir que existe en promedio 1.22 anclas por cada anotación realizada. Este resultado, casi 1 a 1 entre anotación y ancla, puede deberse a que el desarrollo de tener la capacidad de múltiples anclas en la aplicación se realizó un poco después de las primeras pruebas, y los profesores no eran todavía conscientes de esta funcionalidad, pues estaban por el momento acostumbrados a establecer por cada anotación una única ancla.

	Entidades
Tamaño total	**122 KBytes**
número	**671**
Promedio	**185 Bytes**

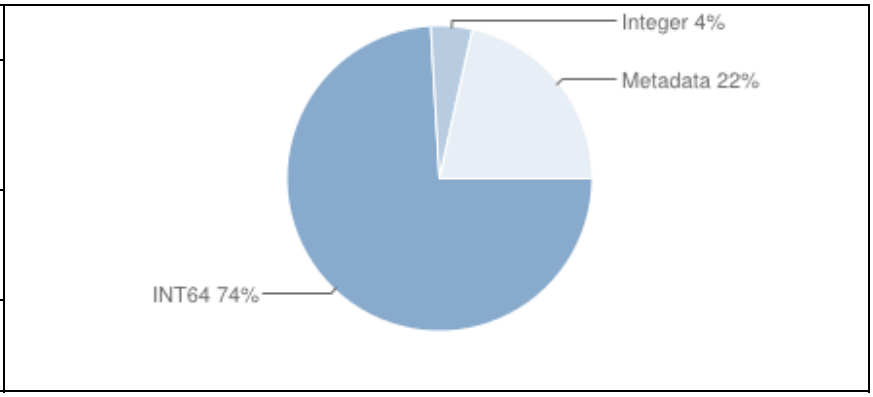

Tabla 6 Estadística de datos de la Entidad Anchor

Como es de esperar, las anclas sólo presentan propiedades numéricas, referidas a la descripción del rectángulo que sirve de selector sobre el fragmento de texto que será anotado. Estos atributos son: las posiciones del punto inicial del rectángulo '*x* e *y*' y las longitudes de sus lados '*alto* y *ancho*'

4.3.6 La entidad User

De igual forma, en la entidad User se muestran 258 objetos creados, a pesar de partir de una población de 67 alumnos. Esto se debe a la inexperiencia inicial de los administradores con la aplicación, lo que les conducía, por ejemplo a dar de alta a un mismo usuario con correos diferentes etc.

Esta clase es la cuarta entidad que ocupa mayor tamaño en el almacén (279 Bytes por entidad), y los atributos del tipo de dato *String*, los cuales cuentan con un 5%, son: 'la lista de

identificadores de libros subidos por el profesor', 'el correo electrónico', 'el perfil de usuario (Profesor o alumno)'y datos básicos de un usuario 'nombre, apellido, DNI,'. El tipo de dato *Integer* agrupa únicamente a las listas de identificadores de grupos a los cuales pertenece el usuario, lo que justifica su bajo porcentaje.

	Entidades
Tamaño total	**71 KBytes**
número	**258**
Promedio	**279 Bytes**

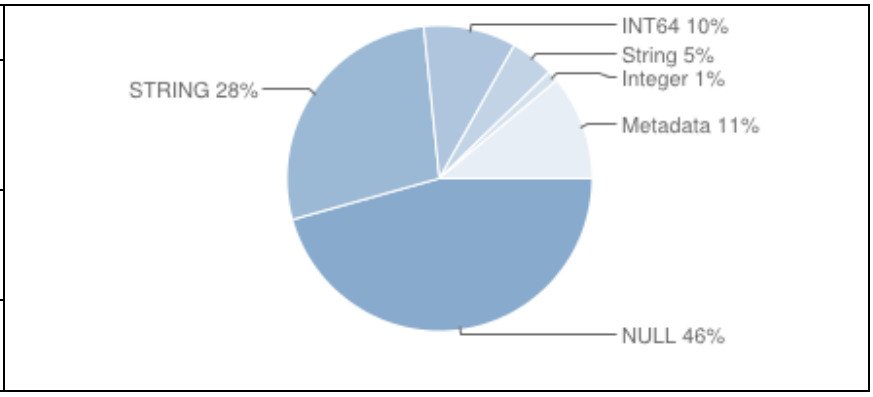

Tabla 7 Estadística de datos de la Entidad User

Es importante explicar el alto porcentaje de elementos NULL observado. Esto es debido a que atributos como: nombre, DNI y apellidos, casi nunca son cumplimentados por los usuarios. De igual forma al tener muchos usuarios mal creados (mismo usuario, diferentes correos), sus campos se generan vacíos, y no son nunca asignados a grupos, lo que hace que las listas de identificadores de grupos sea también NULL.

Otra de las razones de este resultado es que el usuario profesor o alumno es una misma entidad, diferenciada por el atributo *perfil*. En el caso de que el usuario sea un estudiante, las listas de identificadores de libros se encontrarán vacías, debido a que los estudiantes no pueden subir libros al almacén. De igual forma, existe un identificador de configuración de anotación, que permite establecer la visibilidad por defecto de las anotaciones, el tipo de letra de los comentarios, etc. No obstante, esta funcionalidad no se llegó a implementar en la versión actual, por lo que los usuarios creados resultaron ser grabados con esta propiedad a NULL.

4.3.7 Estadísticas de la base de datos

Por último la Tabla 8 muestra las estadísticas de la base de datos completa. De hecho, la primera estadística evidencia cómo la información añadida por el GAE (metadata, STRING, etc.) ocupa la mayor parte del tamaño (sobre el 76% del espacio). Del resto, como se ve reflejado, la propiedad Text, que representa el comentario de la aplicación es la que ocupa mayor cantidad de

espacio en el almacén. De igual forma resulta interesante observar que las referencias nulas suponen una cantidad de espacio no despreciable: un 4%. Esto es ocasionado mayormente por la entidad *User*, ya que, como se indicó anteriormente, la mayoría de los usuarios simplemente trabaja con el campo de email, y olvida cumplimentar su nombre, apellido y DNI. A esto hay que añadir que existen 191 usuarios redundantemente creados.

La figura de la parte inferior de la Tabla 8, muestra el porcentaje que ocupan las entidades en la base de datos, evidenciando que la entidad que ocupa mayor tamaño es *Annotation*, como es de suponerse, seguida de sus anclas.

Es importante notar, así mismo, el espacio relativamente pequeño ocupado por los libros subidos por los profesores. Aunque los libros se suben página a página en formato imagen, dichas imágenes no se almacenan en el *Datastore*, sino en otro servicio de Google que es el Blobstore. Lo que realmente se almacena en el *Datastore* es un delegado de clase *BlobInfo*, que contiene toda la información del blob (imagen subida). Aquí se ve reflejado que ocupa el 9% del almacén. Como se explicó anteriormente, para efectos de uso reales, este dato es incluso mayor que el que se debería esperar: esto es debido a que existen 8 libros repetidos, cada uno con una cierta cantidad de hojas; por cada hoja subida se genera un blob, y por cada blob un entrada en la clase de *BlobInfo*.

Así mismo se puede notar que la clase *Tag* (llamada en la aplicación *File*) ocupa el tercer lugar de tamaño en el almacén entre las entidades creadas por la aplicación. Esto evidencia la importancia de esta clase para los usuarios, ya que existen 394 entradas, que representan las hojas de la taxonomía creada por los profesores. Esto quiere decir que en promedio cada profesor cuenta con 65 *tags* para un número aproximado de 4,6 actividades, lo que indica que, en promedio, cada actividad (libro leído) cuenta con aproximadamente 14 tags, que son utilizadas para etiquetar las anotaciones, pertenecientes a las taxonomías y folksonomías generadas por los estudiantes.

Por último, los usuarios representan el 6 % del almacén de datos, pero como se dijo anteriormente existen 191 usuarios mal creados, de los cuales sólo se guardó su referencia y el correo electrónico, siendo los demás valores NULL.

	Entidades
Tamaño total	2 MBytes
Número	2,923

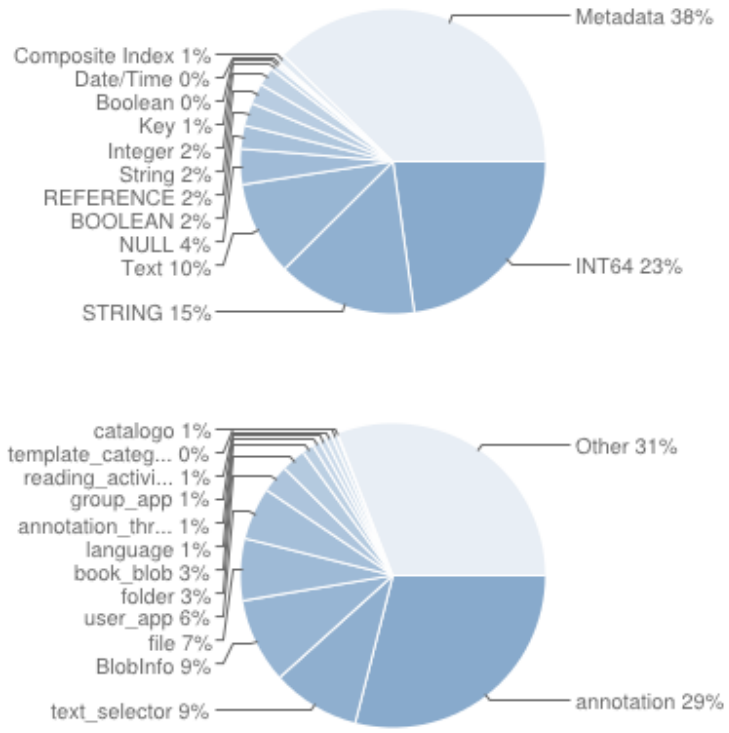

Tabla 8 Estadística de datos de las entidades de la aplicación

4.4 A modo de conclusión

En este capítulo se describió el uso que se le da a las entidades en la aplicación, mediante la observación de las entradas creadas y almacenadas en la base de datos, y se vio que las anotaciones son sin duda los objetos que ocupan mayor tamaño en el almacén. Resulta evidente que la inexperiencia de los usuarios en la aplicación hace que se crearan objetos prácticamente vacíos en algunos casos, como ocurre con la creación de 191 usuarios por encima de la población real que realizó las pruebas de 67 alumnos. Esto generó que mucho de los atributos se guardaran como NULL, y es lo que muestra el gráfico correspondiente al uso global de la base de datos, con un 9% de objetos NULL. De la misma forma, esta inexperiencia generó entradas redundantes en el almacén debido a los 8 libros repetidos subidos por los profesores. Aunque se desconoce el número de hojas por libro, se sabe que, por cada hoja que haya sido repetida, se

genera un blob, y con éste una entrada *BlobInfo*, entidad que ocupa un 9% del espacio total. Este análisis evidencia, quizá, algunos problemas de usabilidad en la aplicación, que deberán ser investigados y solucionados.

Resulta también interesante el descubrir que aproximadamente existe 14 tags para cada actividad, considerando los creados por los profesores y los creados por los alumnos. Este hecho evidencia la importancia práctica para los usuarios del sistema de clasificación incluido en la RIA.

Por último, mediante este análisis de datos sobre las entidades se descubre que los datos de toda la aplicación ocupan en el almacén tan solo 2MB, para un total de entidades distribuidas en 2923 entradas. Dado que las pruebas realizadas no son despreciables en lo que se refiere al número de participantes (profesores y alumnos), puede concluirse que los requisitos en espacio de almacenamiento de la aplicación son razonables.

Capítulo 5 - Conclusiones y Trabajo Futuro

5.1 Conclusiones

En este proyecto de investigación se ha explorado un emergente tipo de sistemas basados en web capaces de simular acciones similares a programas de escritorio. De igual forma se ha estudiado los diferentes dominios de aplicación en donde estos sistemas han sido implementados, haciendo particular énfasis en los dominios de las Humanidades Digitales y de la Educación, dominios a los cuales pertenece el sistema descrito en este proyecto. Se evidencia la complejidad en el diseño y construcción en estos sistemas debido a que la ingeniería de las RIAs pertenece a un campo de investigación relativamente nuevo en el área de la Ingeniería de Software. Se explica la arquitectura y patrón de diseño utilizado para la implementación de una RIA capaz de anotar obras literarias de manera colaborativa, así como el modelo de datos subyacentes, y los servicios ofrecidos que realizan la lógica de negocios de la aplicación.

El proyecto práctico (la aplicación), pertenece al proyecto “Collaborative Annotation of Digitilized Literary Texts” el cual está siendo financiado por Google, por lo que pareció prudente en un principio, ceñirse al uso de sus tecnologías, no sin antes, evaluar tecnologías y marcos cercanos para la implementación de RIAs: GWT (de cara al cliente), y Google Application Engine, como servidor de la aplicación, servidor que muestra un alto rendimiento en sus transacciones y permite, mediante su motor de base de datos *Big Table*, ofrecer alta escalabilidad en la aplicación. A lo largo del desarrollo de @note se ha experimentado con las limitaciones y desventajas que presenta este servidor de aplicación, mostrando los *workarounds* que se describen en foros oficiales de GAE, ideando otros nuevos, y utilizándolos durante la implementación.

Aparte de los aspectos tecnológicos, es necesario, así mismo, resaltar la importancia de implicar activamente a los expertos en el dominio de aplicación de la RIA en el proceso (en nuestro caso, el equipo de filólogos del grupo LEETHI), sobre todo en lo relativo a los aspectos de interacción y presentación.

Actualmente se dispone de una primera versión estable y funcional de la RIA. Dado que dicha aplicación es una aplicación de complejidad no trivial, desarrollada en un ámbito académico con las limitaciones en recursos que ello conlleva (en concreto, la aplicación fue programada por dos personas, bajo la asistencia de dos profesores de la Facultad de Informática),

puede concluirse que las tecnologías y el enfoque empleado constituyen una solución viable para el desarrollo de este tipo de aplicaciones, a pesar de las limitaciones encontradas y descritas en este trabajo.

Para finalizar, cabe destacar que algunos de los resultados preliminares discutidos en este trabajo de investigación aparecen publicados en dos ponencias presentadas en eventos internacionales: (Ruiz et al., 2012), (Gayoso et al., 2012)

5.2 Trabajo Futuro

@note aún se encuentra en una etapa media en su desarrollo. De esta forma, quedan abiertas y en discusión funcionalidades futuras que pueda ofrecer el sistema. En particular, se apuntan las siguientes:

- Una evaluación más profunda con los usuarios finales, a fin de resolver los problemas de usabilidad sugeridos por el análisis realizado en esta memoria.
- Integración de un filtro de *servlets*, con el fin de registrar eventos a llamadas a los servicios RPC, para mostrar estadísticas de las transacciones realizadas al servidor. El SDK de GAE incluye estas herramientas necesarias para evaluar el rendimiento de la RIA.
- Depuración de código fuente
- Introducción de un patrón de diseño más elegante que el actualmente utilizado (como por ejemplo MVC o MVP).
- Migración de la aplicación fuera de Google Application Engine a un servidor propietario de la Universidad y del grupo ILSA.
- Ampliación en las funcionalidades en el sistema @note, como por ejemplo: la exportación de anotaciones en múltiples formatos flexibles y editables que le permitan a los usuarios trabajar con éstas de forma off-line
- Búsqueda de un estándar de intercambio de anotaciones para futuras interconectividades con otros sistemas referentes a este ámbito estudio, siguiendo iniciativas como, por ejemplo, OAC.

Referencias

Anderson, C., 2010. *Pro Business Applications with Silverlight 4*. Apress.

Brambilla, M., Preciado, J.C., Linaje, M. & Sanchez-Figueroa, F., 2008. Business Process -based Conceptual Design of Rich Internet Applications. International Conference on Web Engineering ICWE'08.

Busch, M. & Koch, N., 2009. IST 016004 *Rich Internet Applications: state of the art*. Technical Report 0902. Institute for Informatics. Ludwig-Maximilians-Universität München, Germany

Calvar, G., Couta, J. & Thevenin, D., 2003. A unifying reference framwork for multitarget user interfaces. Interacting with computers 15(3), pp. 289-308.

Cowan, D. & Lucena, P.d., 1995. Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. IEEE Transactions on Software Engineering 21(3), pp. 229-243.

Demoulins, C., Azouaou, F. & Imag, C., 2006. MemoNote, a semantic and personal memory tool based on annotations made on pedagogical documents. Greece, 2006. 3rd International Conference on Engineering Education.

Dolog, P. & Stage, J. 2007. Designing Interaction Spaces for Rich Internet Applications with UML. International Conference on Web Engineering ICWE'07

Feldt, K., 2009. *Programming Firefox: Building Rich Internet Applications with Xul*. O'Reilly.

Freeman, A., 2011. *The Definitive Guide to HTML5*. Apress.

Frommholz, I. & Fuhr, N., 2006. Probabilistic, Object-oriented Logistics for Annotation-based Retrieval in Digital Libraries. Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries, June 11-15, 2006, Chapel Hill, NC, USA

García-Zubia, J., Hernández, U. & Orduña, P., 2007. WebLab-GPIB at the University of Deusto., 2007. Proceedings of the REV 2007 Conference.

García-Zubia, J., Orduña, P., López-de-Ipiña, D., Alves, G.R. 2009. Addressing Software Impact in the Design of Remote Laboratories. IEEE Transactions on Industrial Electronics, 56(12), 4757-4767

Garret, J.J., 2007. Ajax: A New Approach to Web Applications. Adaptive Path. Disponible en: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

Gayoso, J. Ruiz, C., et al., 2012. A Flexible Model for the Collaborative Annotation of Digitized Literary Works. In *Proc. of the 2012 Digital Humanities Conference DH*. Hamburg, 2012.

Gayoso, J., 2012. Modelo para la Anotación Colaborativa de Textos Literarios Digitalizados. Proyecto Fin de Máster. Facultad de Informática UCM. 2012

Gómez, J. & Cachero, C., 2003. OO-H Method: Extending UML to model web interfaces. Information Modelling for Internet Applications, pp. 144-173. Idea Group Publishing.

Google, C., 2011. *Google Web ToolKit Documentation*. code.google.com/webtoolkit.

Han-Zhen, W., Yang, S. & Su, Y.-S., 2008. Free-Form Annotation Tool for Collaboration. 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing.

Haslhofer, B., Simon, R., Sanderson, R., van de Sompel, H. 2011. The Open Annotation Collaboration (OAC) Model. 2011 Workshop on Multimedia on the Web MMWEB '11.

Heath, T. & Bizer, C., 2011. *Linked Data: Evolving the Web into Global Data Space*. Morgan & Claypool Publishers.

Jay, F. & Nezlek, G.S., 2007. Rich Internet Applications The Next Stage of Application Development. 29th International Conference on Information Technology Interfaces

Kazoun, C. & Lott, J., 2007. *Programming Flex 2: The Comprehensive Guide to Creating Rich Media Applications with Adobe Flex*. O'Reilly.

Koch, N., Zhang, G., Baumeister, H. 2008. UML-Based Web Engineering: An Approach Based on Standards. In *Web Engineering: Modelling and Implementing Web Applications*, 157-191

Koivunen, M.-R., 2005. Annotea and Semantic Web Supported Collaboration. 2nd European semantic web conference.

Labriola, M. & Tapper, J., 2011. *Adobe Flex 4.5 Fundamentals*. Adobe Press.

Linaje, M. & Sanchez-Figueroa, F., 2009. Domain-Specific Model for Designing Rich Internet Applicaction User Interfaces. *Springer*.

Linaje, M., Preciado J.C. & Sanchez-Figueroa, F., 2007. A method for model based design of rich internet application interactive user interfaces., 2007. International Conference on Web Engineering, LNCS Vol. 4607.

Mahemoff, M., 2007. Ajax Design Patterns., 2007. O'Reilly.

Martínez-Ruiz, F.J., Muñoz Arteaga, J. & Jean Vanderdonckt, J.M., 2007. A First Draft of a Model-driven Method for Designing., 2007. 4th Latin American Web Congress (LA-Web'07), pp. 32-38, IEEE.

McGann, J., 2008. The future is digital. *Journal of Victorian Culture* 13(1), pp. 80-88.

Mullet, K., 2004. The Essence of Effective Rich Internet Applications. Macromedia Whitepapers.

Preciado, J.C., Linaje, M., Comai, S., & Sánchez-Figueroa, F. 2007. Designing Rich Internet Applications with Web Engineering Methodologies., 2007. Proceedings of the International Symposium on Web Site Evolution.

Roch, T.R.d., Willrich, R., Fileto, R. & Tazi, S., 2009. Supporting Collaborative Learning Activities with a Digital Library and Annotations. Education and Technology for a Better World. IFIP Advances in Information and Communication Technology, 302/2009, pp. 349-358

Ruiz, C., Gayoso, J., et al., 2012. Web-services API in @Note. In *Proc. of the INTEREDITION Symposium on Scholarly Digital Editions, Tools and Infrastructure*., 2012.

Sanz, A. & Goicochea de Jorge, M., 2009. what (cyber)reading for the (cyber)classroom?. Neohelicon 36(3), pp. 533-550.

Schwabe, D. & Rossi, 1998. *An Object Oriented Approach to Web-Based Application Design*. Theory and Practice of Object Systems, 4(4), 207-225

Tazi S, A.-T.Y.D.K., 2003. Editing pedagogical intentions for document reuse. 4th IEEE Technology Based Higher Education and Training.

Toffetti, G., 2007. *Conceptual Modeling and Code Generation of Data-Intensive Rich Internet Applications*. PhD Thesis. Politecnico di Milano

Trias, F., López-Sanz & Esperanza, M., 2010. Estudio comparativo de diferentes propuestas dirigidas por modelos para la implementación de RIAs. XVI Jornadas de Ingeniería del Software y Bases de Datos

W3C, 1999. http://www.w3.org/1999/02/26-modules/User/Annotations-HOWTO.

W3C, 2001. http://www.w3.org/XML/Linking.

Wohed, P et al. 2008. On the Suitability of BPMN for Business Process Modelling. 4th International Conference on Bussines Process Management

Woolston, D., 2007. *Pro Ajax and.NET 2.0 Platform*. Apress.

Yang, D. 2010. *Java Persistence with JPA*. Outskirts Press

Zundert, J.v., 2009. AlfaLab: Construction and Deconstruction of a digital humanities Experiment. Fifth IEEE International Conference on e-science.