Desastres 2.0. Aplicación de tecnologías Web2.0 en situaciones de emergencia

Julio Camarero
Depto Ingeniería de Sistemas Telemáticos
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Ciudad Universitaria s/n 28040 Madrid
Email: juliocamarero@gmail.com

Carlos A. Iglesias
Depto Ingeniería de Sistemas Telemáticos
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Ciudad Universitaria s/n 28040 Madrid
Email: cif@gsi.dit.upm.es

***Resumen*—This article presents a social approach for disaster management, based on a public portal, so-called *Disasters 2.0*, which provides facilities for integrating and sharing user-generated information about disasters. The architecture of Disasters 2.0 is designed following REST principles and integrates external mashups, such as Google Maps. This architecture has been integrated with different clients, including a mobile client, a multiagent system for assisting in the decentralised management of disasters, and an expert system for automatic asignation of resources to disasters. As a result, the platform allows seamless collaboration of humans and intelligent agents, and provides a novel web2.0 approach for multiagent and disaster management research and artificial intelligence teaching.**

I. Introducción

Los desastres o catástrofes naturales están asociados a una situación de caos donde la información suele ser incompleta e imprecisa y, precisamente, esta falta de información dificulta la toma de decisiones y la gestión efectiva de las catástrofes. Conforme al Secretariado Inter-Agencia de Naciones Unidas sobre la Estrategia Internacional para Reducción de desastres (UN/ISDR), entre las once lecciones aprendidas para la gestión de desastres, las dos primeras son [1]:

> *El conocimiento público es un elemento esencial para estar preparados para salvar vidas y el entorno.*
>
> *Los individuos y las comunidades juegan un papel importante en la gestión de riesgos de los desastres naturales.*

Este trabajo propone que las tecnologías de la web2.0 pueden ser una herramienta valiosa para contribuir en ambas medidas, favoreciendo el conocimiento público, así como la participación individual y social en la gestión de los desastres.

La web2.0 [2] ha demostrado el poder de la participación de los usuarios para crear contenidos, opininar y organizarse en redes sociales. Ejemplos como la wikipedia, o del.icio.us nos demuestran la potencia de esta inteligencia colectiva bien encauzada.

Este artículo propone integrar diferentes tecnologías normalmente agrupadas como tecnologías web2.0.

Por una parte, una aplicación potencial de esta inteligencia colectiva puede ser la gestión de desastres naturales. Si todas las personas pudieran informar en tiempo real de dónde se están produciendo, su magnitud o su seguimiento, el tratamiento que se les daría podría ser mucho más efectivo e inmediato. Sería como tener millones de ojos en todos los rincones del mundo trabajando por el bien común.

Por otra parte, la concepción del sistema en sí mismo se ha planteado como un sistema diseñado en torno al uso y provisión de servicios REST [3] que faciliten su combinación (mediante *mashups* [1]).

Este trabajo se ha desarrollado dentro del proyecto TSI Improvisa (TSI2005-07384-C03-01), que aborda el problema de la provisión de servicios de información en escenarios de catástrofes naturales, mediante el desarrollo de redes ad-hoc, computación orientada a servicios y agentes inteligentes. Se han explorado las aplicaciones de la web2.0 a la provisión de servicios de información, y a la gestión de la coordinación entre personas y agentes inteligentes para la gestión de las alertas, centrándose en la descripción de la plataforma web2.0.

El resto del artículo se estructura como sigue. La sección II presenta brevemente las tecnologías utilizadas. A continuación, la sección III describe la arquitectura del sistema Desastres 2.0, describiendo con detalle cada uno de sus componentes. La sección IV explica la aplicación de técnicas inteligentes al sistema. Por último, se describen los trabajos relacionados en la sección V y se recogen las conclusiones y trabajos actuales en la sección VI.

II. Tecnologías fundamentales

II-A. *REST y Restlets*

REST, acrónimo de *Representational State Transfer*, es una arquitectura basada en el modelo cliente-servidor de aplicaciones web que define la forma de representar, acceder y modificar datos en la red. REST entiende todos los datos como recursos y hace accesibles estos recursos mediante URIs *(Uniform Resource Identifiers)*.

[1] *Mashup: Traducido al Castellano como "aplicación web híbrida", es un sitio web o aplicación web que usa contenido de otras aplicaciones web para crear un nuevo contenido completo, consumiendo servicios directamente siempre a través del protocolo http. El contenido usado en un mashup es típicamente usado por terceros a través de una interfaz pública o usando una API.*

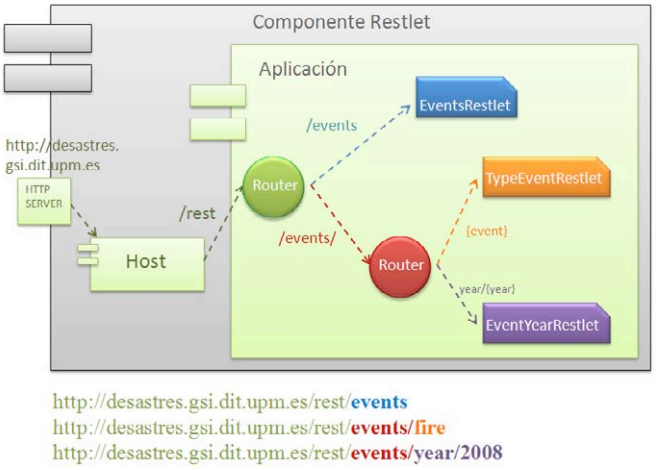

Figura 1. Ejemplo simplificado de encaminamiento con Restlets

Esta arquitectura se basa principalmente en cuatro principios:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición.
- Un conjunto de operaciones bien definidas; las primitivas HTTP GET, POST, PUT y DELETE para acceder a los recursos.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.
- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la misma: la representación de este estado en un sistema REST es típicamente HTML o XML.

Aplicar esta arquitectura a nuestra aplicación supone definir una URI para cada evento, recurso o víctima y permitir su eliminación o modificación mediante los principales métodos HTTP. Así, por ejemplo, bastaría con escribir en el navegador:

- */events* para obtener un listado de eventos
- */id/10* para acceder al evento cuyo id es 10
- */delete/id/10* para eliminar el elemento cuyo id es 10
- */put/10/info/nuevo_valor* para modificar el parámetro *info* al elemento 10
- */post/parametro1=valor1&parametro2=valor2&...* para crear un nuevo evento

Para construir esta arquitectura en nuestra aplicación utilizamos Restlet [4], un *framework* ligero que permite desarrollar REST en Java, desarrollado por *Noelios Consulting* y ofrecido en su página web como *open source*. El funcionamiento de este *framework* se basa en la creación de routers virtuales que mediante comparación de la URI que introducimos con diversos patrones nos redirige de distintas formas a la lógica de negocio de nuestro servidor. Podemos observar un ejemplo simplificado de esta arquitectura en la figura 1.

Los recursos accesibles por REST en nuestra aplicación se enumeran en la figura 2.

Método GET	
/id/{id}	elemento con identificador {id}
/year/{año}	desastres, recursos y víctimas ocurridos en este año
/date/YYYY-MM-DD	desastres, recursos y víctimas ocurridos a partir de la fecha insertada
/date/YYYY/MM/DD	igual que la anterior
/events	todos los desastres (eventos)
/events/{tipo}	muestra sólo eventos de un tipo (*fire*, *flood* o *collapse*)
/events/year/{año}	muestra sólo los eventos de ese año
/events/date/YYYY-MM-DD	sólo los eventos a partir de esa fecha
/resources	todos los recursos
/resources/{tipo}	muestra sólo los recursos de un tipo (*police*, *firemen* o *ambulance*)
/people	todas las personas (víctimas)
/people/{tipo}	muestra sólo las víctimas de un tipo (*slight*, *serious*, *dead* o *trapped*)
Método DELETE	
/delete/id/{id}	elimina el desastre con id {id}
Método POST	
/post/{parametros}	crea un nuevo marcador con los parámetros pasados
Método PUT	
/put/{id}/{parametro}/{valor}	modifica el parámetro {parametro} del marcador con id {id} con el nuevo valor {valor}

Figura 2. Recursos REST de Desastres2.0

II-B. *Google Maps*

Google Maps [5] es el nombre de un servicio gratuito de mapas ofrecido por Google. Consiste en un servidor de aplicaciones de mapas en la red que ofrece imágenes de mapas desplazables, así como fotos del satélite del mundo entero e incluso la ruta entre diferentes ubicaciones. Su utilidad para nuestra aplicación reside en que permite empotrar estos mapas en otras aplicaciones web y ofrece una amplia *API* (Interfaz de programación de aplicaciones) para poder interactuar con ellos. Dentro de este servicio se ofrecen funcionalidades como un geolocalizador, que nos proporciona la latitud y longitud de un punto a partir de una dirección. Google Maps proporciona a nuestra aplicación una interfaz sencilla y amigable con el usuario que le permite marcar desastres en el mapa o desplazar a los recursos (policías, ambulancias y bomberos).

II-C. *JSON*

JSON [6], acrónimo de *JavaScript Object Notation*, es un formato ligero para el intercambio de datos. Se presenta como alternativa al uso de XML por varios motivos:

- Es más simple que XML y por tanto, su procesamiento es más rápido
- Facilidad para ser analizado por JavaScript
- Los datos ocupan menos que en XML

```
{"desastre":{
        "id":    "1",
        "nombre": "Incendio",
        "victimas":   {
                "atrapados": "10",
                "heridos-leves": "0"
                }
        }
}
```

Figura 3. Ejemplo JSON

```
<desastre>
        <id>1</id>
        <nombre>Incendio</nombre>
        <victimas>
          <atrapados>10</atrapados>
          <heridos-leves>0</heridos-leves>
        </victimas>
</desastre>
```

Figura 4. Ejemplo XML

Actualmente se utiliza en menor medida que XML por ser un formato más reciente pero su facilidad para el intercambio de datos en aplicaciones Ajax hace que cada día esté más presente en la red, utilizado, por ejemplo, en aplicaciones de Google como Google Maps o de Yahoo como del.icio.us.

En nuestra aplicación, toda la información enviada desde el servidor por la lógica de negocio (a veces pueden llegar a ser grandes cantidades de datos) es enviada en formato JSON, lo que facilita el cliente JavaScript y reduce el tamaño de los datos a enviar.

Podemos observar un ejemplo reducido de desastre utilizando JSON (figura 3) y su equivalente en XML (figura 4), donde se observa su mayor tamaño.

III. Arquitectura propuesta

La arquitectura de Desastres 2.0 se ilustra en la figura 5. En este apartado se detallan sus componentes.

III-A. El Servidor Desastres 2.0

El servidor de la aplicación está desarollado con tecnología *Java Enterprise Edition* (Servlets y JSPs) y se ejecuta en un servidor Apache Tomcat. Es el responsable de:

- Almacenar de forma persistente la información de catástrofes en una base de datos
- Implementar la lógica de negocio de actualización y recuperación de esta información
- Ofrecer el acceso a la información mediante servicios REST implementados con Restlets, como se ha visto, que invocan la lógica de negocio de acceso a la información de desastres.

III-B. El cliente web Desastres2.0

El cliente web para nuestra aplicación tiene como elemento principal el mapa de Google Maps como se puede observar

Figura 5. Arquitectura de Desastres 2.0

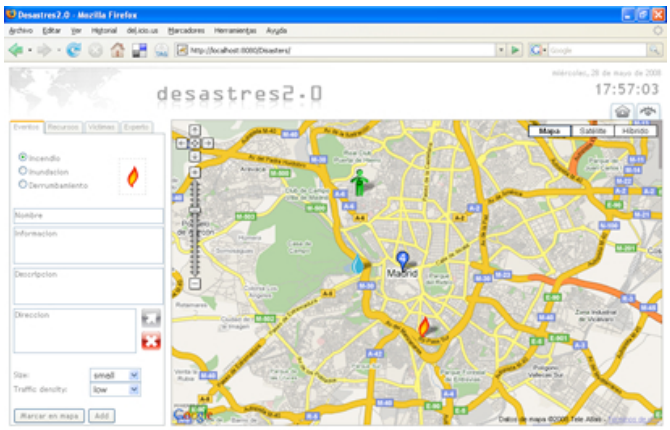

Figura 6. Interfaz gráfica del cliente web Desastres 2.0

en la figura 6. La aplicación proporciona tres formularios visualizables en un sistema de pestañas para añadir los tres tipos de marcadores distintos a nuestro mapa:

- *Eventos:* incendio, inundación o derrumbamiento
- *Recursos:* policías, bomberos o ambulancias
- *Víctimas:* atrapados, heridos leves, heridos graves y fallecidos

Para todos los marcadores es posible añadir un nombre, una descripción y una información asociada. Con objeto de cuantificar los desastres y su impacto para poder gestionarlos

Figura 7. Distintos marcadores para la representación de las víctimas, los desastres y los recursos

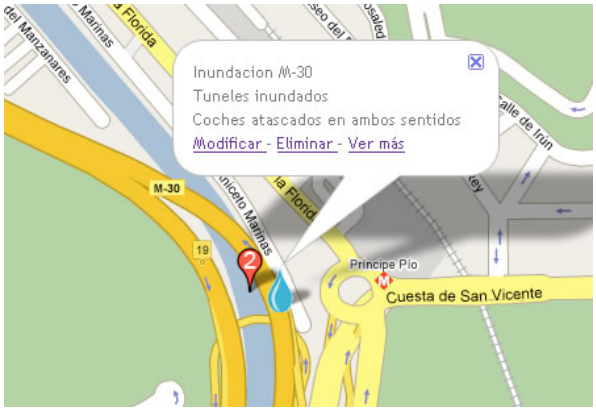

Figura 8. Información de un marcador

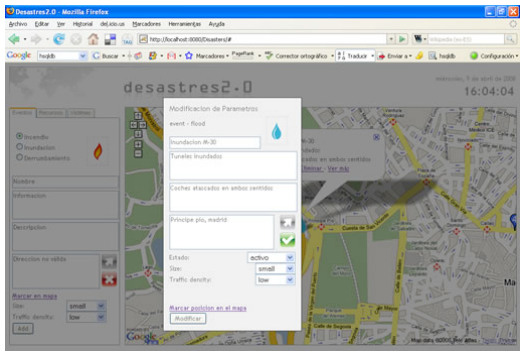

Figura 9. Modificaciones a un marcador

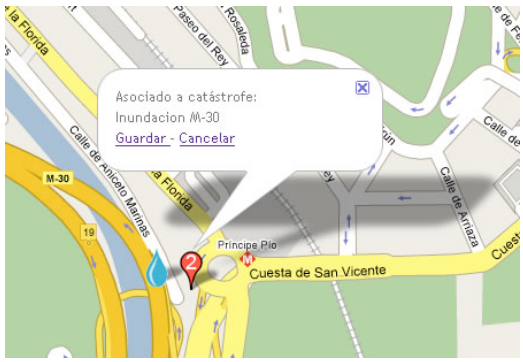

Figura 10. Asociación de un recurso o víctima a un desastre

mejor, es posible añadir informacion relativa a la magnitud del desastre y la densidad de tráfico del lugar donde ha ocurrido. En los recursos se puede especificar el número de unidades y en las víctimas el número de las mismas.

En función del marcador creado o del número de unidades especificadas se mostrará un icono diferente, como puede observarse en la figura 7.

El sistema ofrece una funcionalidad de *validación* de los desastres introducidos gracias al uso del servicio de geolocalización de Google; Al introducir una dirección, se verifica la misma y se efectúa un zoom del mapa a dicha zona.

La aplicación proporciona la información sobre cualquier marcador simplemente haciendo clic sobre su icono (figura 8). Junto a esta información se nos ofrecen diversas opciones como eliminar el marcador, modificarlo u obtener información más detallada sobre el mismo. Si decidimos modificar un marcador la aplicación nos presenta una interfaz sencilla donde se nos ofrece la posibilidad de modificar todos los parámetros del mismo (figura 9).

El cliente implementa, además, una funcionalidad de *asociación visual* de recursos. Tanto los recursos como las víctimas pueden estar asociados a un desastre. Las víctimas pueden estar causadas por un desastre y los recursos pueden estar destinados a solucionarlo. Para asociar víctimas o recursos a un desastre simplemente deberemos arrastrar el marcador de la víctima o el recurso hasta el desastre deseado. La aplicación asocia automáticamente el recurso o víctima al desastre más cercano (figura 10) y a continuación se colocará rodeando al desastre sin taparlo permitiendo así asociar más tipos de víctimas o recursos sin que se obstaculice su visión (figura 11).

Si en un momento dado se desea tener un resumen de los desastres, recursos o víctimas, la aplicación ofrece un cuadro con información resumida accesible desde una pestaña en la parte superior derecha del mapa (figura 12).

La aplicación ofrece también la posibilidad de mostrar en el mapa hospitales, comisarías y parques de bomberos

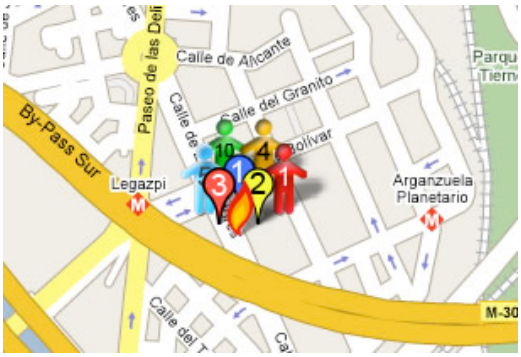

Figura 11. Asociaciones visibles para un desastre

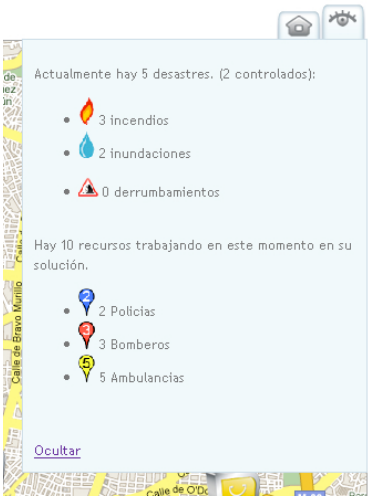

Figura 12. Resumen de actividad de la aplicación

Figura 13. Visualización de hospitales, comisarías y parques de bomberos

seleccionando los edificios que deseamos visualizar en otra pestaña anexa a la anterior (figura 13). A continuación se describen los dos principales componentes del cliente web: el motor Ajax y la API de Google Maps.

El motor Ajax

Para hacer nuestro cliente web funcional introdujimos en él un motor Ajax, con el fin de conseguir una mayor agilidad en la interacción del usuario con la aplicación. El funcionamiento de Ajax se basa en que la primera vez que se visualiza la página se descarga en el cliente un motor que controlará la aplicación y a partir de este momento, la aplicación sólo hará peticiones al servidor para pedir datos, no contenidos, de forma que el encargado de la visualización de la página será el cliente y no el servidor, reduciéndose así el tráfico de datos. Las peticiones, además, serán asíncronas, es decir, tranparentes al usuario que ya no sigue un esquema de funcionamiento petición - espera - visualización. La página nunca se recarga por completo y el usuario simplemente percibe que la información visualizada va cambiando a medida que interactúa con la aplicación como si se tratara de una aplicación de escritorio. Para facilitar la creación del motor Ajax hemos utilizado JQuery [7], un conjunto de librerías que simplifican el uso de JavaScript y además son multinavegador, lo que nos proporciona una capa de abstracción y nos garantiza que nuestro motor Ajax funcionará por igual en todos los navegadores.

El motor Ajax es descargado la primera vez que se accede a la aplicación y desde ese momento el motor está interactuando con el servidor sin que el usuario lo note. Cada segundo, la aplicación realiza una petición (GET) al servidor preguntándole por el estado actual de los desastres. La lógica de negocio efectúa las consultas necesarias a la base de datos y le responde con los datos que han cambiado desde la última petición en formato JSON (la cantidad de información enviada es mucho menor que si tuviéramos que actualizar la página completa). Con estos datos, el motor Ajax actualiza la presentación cambiando el código HTML, la hoja de estilos (CSS) o lo que es más común, la apariencia del mapa, para lo cual hace uso de la API proporcionada por Google Maps. Cuando un usuario añade un desastre en el mapa se produce automáticamente una petición al servidor (POST) enviando los datos introducidos sin que en ningún momento se interrumpa la visualización de la interfaz gráfica. Este esquema de interacción entre el motor Ajax, la interfaz gráfica y la lógica de negocio puede observarse en la figura 14.

API Google Maps

Para interactuar con el mapa, se ha empleado la API de Google Maps, que nos proporciona una serie de clases JavaScript con todos los métodos necesarios para ello. A las clases proporcionadas por Google Maps (GMap2, GEvent...) añadimos una propia denominada *Marcador* que contiene toda la información necesaria para crear los marcadores utilizados en nuestra aplicación. Un diagrama de clases simplificado se puede observar en la figura 15.

III-C. El cliente móvil

La penetración de teléfonos móviles a nivel mundial es muy superior a la de ordenadores con internet, y esta diferencia es aún mayor en las zonas menos desarrolladas donde esta aplicación podría tener una mayor utilidad. Además, el valor de la aplicación Desastres2.0 aumenta con el número de usuarios que tienen acceso a ella y si a esto le añadimos la

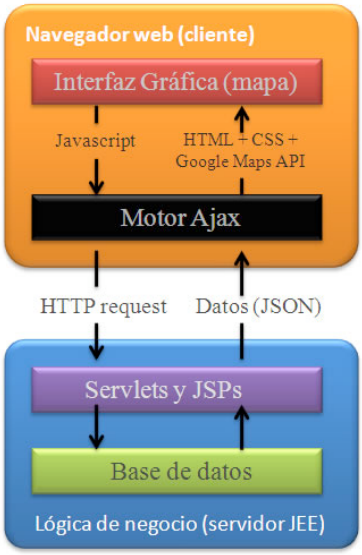

Figura 14. Arquitectura detallada de la interacción del motor Ajax con la aplicación

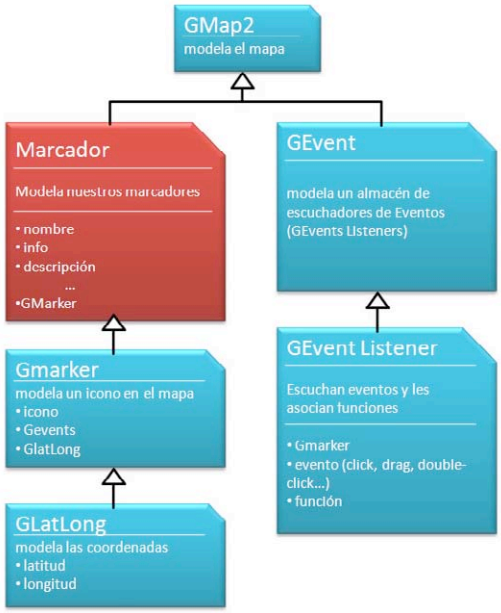

Figura 15. Diagrama de clases JavaScript utilizadas

posibilidad de avisar de un desastre en tiempo real desde la calle, nos damos cuenta de que el cliente móvil es un elemento fundamental para este proyecto.

Haciendo uso de la intefaz REST y el *framework* [2] Mojax [8] se ha desarrollado una aplicación para terminales móviles (teléfonos, PDAs...) que ofrece una adaptación de la interfaz gráfica a estos terminales pero con las mismas funcionalidades que el cliente web. La aplicación permite obtener un listado de desastres activos, víctimas y recursos, así como obtener todos los detalles sobre los mismos y visualizarlos en un mapa. También permite añadir cualquiera de estos elementos con toda la información deseada validando la dirección obtenida mediante un servicio geolocalizador así como modificarlos, eliminarlos o asociar recursos y víctimas a desastres.

Mojax es un *framework* propiedad de mFoundry creado para desarrollar aplicaciones móviles utilizando *mobile Ajax*. Adapta los conceptos de XML, CSS y JavaScript a terminales móviles y los combina con J2ME. El desarrollador sólo tiene que escribir la aplicación usando los estándares de Ajax y ésta es convertida por Mojax en una aplicación J2ME adaptada al teléfono móvil desde el que se ha hecho la petición. Gracias al objeto HTTPRequest implementado por Mojax, se pueden hacer peticiones asíncronas a la lógica de negocio de la aplicación Desastres2.0 mediante la interfaz REST. Los datos en formato JSON devueltos por ésta son leídos por la aplicación móvil y presentados en consecuencia. A continuación se muestran diversas capturas de pantalla de un terminal móvil donde se puede observar la pantalla principal (figura 16), un listado de desastres (figura 17), o la presentación en un mapa de un desastre (figura 18). Podemos destacar que en el cliente móvil se ha utilizado el servicio de mapas de Yahoo en lugar de el de Google como se hizo en el cliente web. Esto es debido a que la compatibilidad de Javascript en terminales móviles es aún bastante reducida, y las APIs JavaScript de mapas (tanto de Google como de Yahoo) no eran compatibles con Mojax. Sin embargo, el servicio de Yahoo Maps nos ofrece una API por la cual nos envía imágenes en formato JPEG de un mapa a partir de una dirección o unas coordenadas. Utilizando este servicio, el cliente móvil recibe (mediante peticiones asíncronas) las imágenes que necesite y permite desplazarse por él o realizar zoom a la zona que deseemos.

III-D. Otros clientes

El servicio ofrecido mediante la API Rest hace que nuestra aplicación sea fácilmente utilizable y extendible a muchos otros clientes. Los datos de nuestra aplicación podrían ser consumidos por otras aplicaciones (*mashups*). Utilizando este servicio se ha desarrollado el cliente móvil y diversas aplicaciones de inteligencia artificial que se desarrollarán en la sección IV.

IV. Aplicación de Sistemas Inteligentes

Este trabajo ha explorado el uso de técnicas inteligentes en la aplicación Desastres 2.0, con el fin de mejorar la asignación y coordinación de los recursos en situaciones de desastres.

IV-A. Planificador basado en reglas

Utilizando como escenario la aplicación Desastres 2.0 hemos desarrollado un sistema experto que planifica los recursos a asignar a una serie de desastres a partir de los recursos disponibles en una base de datos. Este sistema experto está escrito en Jess [9], un motor de reglas para Java que nos permite

[2] *Framework: Estructura de soporte software definida que facilita el desarrollo y organización de otros proyectos software. Típicamente se compone de programas, bibliotecas y un lenguaje interpretado.*

Figura 16. Menú principal de la aplicación móvil

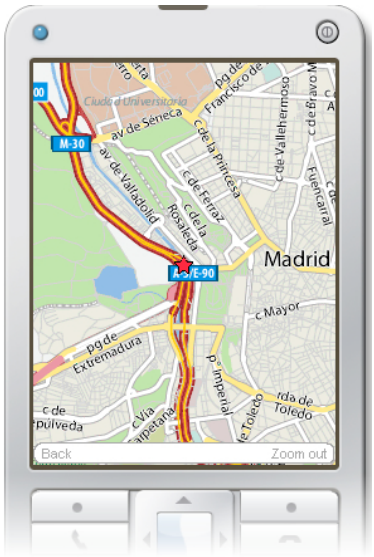

Figura 18. Visualización de elementos en un mapa

Figura 17. Pantalla con el listado de desastres activos

Figura 19. Proceso del sistema experto

crear reglas de alta complejidad aplicando un sistema de coincidencia de patrones.

El sistema experto actualiza su base de conocimiento mediante una petición REST de los desastres activos y de los recursos disponibles (policías, bomberos y ambulancias que no están ocupados), y asigna a cada desastre los recursos adecuados (policías, bomberos y ambulancias) siguiendo unas reglas en función del tipo de desastre, su magnitud, el número de heridos y su gravedad o la densidad de tráfico asociada. La figura 19 muestra el esquema seguido para la integración del sistema experto con la arquitectura REST de Desastres 2.0. Este sistema experto trabaja en tiempo real, de forma que a medida que los desastres se vayan modificando (solucionándose o agravándose) se irán realizando nuevas asignaciones liberando o llamando a otros recursos. En caso de no disponer de todos los recursos suficientes para responder a un desastre, el sistema envía todos los que pueda y a medida que se vayan liberando en otros desastres irán siendo asignados a los siguientes desastres priorizando según la necesidad de cada uno.

IV-B. Sistema multiagente de gestión de desastres

Aprovechando el servicio de la aplicación Desastres 2.0 se ha desarrollado un sistema multiagente utilizando Jadex [10]. En una primera versión, se ha dotado a cada recurso

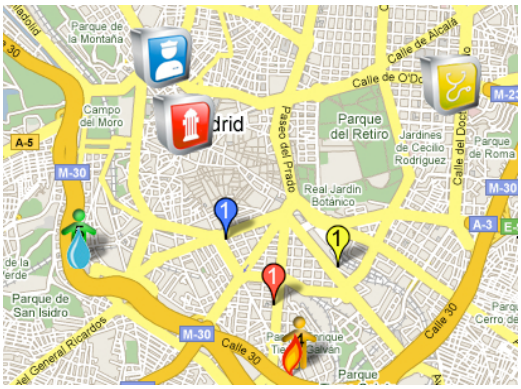

Figura 20. Sistema multiagente

(policías, bomberos y ambulancias) de autonomía propia de manera que cada uno puede decidir cómo actuar en situaciones de emergencia. La segunda versión desarrolla un sistema con varios niveles de coordinación y una estructura jerárquica en que un servicio (112) coordina las acciones de los diferentes recursos, con el fin de cubrir todos los desastres y evitar colapsar las vías de comunicación.

Este sistema se comunica con la interfaz gráfica del cliente web y permite visualizar cómo los agentes se desplazan hacia los desastres asignados, recogen a los heridos y solucionan los desastres, como puede observarse en la figura 20.

V. Trabajos relacionados

En el área de desastres y tecnologías web 2.0, podemos citar el proyecto de fuegos activos de San Diego [11] que permite visualizar en un mapa los fuegos activos en San Diego. La principal diferencia de este proyecto es que es un *mashup* de información que se agrega y visualiza, pero no permite la participación social o la gestión interactiva de los recursos ni ofrece una API REST para integrar aplicaciones externas.

Los sistemas multiagente han sido aplicados a la simulación [12] y gestión de desastres [13]. Este trabajo facilita su aplicación y experimentación, ya que uno de los principales problemas es la integración con el sistema de información geográfica.

VI. Conclusiones y Trabajos Futuros

En este trabajo de investigación se ha presentado una plataforma basada en tecnologías web2.0 para la gestión de desastres que facilita la compartición de información y la colaboración. El trabajo ha definido la arquitectura REST seguida que facilita que esta aplicación pueda ser integrada por terceros, y combinada con otra información mediante *mashups*.

Finalmente, esta plataforma ha sido completada con un cliente móvil y con la integración de una plataforma de agentes sobre la plataforma Jadex.

La combinación de tecnología de agentes con tecnología web2.0 está siendo una línea de investigación muy prometedora, tanto para la experimentación como para docencia de tecnologías multiagente.

Agradecimientos

Este trabajo de investigación ha sido cofinanciado por el Ministerio de Educación en el proyecto TSI Improvisa (TSI2005-07384-C03-01).

Referencias

[1] Secretariat of the International Strategy for Disaster Reduction / United Nations, "Lessons for a safer future: Drawing on the experience of the indian ocean tsunami disaster," International Strategy for Disaster Reduction (ISDR), Tech. Rep., 2007.

[2] T. O'Reilly, "What is web 2.0: Design patterns and business models for the next generation of software," *O'Reilly Media*, September 2005. [Online]. Available: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[3] R. T. Fielding, "REST: architectural styles and the design of network-based software architectures," Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm

[4] Noelios Consulting, "Restlet, Lightweight REST framework for Java," February 2008, homepage of the Restlet project. [Online]. Available: http://www.restlet.org

[5] Google Inc., "Google Maps," February 2008, maps service provided by Google. [Online]. Available: http://www.maps.google.com

[6] JSON Project, "Introducin JSON," homepage of JSON project. [Online]. Available: http://www.json.org

[7] K. Swedberg and J. Chaffer, *Learning jQuery : Better Interaction Design and Web Development with Simple JavaScript Techniques*. Packt Publishing, 2007.

[8] mFoundry Inc., "Mojax, framework for mobile ajax," official website of Mojax. [Online]. Available: http://mojax.mfoundry.com

[9] Sandia National Labs, "Jess, the rule engine for the java platform," official website of Jess Project. [Online]. Available: http://www.jessrules.com

[10] A. Pokahr, L. Braubach, and W. Lamersdorf, "Jadex: A bdi reasoning engine." in *Multi-Agent Programming*, ser. Multiagent Systems, Artificial Societies, and Simulated Organizations, R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, Eds. Springer, 2005, vol. 15, pp. 149–174. [Online]. Available: http://dblp.uni-trier.de/db/books/collections/map2005.htmlPokahrBL05

[11] C. Rush, "San diego county fires," mashup on Google Maps about disasters. [Online]. Available: http://maps.google.com/maps/ms?msa=0msid=114250687465160386813.00043d08ac31fe3357571

[12] Y. Nakajima, H. Shiina, S. Yamane, T. Ishida, and H. Yamaki, "Disaster evacuation guide: Using a massively multiagent server and gps mobile phones." in *SAINT*. IEEE Computer Society, 2007, p. 2. [Online]. Available: http://dblp.uni-trier.de/db/conf/saint/saint2007.htmlNakajimaSYIY07

[13] J. R. Velasco, A. López-Carmona, M. Sedano, M. Garijo, D. Larrabeiti, and M. Calderón, "Role of multi-agent system on minimalist infrastructure for service provisioning in ad-hoc networks for emergencies," 2006, pp. 151–152, first International Workshop on Agent Technology for Disaster Management AAMAS06.