# Disasters2.0. Application of Web2.0 technologies in emergency situations

**Julio Camarero Puras**
Dpto. Ingeniería de Sistemas Telemáticos
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Email: jcp@gsi.dit.upm.es

**Carlos A. Iglesias**
Dpto. Ingeniería de Sistemas Telemáticos
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Email: cif@gsi.dit.upm.es

## ABSTRACT

This article presents a social approach for disaster management, based on a social portal, so-called Disasters2.0, which provides facilities for integrating and sharing user generated information about disasters. The architecture of Disasters2.0 is designed following *REST* principles and integrates external mashups, such as Google Maps. This architecture has been consumed with different clients, including a mobile client, a multiagent system for assisting in the decentralized management of disasters, and an expert system for automatic assignment 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.

### Keywords

Disasters, Web2.0, Mobile, mashup, REST, Intelligent techniques.

## INTRODUCTION

Natural disasters are associated to chaotic situations in which information is usually incomplete and imprecise. This lack of information complicates the efficient management of catastrophes and makes the decision making process a difficult task. According to the Secretariat of the International Strategy for Disaster reductions of the United Nations (*UN/ISDR*), within the eleven lessons learnt for disaster management, the first two are [1]:

> *Public awareness is an essential element of preparedness for saving lives and livelihoods.*
>
> *Individuals and communities play important roles in managing risks from natural hazards.*

This article proposes that Web2.0 technologies can be a valuable tool to contribute to address both lessons, achieving public knowledge of new disasters and both individual and social participation in disaster management. Web2.0 [2] has proven the power of users' participation to create content, give opinions and organize themselves in social networks. Examples such as Wikipedia or del.icio.us have shown us the potential of this *collective intelligence* [3] when it is used well.

Thus, this article proposes that managing natural disasters can be a potential application of this collective intelligence. If anybody could report in real time the place where a disaster is happening, its magnitude or its monitoring, then the response could be much more effective and immediate.

This project has been developed within the Spanish national research project Improvisa [4] (TSI2005-07384-C03-01), whose aim is to provide information services in natural disaster scenarios by using ad-hoc networks, service-oriented architectures and intelligent agents. Web2.0 applications have been investigated in order to provide information services and manage coordination among people and intelligent agents to handle alarms.

The rest of this article is structured as follows. First, it is presented how disasters can be managed. Next, the architecture of the Disasters2.0 system is shown, describing all its components. The next section explains the application of intelligent techniques to the system. Finally, related works and conclusions are summed up as well as the future work.

## MODELING DISASTERS AS REST RESOURCES

*This section presents one of the main contributions of the article, which consists of modeling disasters as REST resources, providing an open and light interface for its management.*

*REST [5]*, which stands for *Representational State Transfer*, is an architecture based on the client-server model used for web applications which defines a way to represent, access and modify data in the Internet. *REST* understands all data as **resources** and makes them available through URIs (Uniform Resource Identifiers). This architecture is based in four principles:

- Application state and functionality are abstracted into resources
- Every resource is uniquely addressable using a universal syntax for use in hypermedia links (URI)
- A stateless client-server protocol: *HTTP* (Hypertext Transfer Protocol)
- All resources share a uniform interface for the transfer of state between client and resource, consisting of a constrained set of well-defined operations (*HTTP* Methods: *GET, POST, PUT* and *DELETE*) and a constrained set of content types (typically *HTML* or *XML*)

The system presented in this article, Disasters 2.0, has been designed according to these *REST* principles. The main entities of a disaster have been modeled as resources. The system has considered the following entities: events such as fire or flood, allocated resources such as policemen or firemen, and damages such as victims. These resources are accessible through a *REST* interface using standard *HTTP* methods. For example, we could type in our browser:

- */events* to obtain a list of events
- */id/10* to access the event with identifier 10

The *REST* interface for disasters proposed in this article is illustrated in Table 1.

| URL | Resource | Example |
|---|---|---|
| *GET Method* | | |
| /id/{id} | Disaster, resource or victim with identifier {id} | /id/5 |
| /date/YYYY/MM/DD | Disaster, resource or victim added after this date | /date/2002/05/15 |
| /events | All the disasters (events) | /events |
| /events/{type} | All the disasters of a type (fire, flood or collapse) | /events/fire |
| /events/date/YYYY/MM/DD | Disasters from this date | events/date/2007/05/15 |
| /resources | All the resources | /resources |
| /resources/{type} | All the resources of a type (policemen, firemen o ambulances) | /resources/police<br>/resources/firemen |
| /resources/date/YYYY/MM/DD | Resources added from this date | resources/date/2007/05/15 |
| /people | All the victims (people) | /people |
| /people/{type} | All the victims of a kind (slightly injured, seriously injured, dead o trapped) | /people/dead |
| /people/date/YYYY/MM/DD | Victims from this date | people/date/2007/05/15 |
| *DELETE Method* | | |
| /delete/id/{id} | Deletes the entity with identifier id | /delete/id/2 |
| *PUT Method* | | |
| /put/{id}/{parameter}/{value} | Sets new value {value} to the parameter {parameter}of entity {id} | /put/3/latitud/1.4005 |
| /put/id/add<br>/put/id/remove | Increase/decrease in one unit a marker of resources or victims | /put/2/add<br>/put/8/remove |

| *POST Method* | | |
|---|---|---|
| /post/{parameters} | Creates a new marker with all the parameters sent. (If parameters include an address and do not include latitude and longitude, this address is validated and assigned the corresponding latitude and longitude parameters. | /post/type=fire &name=Fire &info=devastated area &latitud=…. |

**Table 1. *REST* interface for accessing disasters**

## ARCHITECTURE

*This section shows the innovative architectural aspects of the system. The main components of this architecture (the server, the web client and the mobile client) are explained in more detail.*

In order to provide *REST* resources, the system Disasters 2.0 follows a client-server architecture as shown in Figure 1. As it can be seen in the figure, the business logic of the server manages all the information across the platform and the *REST* API allows any application to interact with this business logic. The web client is connected to the business logic using an Ajax Engine which will be explained in this section and uses the mashup Google Maps to display the information. The rest of the clients will be presented in the following sections.

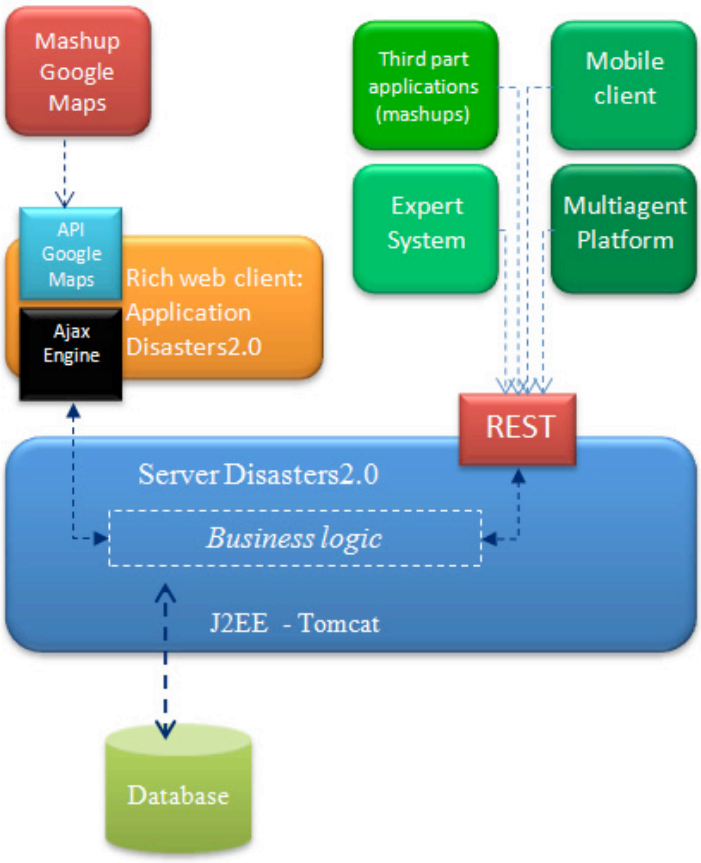


**Figure 1. Architecture of Disasters2.0**

### Server Disasters2.0

The server of our application has been developed with Java Enterprise Edition technologies (Servlets and JSPs) and runs on Apache Tomcat. The server is responsible for:

- Saving persistently all the information about disasters in a database
- Implementing the business logic to update and recover that information
- Serving the information through *REST* services to any client

The framework Restlet [6] has been used to design the *REST* architecture. Restlet is an open source lightweight framework for Java developed by Noelios Consulting. This framework is based on creating virtual routers which redirect *HTTP* requests to the business logic based on URL patterns. An example of this architecture is shown in Figure 2. In this figure, three different URLs can be seen at the bottom. Each of them will be redirected to a different *"restlet" (piece of business logic)* based on the structure of the URL.

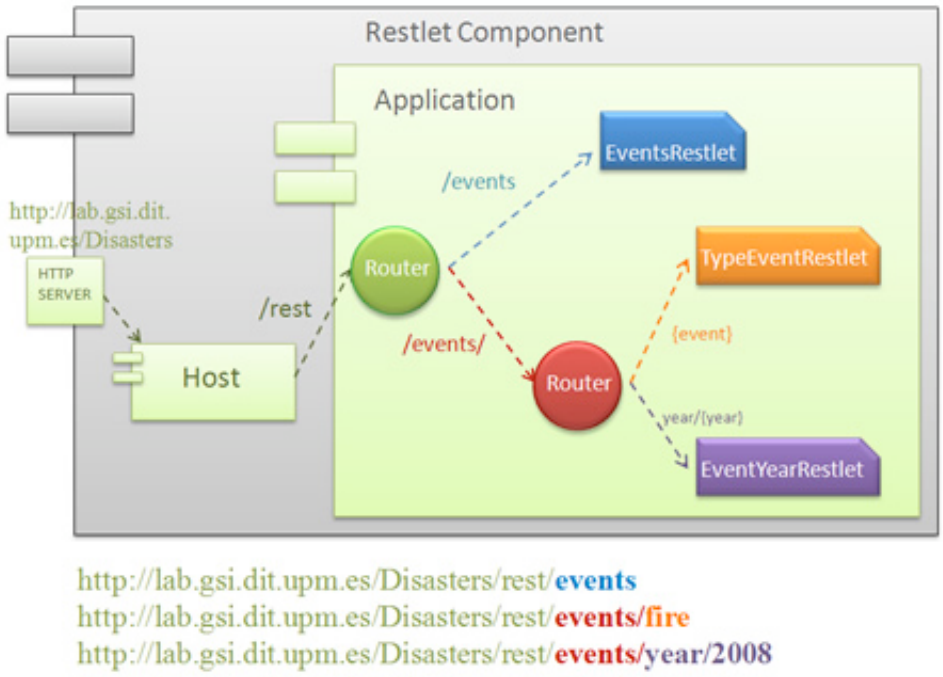


**Figure 2. Pattern-based processing of *REST* requests**

The server provides the resources following the *JSON* [7] format, although *XML* format data (Extensible Markup Language) can also be retrieved by adding the word "xml" at the beginning of the requested URI (e.g. http://lab.gsi.dit.upms.es/Disasters/rest/**xml**/events).

*JSON*, which stands for *JavaScript Object Notation*, is a lightweight computer data interchange format. We used *JSON* instead of *XML* for several reasons [8]:

- *JSON* is processed more easily and faster than *XML* because its structure is simpler.
- *JSON* is a better data exchange format (*XML* is a better document exchange format) and is already integrated in JavaScript.
- Data are smaller using *JSON* than *XML*

The next two figures show an example of data belonging to a disaster using both *JSON* and *XML* languages where we can see that the number of characters using *JSON* is smaller.

```
{ “disaster”: {
        “id”: “1”,
        “name”: “fire”,
        “victims”: {
                “trapped”: “10”,
                “slight”: “0”
        }
    }
}
```

**Figure 3. Example using *JSON***

```
<disaster>
        <id>1</id>
        <name>fire</name>
        <victims>
                <trapped>10</trapped>
                <slight>0</slight>
        </victims>
</disaster>
```

**Figure 4. Example using *XML***

### Web Client Disasters2.0

The main element in our web application is the map provided by Google Maps as shown in Figure 5.

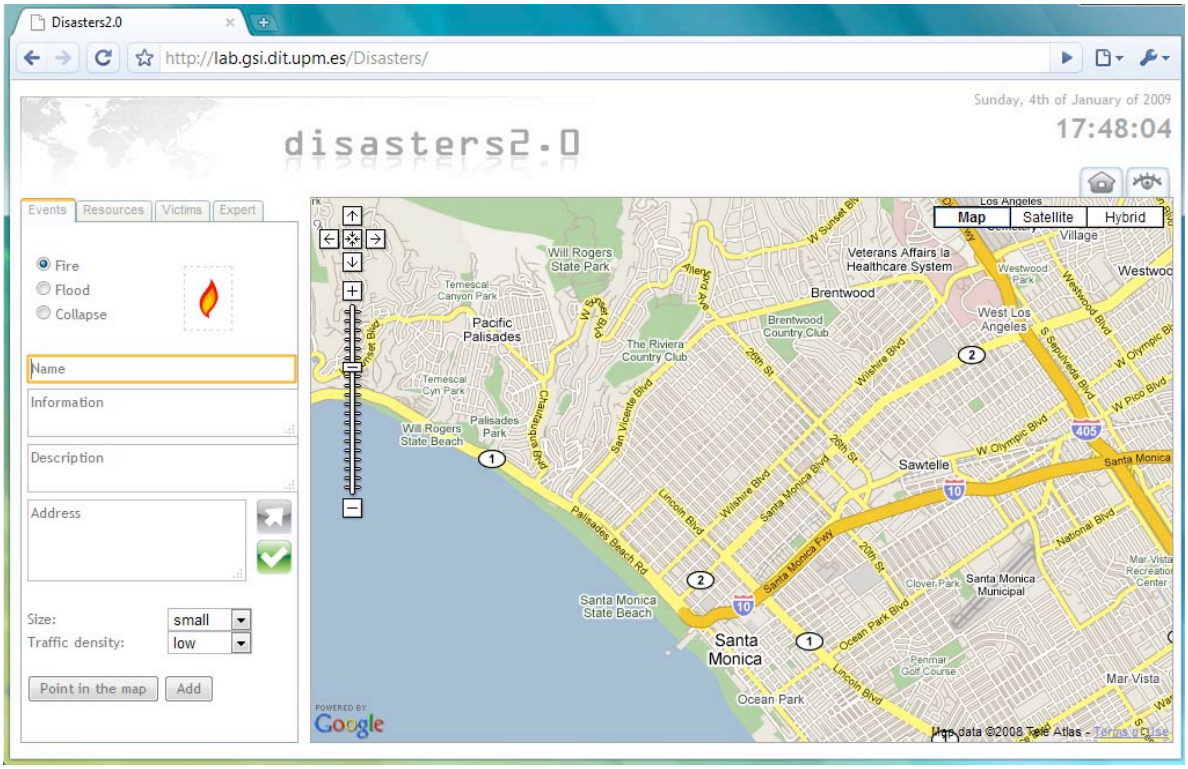


**Figure 5. User interface of the web client**

The web application provides three forms organized by tabs which enable to add any kind or marker to our map:

- Events: fires, floods and collapses
- Resources: policemen, firemen and ambulances
- Victims (people): trapped, slightly injured, seriously injured and dead

Every marker can have information associated such as name, description, additional information and location.

Events may also have some information related to the disaster, such as the magnitude of the disaster, the current state, and the traffic around the affected area. This information can be very useful in order to coordinate an adequate response. Resources and victims may also indicate the number of units or victims. Marker icons will be different depending on the current state or the number of units as shown in Figure 6.

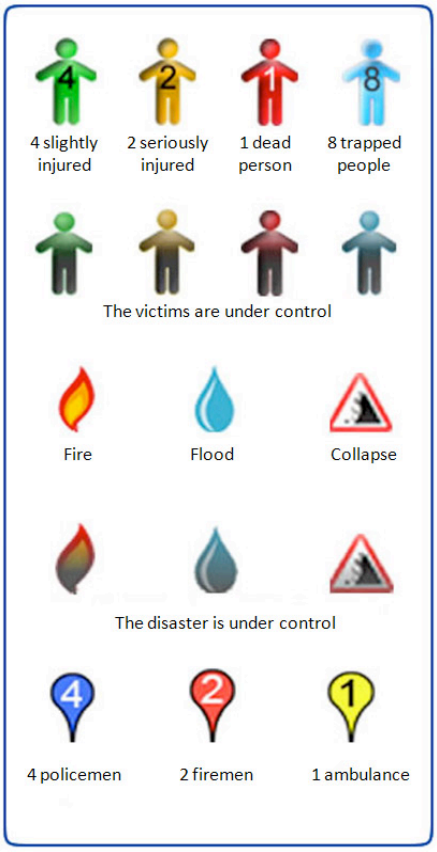


**Figure 6. Markers used in the application**

The application leverages Google Maps user interface to provide the user a seamless interaction in a very intuitive way. Figure 7 and Figure 8 show details of this friendly user interface.

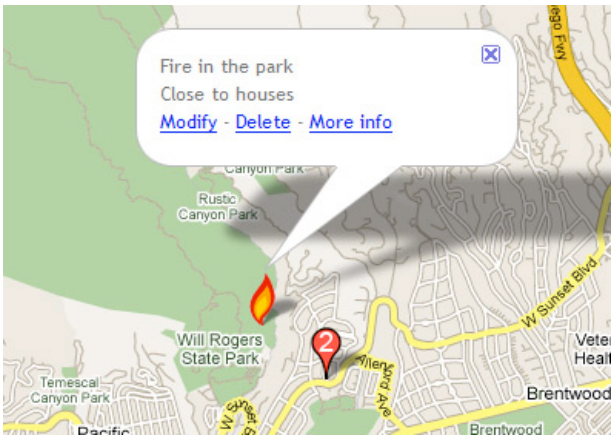


**Figure 7. Information for a marker**

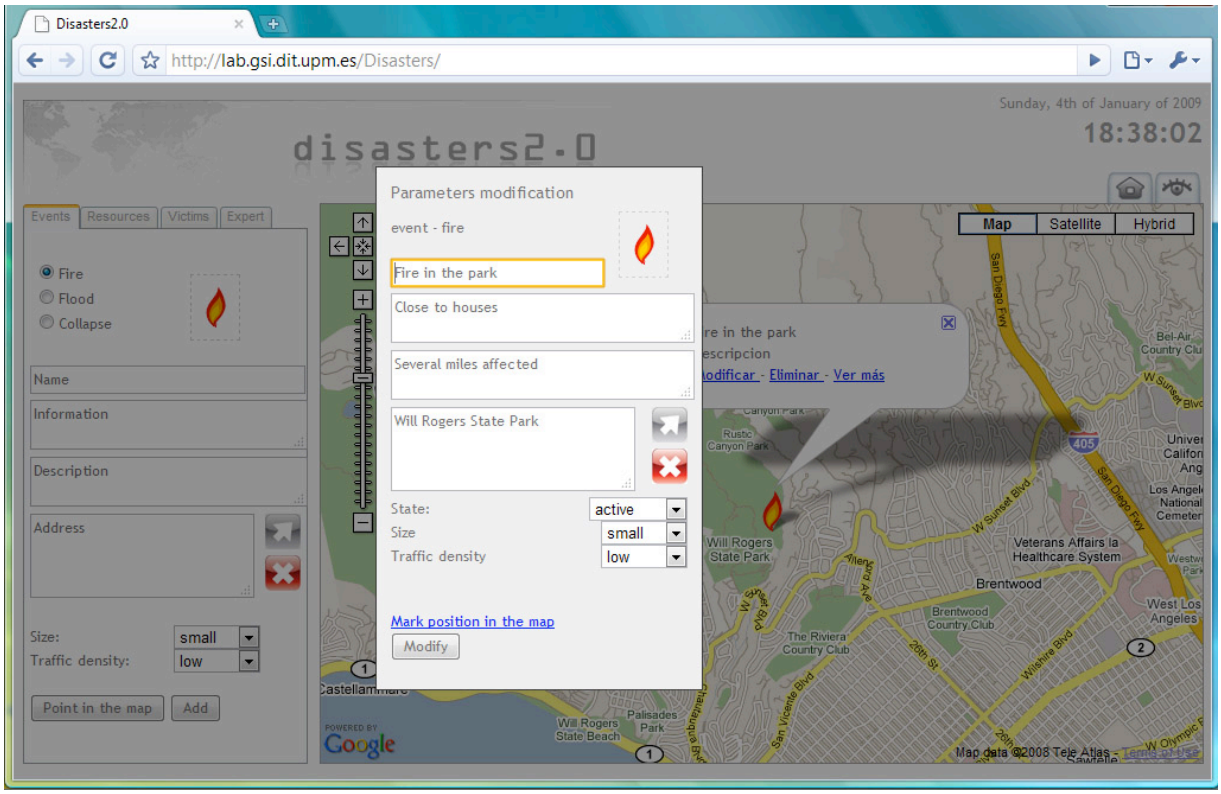


**Figure 8. Modifying a marker**

The system provides a *validation* functionality using Google's geolocator service for every address introduced.

The client implements a *visual association* functionality. Both resources and victims can be associated to an event (victims from a disaster and resources aimed to fix it) using a drag-and-drop system as shown in Figure 9.

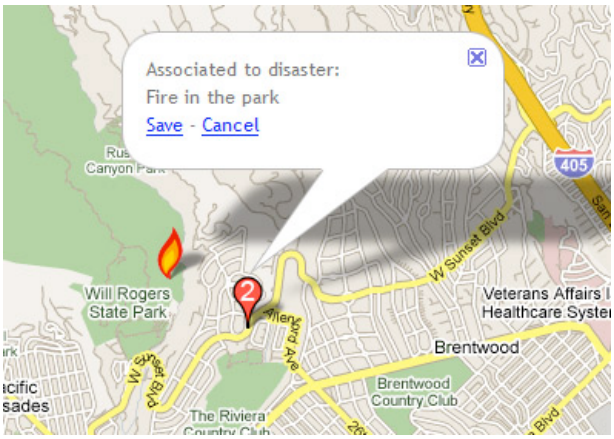


**Figure 9. Associating firemen to a fire**

The web client also offers real time summaries of activity (Figure 10) and can display important buildings such as hospitals, police stations and fire stations if selected (Figure 11).

Next, the main components of the web client will be explained: the Ajax engine and Google Maps API.

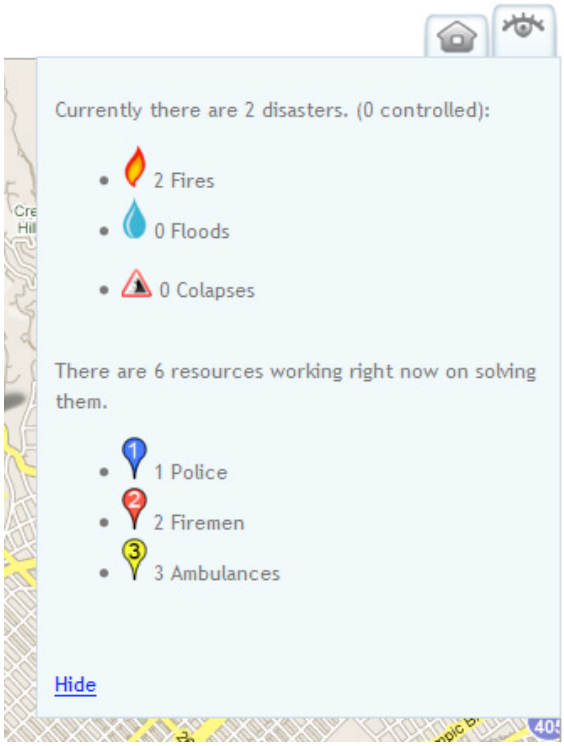


**Figure 10. Summary of activities in the application**

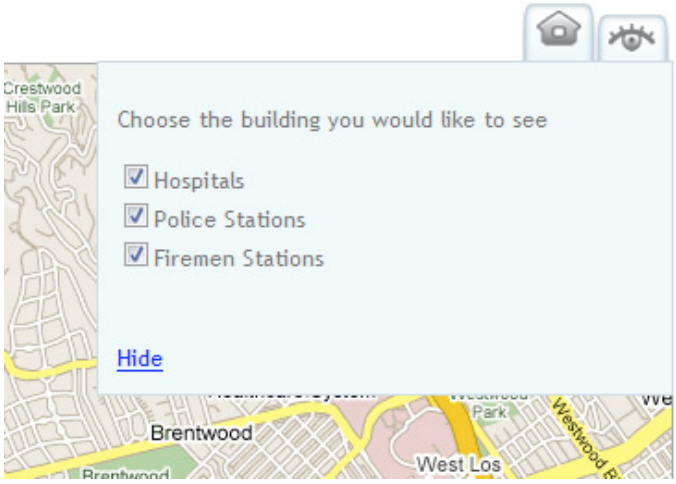


**Figure 11. Showing buildings in the map**

*Ajax Engine*

In order to make our web client more functional, we introduced an Ajax engine, which improves the interaction of the user with the application. An Ajax engine works the following way: the first time a client accesses the web application, the engine is downloaded and from this moment, this engine will only request data (not content anymore) to the server. Therefore the client and not the server will be the responsible for the presentation, reducing the data traffic considerably. In addition, the *HTTP* requests will be asynchronous, which means they will be transparent to the user, who does not have to wait as in the traditional web applications model: request – wait – response. The whole page is never reloaded, just the updated information is sent. The user just feels the application is changing while he interacts with it as if it was a desktop application. In order to ease the creation of the Ajax engine, we have used JQuery [9], a set of JavaScript multibrowser libraries which simplifies the use of JavaScript and ensures that our Ajax engine will work in all the modern browsers.

Our Ajax engine is downloaded the first time a client accesses the web application and from that moment, this engine is communicating with the server in a way the user does not perceive it. Every second, the web application makes a request (*GET*) to the server asking for the current state of disasters. The business logic will make all the necessary queries to the database and as a result the server will respond with the data that have changed from the last request in *JSON* format (this way, the data sent from the server is much smaller than if we always send the whole page). Using the received data, the web client updates the presentation changing the DOM tree, the style sheets (*CSS*) or what is more common, the appearance of the map using Google Maps API. When a user introduces a new marker in the map, a new *HTTP* request (*POST*) is automatically sent to the server while the user interface is never interrupted, so the user does not have to wait for the page to reload. The diagram of this interaction among the Ajax engine, the user interface and the business logic is shown in Figure 12.

*Google Maps API*

The communication between the web application and the map is achieved using Google Maps. Google Maps [10] is a free web mapping service application provided by Google that powers many map-based services including the Google Maps website, a Geolocator, satellite pictures of the whole world, directions between two addresses or embedded maps on third-party websites via the Google Maps API. Google Maps provides our web client with a friendly user interface which allows users to mark disasters in a map or drag and drop resources (policemen, ambulances or firemen) from one point to another.

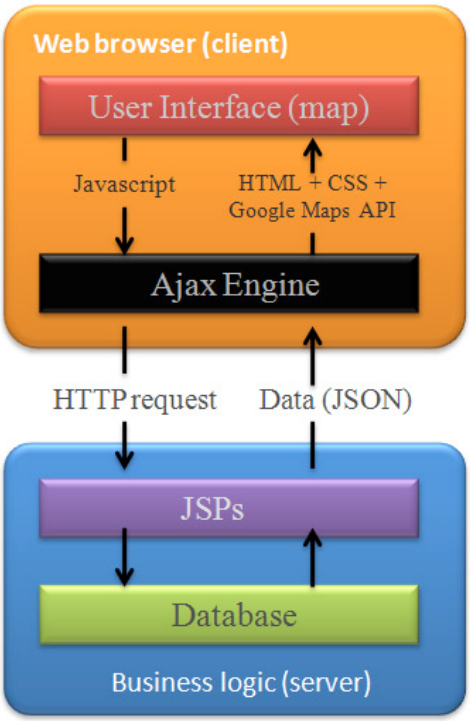


**Figure 12. Web client interaction with the server**

Google Maps API provides several JavaScript classes to work with the map (GMap2, GEvent, GlatLong…). This API has been extended with a new class called Marker, which contains the Google Maps GMarker class and all the information needed for the markers in our application. A simplified classes diagram is shown in Figure 13.

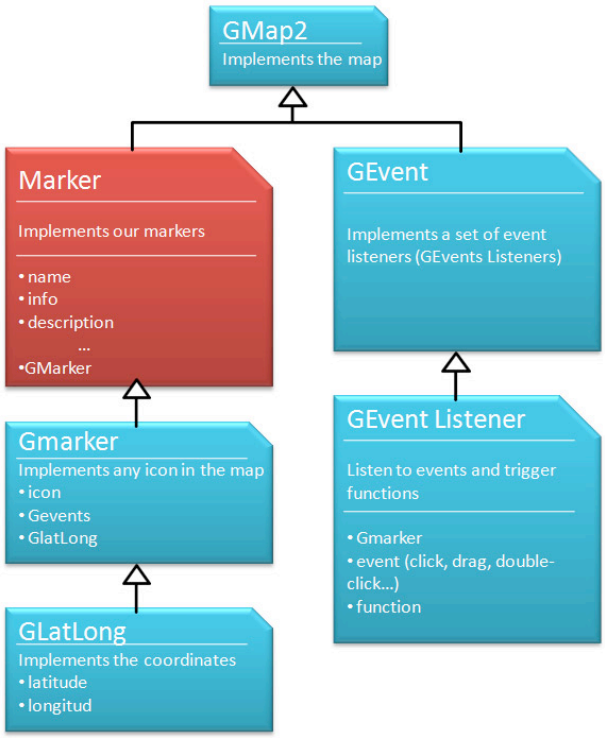


**Figure 13. JavaScript classes diagram**

**Mobile Client**

Mobile phone penetration rate is far higher than computers with Internet access, and this difference is even bigger in underdeveloped areas, in which this application could be really useful. In addition, the value of Disasters2.0 increases as the number of users increases and the possibility of alerting disasters in real time from anywhere through a mobile phone makes a mobile client almost mandatory.

Using the *REST* interface and the framework Mojax [11], an application for mobile devices (phones, PDAs…) has been developed as a client of the system Disasters2.0. This mobile client offers an adaptation of the web client to the small screen of these devices. The mobile client allows browsing between lists with all the active disasters, victims and resources and also provides all the information related to any of these markers including positioning them in a map. The application also allows users to add any of these elements to the server with all the information associated as in the web client including the validation of the address using a geolocator service, as well as updating or deleting markers.

*Mojax* is a free framework developed by *mFoundry* aimed to develop mobile applications using *mobile Ajax*. This framework allows the user to develop an application using Ajax standards (*XML*, *CSS* and JavaScript) and when the application is downloaded from a mobile device, it is converted to a J2ME application adapted to the device from which it has been downloaded. Using the *HTTPRequest* object implemented by Mojax, we can make asynchronous calls from the mobile application to the business logic of Disasters2.0 through the *REST* interface. *JSON* data are responded from the server and the mobile application changes its appearance in consequence. Below there are some screenshots from a mobile device in which we can find the main screen (Figure 14), a list of disasters (Figure 15) and the location of a disaster in a map (Figure 16).


**Figure 14. Main screen of the mobile application**


**Figure 15. Screen with list of disasters**

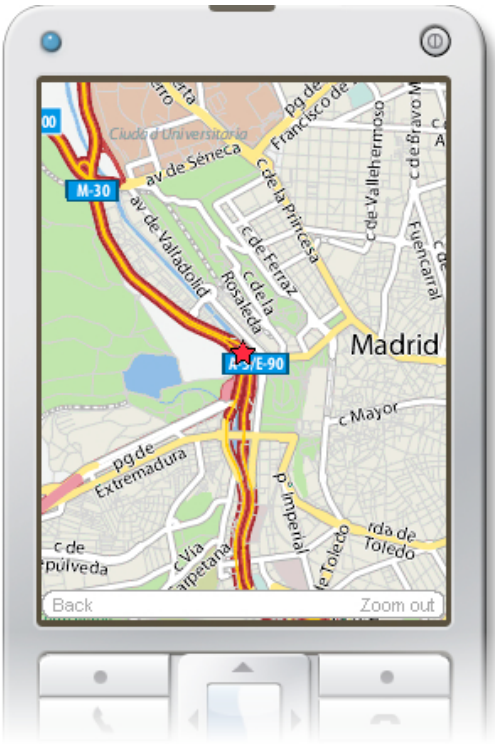


**Figure 16. Showing disasters in a map**

In the mobile client, the Yahoo Maps [12] service has been used instead of the Google Maps one as it was done in the web client. The reason for this is that JavaScript compatibility in mobile devices is still very reduced and JavaScript APIs for Google Maps or Yahoo Maps were not compatible with Mojax. However, Yahoo Maps offers another service consisting on serving map images in JPEG format from a request indicating several parameters such as latitude, longitude, zoom or the address. Using this service, our mobile client requests the images needed (making asynchronous calls) and allows the user to move around or zoom in and out the map.

## APPLICATION OF INTELLIGENT TECHNIQUES

*This project has researched the application of intelligent techniques in the system Disasters2.0 aiming to improve the assignment and coordination of resources in disaster situations. This section presents ongoing research on the application of these techniques in this architecture.*

Leveraging the service offered by the *REST* API, Disasters2.0 can be integrated with external clients in a seamless way, so that they are able to mashup the consumed disaster information. Two external clients have been developed to research an intelligent system approach to Dissasters2.0: a planner based on rules and a multiagent platform.

### Planner Based on Rules

Using the scenario of Disasters2.0 we have developed an expert system which assigns free resources available in a database (policemen, firemen and ambulances) to active events (fires, floods and collapses). This expert system has been developed using Jess [13], a rule engine for Java which allows the creation of high complexity rules applying a pattern matching system.

The business logic of the server keeps updated a database with all the resources and useful information about them such as their current location, their unique identifier or if they are busy at the moment. This expert system updates its knowledge base making requests to the *REST* interface of Disasters2.0 to obtain the list of active disasters and all the resources which are not busy. Then, it tries to assign to each disaster the needed and closest resources following several rules based on the type and size of the disaster, the number and type of victims associated or the traffic around the affected area. For example, the number of policemen assigned to a disaster will be proportional to the traffic density by the affected area, the number of ambulances follows a simple

algorithm which evaluates the number of victims and their gravity and the number of firemen considers the size of the disaster, the number of victims, the number of trapped people and the possible growth of the disaster. These rules can be easily modified and are expected to be improved by emergency personnel and learning from real situations.

Figure 17 shows the behavior for integrating this expert system with the *REST* architecture of Disastres2.0.

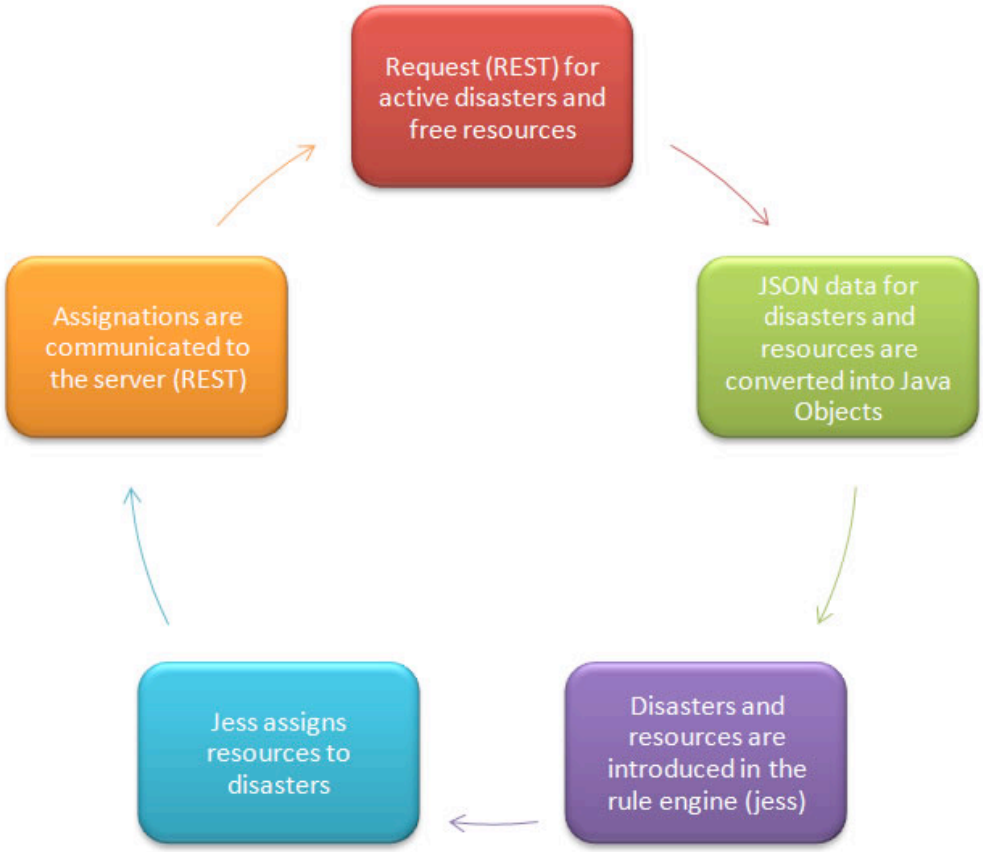


**Figure 17. Expert System process**

The expert system works in real time, which means that as soon as the disasters are updated (getting better or getting worse) or added to the application, new assignments will be made. In case the system cannot find all the resources needed to assist a disaster (as they may be busy at that moment), all the available resources will be assigned to it and as soon as busy resources are freed from any other disaster they will be assigned to the disaster which most needs them.

This expert system can help us to decide how many resources should be sent to fix active disasters (depending on rules which can be improved learning from experience) or how to distribute the resources we have in case they are not enough to cover all the disasters.

#### Multiagent System for Managing Disasters

A multiagent system has been designed in order to support the decision of managers. The system has been developed with Jadex [14]. Jadex is a Belief-Desire-Intention (BDI) reasoning engine that allows for programming intelligent software agents in XML and Java. Intelligent agents provide a technology for addressing distributed problems from an autonomous, distributed and cooperative approach.

Four agent classes have been identified, each one with their own objectives and plans to achieve them: policemen, ambulances, firemen and a central service:

- Policemen are patrolling until they know of a disaster. When there are disasters they will go there and stay until the disaster is resolved.

- Ambulances are located in hospitals initially. When they know of a disaster with victims they will go there and bring the victims to the hospital.
- Firemen are located in fire stations until they know of a disaster. They will go the affected area, free the trapped people and then try to resolve the situation. They are the ones who decide if a disaster has been resolved.
- Central service. Coordinates the previous agents.

In a first version of the platform, every resource (firemen, policemen and ambulances) can decide on its own how to act in an emergency situation. In this scenario, every resource asks Disasters2.0 periodically through the REST interface about the active disasters and after that they try to solve them all one by one in order of gravity. All the interactions with the disasters and the location of the agents are sent back to Disasters2.0 and therefore shown in the map.

A second version of the system develops several levels of coordination and a hierarchical structure in which an emergency central service coordinates the resources in order to cover all the disasters and avoid collapsing roads or small areas. In this scenario the central service is the only agent asking periodically about the state of the disasters and the rest of the agents send periodically information about their location so that they can be visualized in the map.

The web client allows the user to see the agents moving to the assigned disasters, taking the victims to the closest hospital and resolving the disaster as shown in Figure 18.

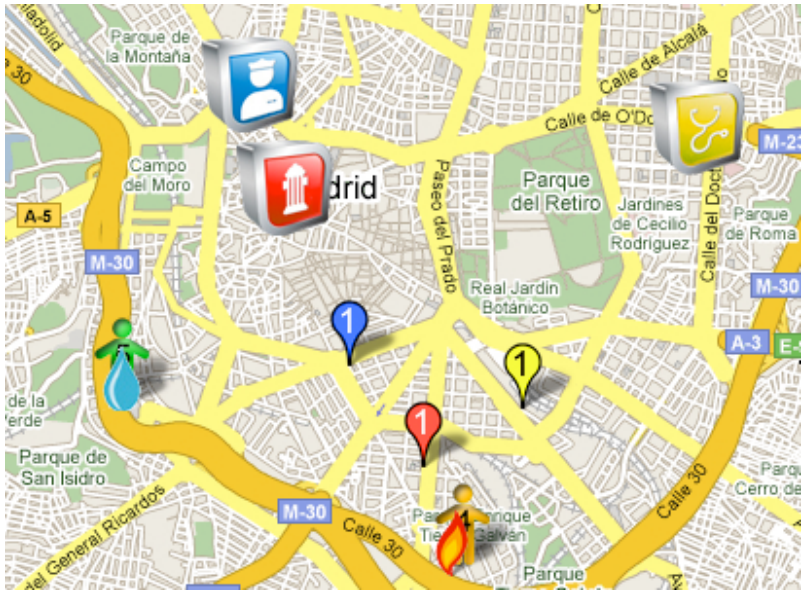


**Figure 18. Multiagent Platform**

## RELATED WORKS

Regarding the utilization of web2.0 technologies for disasters information, the “San Diego Wildfires” [15] project can be mentioned as it shows all the active fires in San Diego in a map. The main novel of our project is that it provides a *REST* API so that disasters2.0 can be integrated and used as a mashup by other applications. In addition, “San Diego Wildfires” does not allow social participation or interactive management of the disasters.

Multiagent systems have already been utilized for simulating [16] and managing [17] disaster situations. This project provides an easy way to apply and test the intelligent agents because one of the main problems in this area is the integration of the multiagent platform with the geographic information system.

## CONCLUSION

In this research project we have presented a system based in Web2.0 technologies aimed to ease disaster management by enabling collaboration and sharing information among citizens.

This article has defined the *REST* architecture followed to allow this system to be extended and integrated in other applications as a mashup.

The system, which is available as an open source project [18], provides a resource oriented interface for disasters. This interface has been validated with different clients, such as a social web application, a mobile client, an expert system and a multiagent system.

Currently, the system is being evaluated by local authorities in order to analyze its integration with a real system. In addition, initial work is being done on the implementation of a disaster simulator in order to provide researchers a benchmark for disaster management.

## ACKNOWLEDGEMENTS

This research project has been co-funded by the Spanish Education Ministry in the project TSI Improvisa (TSI2005-07384-C03-01).

## REFERENCES

1. Secretariat of the International Strategy for Disaster Reduction / United Nations. (2007) “Lessons for a safer future: Drawing on the experience of the Indian Ocean tsunami disaster,” International Strategy for Disaster Reduction (ISDR), Tech. Rep.
2. T. O’Reilly (2005), “What is web 2.0: Design patterns and business models for the next generation of software,” O’Reilly Media. [Online]. Available: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
3. P. Levy (1997) “Collective intelligence: mankind's emerging world in cyberspace”, Perseus Books Cambridge, MA, USA.
4. Improvisa Project, (February 2009) “Minimalist Infrastructure for providing services in ad-hoc networks”, homepage of the Improvisa project. [Online]. Available: http://www.gsi.dit.upm.es/~improvisa/
5. R. T. Fielding, (2000) “*REST*: architectural styles and the design of network-based software architectures,” Doctoral dissertation, University of California, Irvine. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
6. Noelios Consulting, (December 2008) “Restlet, Lightweight *REST* framework for Java,” homepage of the Restlet project. [Online]. Available: http://www.restlet.org
7. JSON Project, (December 20008) “Introducing JSON,” homepage of JSON project. [Online]. Available: http://www.json.org
8. B. Erfianto, A.K. Mahmood, A.S.A Rahman, (2007) “Modeling Context and Exchange Format for Context-Aware Computing” in 5th Student Conference on Research Development (SCOReD 07), Dec. 11-12, Bangi, Selangor. Pp 1-5.
9. K. Swedberg and J. Chaffer, (2007) Learning jQuery: Better Interaction Design and Web Development with Simple JavaScript Techniques. Packt Publishing
10. Google Inc., (December 2008) “Google Maps”, maps service provided by Google. [Online]. Available: http://www.maps.google.com
11. mFoundry Inc.,(December 2008) “Mojax, framework for mobile ajax,” official website of Mojax. [Online]. Available: http://mojax.mfoundry.com
12. Yahoo Inc., (December 2008) “Yahoo Maps”, maps service provided by Yahoo. [Online]. Available: http://www.maps.yahoo.com
13. Sandia National Labs, (December 2008) “Jess, the rule engine for the java platform,” official website of Jess Project. [Online]. Available: http://www.jessrules.com
14. A. Pokahr, L. Braubach, and W. Lamersdorf, (2005) “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, vol. 15, pp. 149–174. [Online]. Available: http://dblp.uni-trier.de/db/books/collections/map2005.html#PokahrBL05
15. C. Rush, (2007) “San Diego Wildfires 2007 Interactive fire map”. [Online]. Available: http://www.signonsandiego.com/firemap/
16. Y. Nakajima, H. Shiina, S. Yamane, T. Ishida, and H. Yamaki, (2007) “Disaster evacuation guide: Using a massively multiagent server and gps mobile phones.” in SAINT. IEEE Computer Society, p. 2. [Online]. Available: http://dblp.uni-trier.de/db/conf/saint/saint2007.html#NakajimaSYIY07

17. J. R. Velasco, A. López-Carmona, M. Sedano, M. Garijo, D. Larrabeiti, and M. Calderón, (2006) "Role of multi-agent system on minimalist infrastructure for service provisioning in ad-hoc networks for emergencies," *First International Workshop on Agent Technology for Disaster Management AAMAS06*. Hakodate, Japan. pp 151–152.
18. J.Camarero, C.A. Iglesias (February 2009) "Disasters2.0" website of the Disasters2.0 project. [Online]. Available: http://lab.gsi.dit.upm.es/web/disasters