El proyecto Sméagol de la Càtedra de Programari Lliure de la Universitat Politècnica de Catalunya

The Sméagol Project of the Càtedra de Programari Lliure chair at the UPC

Alex Muntada, Àngel Aguilera, Isabel Polo

Resumen

El proyecto Sméagol es un piloto de proyecto de colaboración entre diferentes unidades estructurales de la Universitat Politècnica de Catalunya (UPC), coordinado a través de la Càtedra de Programari Lliure (CPL, Cátedra de Software Libre). El objetivo del proyecto es lograr trabajar tal como se hace en las comunidades de software libre, pero dentro del contexto de la UPC, para desarrollar una herramienta estratégica que se pueda implantar paulatinamente con la ayuda de todos los participantes. El proyecto sirve al mismo tiempo para introducir a los participantes en el uso de las metodologías ágiles, el desarrollo basado en tests, los ciclos cortos de desarrollo, las arquitecturas de tipo REST y distintas herramientas de trabajo colaborativo.

Palabras clave: metodologías ágiles, desarrollo basado en tests, REST, comunidad, software libre.

Summary

The Sméagol project is the pilot of a collaborative project between different structural units of the Polytechnic University of Catalonia (UPC), co-ordinated through the Càtedra de Programari Lliure (CPL, Freeware Chair). The project aims to work in the same way as freeware communities, but within the context of the UPC, to develop a strategic tool that can be introduced gradually with the help of all the participants. The project also serves to introduce the participants to the use of flexible methodologies, test based development, short development cycles, REST architectures and different collaborative working tools.

Keywords: flexible methodologies, test based development, REST, community, freeware.

El objetivo es lograr trabajar tal como se hace en las comunidades de software libre, pero dentro del contexto de la UPC

1. Introducción

La UPC es una organización grande que se distribuye geográficamente en distintos campus y organizativamente se compone de diversas unidades estructurales (escuelas, departamentos, centros de investigación, etc.) que cuentan con personal de todo tipo. Esta organización tiene ventajas claras: autonomía, flexibilidad y proximidad a los usuarios. Pero al mismo tiempo puede suponer también algunos inconvenientes: dispersión de los recursos y aislamiento de otras unidades. Del mismo modo, tanto el personal TIC de la universidad como los proyectos que lleva a cabo, reflejan esta organización, y como resultado a menudo ocurre que existen soluciones muy parecidas en distintas unidades estructurales que se han realizado de forma totalmente aislada.

El objetivo era obtener una solución útil, escalable y distribuida para toda la universidad

A finales del año 2006 la CPL de la UPC creó una lista de correo abierta e invitó a todo el personal TIC de la universidad para estudiar la posibilidad de realizar un proyecto de forma conjunta entre distintas unidades estructurales siguiendo las metodologías de las comunidades de software libre. El objetivo principal era encontrar la forma de realizar un proyecto de interés para varias unidades, repartiendo la carga de trabajo entre sus distintos centros de cálculo, para obtener una solución útil, escalable y distribuida para toda la universidad.

Durante algunos meses se hicieron propuestas para distintos temas y finalmente se decidió escoger la gestión de reservas de recursos debido a que se trata de un tema estratégico en varias unidades estructurales de la UPC. En la actualidad existen soluciones muy diversas implantadas en las distintas unidades: algunas se basan en herramientas realizadas por los propios centros de cálculo, otras han sido realizadas a medida por empresas externas. Existen también algunas que usan herramientas de software libre, etc. En esta situación sería muy complejo diseñar una solución que permitiera gestionar todas estas herramientas de forma uniforme: distintas bases de datos, distintas interfaces de usuario, distintos sistemas operativos, etc.

Poco después de tomar la decisión de cuál sería el tema del proyecto, la CPL convocó a los responsables de distintas unidades estructurales de la UPC y de sus respectivos centros de cálculo para estudiar cómo enfocar el proyecto, determinar su interés en él y la disponibilidad de personal TIC para participar en su desarrollo. En esta reunión la CPL presentó la idea de desarrollar una serie de implementaciones de referencia para cada uno de los componentes de un sistema de reservas de recursos basado en una arquitectura distribuida y federada.

Los tres componentes principales de esta arquitectura son los clientes, los servidores y los agregadores

Los tres componentes principales de esta arquitectura son los clientes, los servidores y los agregadores. La comunicación entre ellos se basa en API de tipo REST. Los clientes pueden ser de muchos tipos: desde agendas electrónicas y móviles a robots automatizadores de reservas, pasando por el clásico navegador web. Una unidad estructural puede tener uno o más servidores, según sus necesidades, estructura interna, etc. Finalmente, los agregadores pueden tener varios papeles: como frontend para obtener una visión centralizada de todos los servidores, como proxies que ofrezcan vistas de servidores, como pasarelas entre el sistema y otros gestores de reservas existentes, etc.

Las posibilidades que ofrece este proyecto son muchas y la intención no es definirlas desde el principio sino a medida que el proyecto avance y surja la necesidad por parte de sus usuarios. En este sentido, no se apuesta por un diseño tradicional según los principios de la ingeniería del software. Por el contrario, se siguen las recomendaciones de las metodologías ágiles para diseñar pequeñas partes del proyecto que permitan avanzar sin llegar a condicionar la forma final del mismo. Se prioriza la adaptación al cambio por encima de una solución clásica de tipo contractual debido a que la UPC es un organismo muy grande y complejo, en el que no es posible dar con una solución que sirva a todo el mundo, sobretodo teniendo en cuenta la autonomía de muchas de estas unidades. Además, la experiencia de las metodologías ágiles se sustenta sobre la idea que los diseños cambian inevitablemente cuando uno se pone a desarrollarlos, pero sobre todo cuando los usuarios empiezan a usar las herramientas. Por ello se prefiere desarrollar a partir de diseños más simples y acotados, que evolucionen según las necesidades que los propios usuarios detecten en el uso diario.

Se prioriza la adaptación al cambio por encima de una solución clásica de tipo contractual

2. Objetivos del proyecto

El objetivo práctico del proyecto es dar una solución genérica a un problema común dentro de la UPC. El producto que surgirá del proyecto en este caso será un gestor de reservas de recursos distribuido y federado, que ofrezca la posibilidad de aglutinar al mismo tiempo a otros gestores de recursos existentes a través de una interfaz común.

Pero en el fondo, el objetivo práctico es un pretexto para lograr otros objetivos más interesantes para el futuro del personal TIC y de la propia universidad.

2.1. Aprender nuevas metodologías

A menudo la carga de trabajo y la repetición producen un estancamiento en la forma de abordar los problemas que a la larga puede ser perjudicial tanto para el personal TIC como para las unidades a las cuales pertenecen. No hay tiempo suficiente para aprender y experimentar por cuenta propia y los cursos de formación y reciclaje no son suficientes o adecuados. Por este motivo y teniendo en cuenta la experiencia en varias comunidades de software libre, la CPL hace una apuesta firme para buscar un contexto en el que este aprendizaje de nuevas metodologías sea no sólo deseable sino necesario.

En este sentido, las metodologías ágiles aportan frescura a la forma de plantear soluciones a los problemas. Ya no se trata de realizar complicados diseños que den solución a todos los problemas a la vista, al contrario, se asume que no es posible tener en cuenta todos los problemas de entrada. Se divide la solución en trozos, se separan los problemas, se abordan primero los que más interesan a los usuarios, se realizan prototipos y se deja a los usuarios que los usen y que den su opinión, que pidan mejoras, etc. Pero cuando se habla de prototipos se trata de soluciones completas, usables y útiles en el mundo real en mayor o menor medida. Estos prototipos evolucionan a lo largo del tiempo y se convierten en soluciones estables a medida que las propuestas de los desarrolladores y las ideas de los usuarios convergen.

Una herramienta básica para poder realizar este desarrollo ágil son los tests

En el planteamiento tradicional de la ingeniería del software de diseñar desde un principio toda la solución, a menudo ocurre que ésta se aleja demasiado del usuario y a veces se produce un rechazo insalvable. En definitiva, desarrollo ágil significa tener capacidad para adaptarse a los cambios sin muchas dificultades, ya sea por los cambios determinados por las necesidades de los usuarios como por cualquier otro motivo que pueda afectar al desarrollo de un proyecto.

Una herramienta básica para poder realizar este desarrollo ágil son los tests y lo que se conoce como desarrollo basado en tests, que no es exclusivo de las metodologías ágiles, por supuesto. La idea del desarrollo basado en tests se basa en realizar al mismo tiempo pruebas unitarias y funcionales de las distintas partes de la solución. Los tests unitarios permiten verificar que cada pequeña parte del desarrollo funciona como es debido y los tests funcionales verifican que los procesos que usan estas pequeñas unidades funcionan correctamente en su conjunto, permitiendo cubrir todos los aspectos de la solución. De esta forma, si se produce algún cambio de comportamiento en la solución sin que uno se de cuenta, los tests no sólo harán saltar la alarma sino que ayudarán a encontrar rápidamente dónde se ha producido el problema.

Desarrollando primero los tests, uno puede hacerse una idea más precisa de cómo hay que diseñar las herramientas

Pero el desarrollo basado en tests puede llevarse a un extremo todavía más implicado en el diseño de la solución. Teniendo en cuenta que los tests usan las herramientas diseñadas para dar solución a un problema, como lo harán los usuarios en el futuro, existe la idea de desarrollar antes los tests que la propia solución, como si se tratara de casos prácticos de uso. Es decir, desarrollando primero los tests uno puede hacerse una idea mucho más precisa de como quiere que se usen las herramientas y por tanto cómo hay que diseñarlas.

Al principio, fallarán los tests que no tengan todavía implementada su parte de solución, pero eso es una ventaja porque nos permite tener una visión general de las partes que se han ido completando y de las que quedan pendientes. Parece una idea revolucionaria pero en el fondo se trata de ser prácticos y aprovechar todo el potencial de los tests.

En cuanto a la arquitectura de la solución, hoy en día la mayoría de soluciones se basan en web y en los llamados servicios web debido al potencial que tiene una arquitectura que puede aprovechar toda la infraestructura existente para web. Los servicios web tradicionalmente han trasladado el modelo de Remote Procedure Call (RPC, Llamada a un Procedimiento Remoto) y lo han aplicado a la capa de transporte HTTP usando XML como lenguaje de codificación. Sin embargo, el sistema web se diseñó de

forma mucho más simple, con pocas operaciones que permiten gestionar distintos recursos mapeados en un espacio de nombres concreto, los Uniform Resource Identifier (URI, Identificador Uniforme de Recurso). En este sentido, las arquitecturas de tipo REST reivindican el retorno a un modelo más simple de servicios web, más parecido a como funcionan los navegadores y alejado del modelo de los RPC, tal como se explicará más adelante.

2.2. Utilizar herramientas de trabajo en grupo

Las herramientas de trabajo en grupo son un pilar básico para cualquier comunidad de software libre. En un contexto en el que no existe la proximidad física ni horaria son imprescindibles las herramientas de comunicación y coordinación del trabajo en grupo.

La herramienta más común, sin tener en cuenta el omnipresente correo electrónico, sería un sistema de control de versiones para el código a desarrollar. Estos sistemas permiten fácilmente hacer un buen seguimiento de los cambios, documentar los motivos por los cuales se realizan y almacenan un histórico de muy fácil acceso. Pero, sobre todo, facilitan a los distintos miembros del proyecto la posibilidad de trabajar de forma concurrente y sin estorbar a los compañeros.

Las herramientas de trabajo en grupo son un pilar básico para cualquier comunidad de software libre

Otra herramienta muy común es el gestor de proyecto. Existen muchos tipos de gestores de proyecto, algunos creados específicamente para proyectos de desarrollo de software. En general todos estas herramientas permiten hacer un seguimiento de cómo evoluciona el proyecto y crear listas de tareas en forma de tiques para repartir el trabajo y tener un mejor control de los pasos a seguir, pero también incluyen gestión de documentación y a veces incluso del sistema de control de versiones. Cuando estas herramientas se abren al público permiten la colaboración de personas externas al proyecto, tal como ocurre en las comunidades de software libre, y ofrecen la posibilidad de dar visibilidad y transparencia a todo el desarrollo del proyecto.

Cuando no existe proximidad entre los miembros del equipo, la única posibilidad para comunicarse de forma síncrona es la mensajería instantánea. Esta herramienta permite tanto tener conversaciones individuales como reuniones no presenciales de forma mucho más fácil que con cualquier otra, y además cada uno conserva un registro de la conversación, cosa que puede ser muy útil para hacer resúmenes o actas. Cierto es que al principio puede resultar chocante si uno no está acostumbrado o parecer poco ágil tener que escribir en vez de realizar una llamada telefónica, pero en poco tiempo se empiezan a percibir los beneficios de este mecanismo: no es intrusivo, es económico y al alcance de todo el mundo, guarda registro de las conversaciones, permite copiar y pegar (ejemplos, enlaces, etc.), puede participar cualquiera, etc.

Cuando no existe proximidad entre los miembros del equipo, la única posibilidad para comunicarse de forma síncrona es la mensajería instantánea

En el caso del proyecto *Sméagol* las herramientas usadas son *Subversion*[1] para el control de versiones, *Trac*[2] para la gestión del proyecto y XMPP/*Jabber*[3] para la mensajería instantánea.

2.3. Encontrar retos que incentiven

La rutina diaria puede ser uno de los factores de desmotivación más importantes en el trabajo. Para despejar esta desmotivación es necesario afrontar nuevos retos y aprender nuevos conceptos que resulten interesantes a cada uno. En el proyecto Sméagol el consenso es muy importante, por lo que se tienen en cuenta todas las opiniones y se valora muy positivamente la participación activa de los miembros del equipo. Algunas tareas se plantean como pequeños retos del proyecto y a menudo se reparte el trabajo a realizar según las preferencias e intereses particulares de cada uno. Es muy importante estar contentos con el trabajo y los compañeros ya que la felicidad es un incentivo difícil de superar.

Otro mecanismo para aportar aire fresco es la colaboración con otras personas no vinculadas directamente en el proyecto pero que tienen interés en el mismo, tanto si trabajan dentro de la UPC como si no. En el caso del personal TIC de otras unidades de la UPC el incentivo es aún mayor ya que se

trata de compañeros a menudo conocidos pero con los cuales no ha habido oportunidad de trabajar en común. Sus ideas y experiencia pueden ser de gran valor y aportan una visión fresca que en general será positiva para el proyecto.

3. Arquitectura del sistema

Debido a que el proyecto pretende encontrar una solución distribuida y federada para la gestión de reservas de recursos, y éste no es un problema de fácil solución sin la participación de toda la universidad, se ha escogido empezar por modelar una arquitectura sencilla pero escalable a largo plazo, centrando primero la solución de los problemas más interesantes y cercanos y dejando para más adelante los que se pueden obviar por el momento.

Así pues, se ha empezado definiendo una arquitectura con 3 capas: cliente, servidor y agregador. La capa del agregador será la que permita la distribución o federación del sistema, pero por el momento no es un problema a tener en cuenta. En cambio, tanto el cliente como el servidor son componentes esenciales.

Tanto el cliente como el servidor son componentes esenciales

3.1. Servidor

Un servidor en el proyecto Sméagol no es más que un backend que se encarga de gestionar recursos y sus respectivas reservas. Lo normal sería que exista un solo servidor por unidad estructural pero esta limitación no es estricta. La comunicación con el servidor se realiza mediante una API de tipo REST y el servidor del proyecto es una implementación de referencia de cómo debe funcionar. En este sentido, por ejemplo, sería posible crear distintas implementaciones del servidor en varios lenguajes de programación. Lo importante en este caso es que el servidor siempre hable el mismo lenguaje, tal como marca la API. Como es el servidor el que se encarga de proteger los tesoros de una unidad estructural se le llama Sméagol, en honor al personaje de los libros de J.R.R. Tolkien, y de ahí también el nombre del proyecto.

3.2. Cliente

Los clientes en cambio pueden ser de muchos tipos y con muchos propósitos distintos. El papel del cliente es el de hacer de frontend con el usuario del sistema y comunicarse con el servidor a través de la API de tipo REST que el servidor esté usando. Además el cliente puede añadir al sistema todas las características particulares que interesen a una unidad estructural y que no tienen cabida en el servidor (por ejemplo, autenticación de usuarios, permisos y roles, etc.), pero en el fondo no añaden al sistema funcionalidades que no tenga el servidor dentro del marco de la gestión de reservas de recursos.

El cliente hace de frontend con el usuario del sistema

Para facilitar el trabajo a los usuarios y la integración con sistemas existentes, se permite que las herramientas de calendario típicas puedan acceder a los datos del servidor como clientes usando el estándar iCalendar[4]. Así se permite muy fácilmente la publicación de los datos de las reservas en un lenguaje estándar y conocido de sobra, que abre todo un mundo de posibilidades que eran inalcanzables hasta el momento.

3.3. Agregador

Los agregadores añaden al sistema la posibilidad de compartir información de los servidores, localizar recursos, delegar reservas, etc. Pueden funcionar aparentemente como servidores ya que entienden su misma API de tipo REST pero no almacenan datos, tan sólo realizan el papel de intermediarios entre otros clientes y servidores, de los cuales mantienen cierta información.

Un uso típico para un agregador es el de directorio para una visión de campus de los sistemas de gestión de reservas de recursos. Las unidades que quieran poner a disposición del público su sistema de reservas tan sólo tienen que asociar sus servidores al agregador de campus. Pero el agregador puede además añadir la funcionalidad de distribución del uso de los recursos de todo el sistema, al tener a su alcance

la información de los servidores de las unidades, puede repartir las reservas según criterios más globales (por ejemplo, las reservas de aulas de un campus para las distintas asignaturas de diferentes escuelas de una universidad).

4. Servicios web de tipo REST

Según el W3C, un servicio web es un sistema software diseñado para soportar la interoperabilidad entre máquinas que interaccionan en una red[5]. Tradicionalmente estos servicios se han diseñado de forma similar a otras aproximaciones para resolver la interoperabilidad de sistemas (CORBA, DCOM, RMI), aprovechando el modelo existente de RPC pero transladándolo al protocolo HTTP y usando serialización de datos con XML.

La disertación sobre Representational State Transfer (REST) de Fielding[6] pretende recuperar el espíritu original de los servicios web, ante todo su simplicidad, pero también la importancia del espacio de nombres y de la accesibilidad a los recursos. Fielding no es muy concreto ideando una arquitectura REST, más bien define unos criterios y características que hacen que una arquitectura sea de tipo REST[7].

La disertación sobre REST de Fielding pretende recuperar el espíritu original de los servicios web

4.1. Cliente y servidor

La separación entre ambos permite que el cliente no tenga que preocuparse por cómo se almacenan los datos ni el servidor de cómo es la interfaz de usuario o de cuál es su estado. Mediante una interfaz uniforme que comunica a clientes y servidor es posible diseñar un sistema sencillo y escalable en el que tanto cliente como servidor pueden ser sustituidos sin problemas mientras la interfaz uniforme se respete.

4.2. Falta de estado

El cliente puede tener estado pero éste nunca será enviado al servidor. Cada petición del cliente contiene toda la información necesaria para realizar la operación deseada sobre un recurso. De esta forma el servidor es más escalable y tolerante a fallos de la red. Sin estado del usuario en el servidor es muchísimo más fácil diseñar un sistema en clúster para repartir la carga.

4.3. Caché de respuestas

Si cliente y servidor tienen soporte para el control de caché, es posible evitar que algunas peticiones repetidas lleguen de nuevo al servidor si el cliente ya la había realizado con anterioridad y la respuesta todavía no ha caducado, según la expiración que ha definido el servidor para ella. Esta característica no sólo mejora el rendimiento y la escalabilidad del sistema sino también el de la red.

El caché de respuestas evita que algunas peticiones repetidas lleguen de nuevo al servidor

4.4. Interfaz uniforme

Si cliente y servidor tienen la interfaz uniforme entre cliente y servidor es el pilar básico de cualquier arquitectura de tipo REST y debe consistir en los principios siguientes[8]:

- **Recursos direccionables:** los servicios web publican sus recursos en direcciones individuales que los identifican (URI) de forma única e inequívoca.
- **Representación de un recurso:** un recurso puede tener distintas representaciones e idealmente cada una de ellas con una dirección distinta, pero todas ellas identifican en realidad al mismo recurso. El servidor conoce el aspecto real de un recurso pero los clientes sólo conocen sus representaciones.
- **Conectividad:** la existencia de direcciones para los recursos permite crear enlaces entre ellos e incluso entre recursos de distintos servidores. Las operaciones básicas sobre estas direcciones son

4: lectura (GET), creación (POST), actualización (PUT) y borrado (DELETE). Exactamente las mismas operaciones disponibles en el protocolo HTTP. Mediante las direcciones y estas operaciones se permite acceder y manipular muy fácil y rápidamente la información de los servicios web, hecho que abre todo un mundo de posibilidades poco comunes hasta hace poco. La conectividad es lo que hace que la navegación web para las personas haya tenido un éxito impredecible. El mismo principio aplicado a los servicios web resulta igual de interesante o incluso más.

- **Seguridad e idempotencia:** las operaciones de lectura (GET) siempre son seguras, nunca van a cambiar el recurso leído. Por otro lado, las operaciones de actualización y borrado (PUT y DELETE) son idempotentes, es decir, repetir una misma operación de actualización o borrado siempre va a terminar con el mismo resultado. En cambio, la operación de creación (POST) no puede ser segura ni idempotente, aunque en este último caso se podría llegar a diseñar para que lo fuera.

4.5. Sistema por capas

En un sistema por capas el cliente no es capaz de determinar si está hablando con un servidor o en realidad lo está haciendo con un intermediario. Esta característica es imprescindible para poder diseñar sistemas distribuidos, con balanceo de carga y sistemas de caché, totalmente transparentes al cliente.

Las operaciones básicas sobre estas direcciones son: lectura, creación, actualización y borrado

5. Conclusiones

El enfoque del proyecto Sméagol ha sido desde un principio el de aprender y compartir experiencia y conocimientos entre compañeros de la UPC al mismo tiempo que se busca el incentivo que motive a los miembros del equipo a seguir adelante. Por otro lado, el no tener fijado ningún calendario (a parte del que se fije desde el propio proyecto) permite que el proyecto se vaya adaptando a los distintos cambios de horarios y dedicación de los miembros del equipo, tal como ocurriría en una comunidad de software libre. En conjunto es un escenario poco habitual para el personal TIC y se agradece el disponer de la confianza y autonomía para decidir el camino a seguir a medida que el proyecto avanza. En el camino se van aprendiendo técnicas y metodologías nuevas, se corrigen errores y se mejoran algunos aspectos. Además se conocen otros puntos de vista y se promueve la renovación y la investigación de novedades.

Tal vez la falta de liderazgo, el vaivén inicial de objetivos y el desconocimiento de las herramientas y de las metodologías han complicado el arranque inicial del proyecto. Pero pasado un tiempo se ha mejorado notablemente y se ha establecido una dinámica de trabajo constante, a pesar de la poca disponibilidad de tiempo de los miembros del equipo.

El enfoque del proyecto Sméagol ha sido aprender y compartir experiencia y conocimientos entre compañeros de la UPC

Finalmente, cabe destacar que como objetivo de fondo del proyecto se pretende que, una vez creada una comunidad de desarrolladores a su alrededor, las distintas aportaciones al proyecto lo mejoren y amplíen en su conjunto, dando lugar a una solución completa y viva para toda la UPC, disponible además como software libre para todo el mundo en el sitio web del proyecto[9]. El éxito definitivo sería conseguir que el experimento se pudiera repetir con otros proyectos.

Referencias

[1] Subversion (Nov 2009) *http://subversion.tigris.org/*

[2] Trac (Nov 2009) *http://trac.edgewall.org/*

[3] XMPP/Jabber (Nov 2009) *http://xmpp.org/ http://www.jabber.org/*

[4] iCalendar. RFC 5545 (Nov 2009) *http://tools.ietf.org/html/rfc5545*

[5] Web Services Glossary. W3C Working Group Note 11 February 2004 *http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice*

[6] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures. Chapter 5. Representational State Transfer (REST)*. University of California, Irvine, 2000 *http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm*

[7] Representational State Transfer. Constraints. Wikipedia (Nov 2009) *http://en.wikipedia.org/wiki/Representational_State_Transfer#Constraints*

[8] Richardson, Leonard; Ruby, Sam. *RESTful Web Services. Chapter 8. REST and ROA Best Practices.* O'Reilly 2007

[9] El proyecto Sméagol (Nov 2009) *http://devel.cpl.upc.edu/recursos*

Alex Muntada
Àngel Aguilera
Isabel Polo
Universitat Politècnica de Catalunya