WeKnowIt

Emerging, Collective Intelligence for Personal, Organisational and Social Use

FP7-215453

D7.3.2

Emergency Response Case Study Implementation

Dissemination level	Confidential
Contractual date of delivery	Month 33, 30-12-10
Actual date of delivery	25-02-11
Workpackage	WP7, Case studies
Task	T7.3.2
Type	Report
Approval Status	Draft
Version	0.9
Number of pages	149
Filename	

Abstract:

This deliverable describes the implementation of the Emergency Response Case Study Demonstrator.

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	19/11/2010	TOC and Timeplan	USFD
0.2	10/12/2010	First Integrated Version	USFD
0.3	17/12/2010	Second Integrated Version	USFD
0.4	20/12/2010	Architecture Section added	SMIND
0.5	20/12/2010	Integrated version for review	USFD
0.6	20/12/2010	Restructured and integrated version for review	USFD
0.7	23/12/2010	New version addressing reviewer's comments	USFD
0.8	30/12/2010	Final Version	USFD
0.9	31/12/2010	Final version with corrected title	USFD

Author list

Organization	Name	Contact Information	Reason
USFD	Vita Lanfranchi	v.lanfranchi@dcs.shef.ac.uk	Author
USFD	Simon Tucker	s.tucker@dcs.shef.ac.uk	Author
USFD	Gregoire Burel	g.burel@dcs.shef.ac.uk	Author
USFD	Aba-Sah Dadzie	a.dadzie@dcs.shef.ac.uk	Author
USFD	Annalisa Gentile	a.gentile@dcs.shef.ac.uk	Author
USFD	Philip Webster	p.webster@dcs.shef.ac.uk	Author
USFD	Elizabeth Cano	e.cano@dcs.shef.ac.uk	Author
SMIND	Tomek Kaczanowski	tomasz.kaczanowski@softwaremind.pl	Architecture
CERTH-ITI	Phivos Mylonas		Reviewer

Executive Summary

This document describes the implementation of the final version of the emergency response demonstrator for the WeKnowIt project. WeKnowIt is exploring the concept of Collective Intelligence – a form of intelligence derived from the cooperation of multiple layers of intelligence using a shared knowledge base. The emergency response demonstrator shows how Collective Intelligence can support individuals and organisations involved in a serious emergency.

As WeKnowIt has adopted a user-centred design approach in two iterative stages, the output of the first design-development cycle (first demonstrator version – described in Deliverable D7.3.1 [10]) was used as a basis for the second cycle of design and development.

Following the first implementation a user evaluation was performed (reported in D7.5.1 [11]) and then a series of user interviews and focus group were conducted with members of the Emergency Response Team in Sheffield and Doncaster. This coupled with internal testing and review resulted in re-assessed requirements, recommendations and new requirements, that were addressed by the final demonstrator.
In parallel with this, the intelligence layers produced by the technical workpackages were refined, incremented, and integrated in the core.
The final demonstrator integrates services from the Personal, Mass, Social and Media Intelligence layers to collect and analyse inputs and produce a Collective Intelligence output. Mobile and Web applications developed for the first demonstrator were then improved to reflect the user and developer feedback and to incorporate the new available features – these applications interact with the core system via the composition layer.

Moreover new requirements expressed by the user during the second cycle of user studies highlighted the need for post-incident management tools, to help the organisational user in the debriefing and investigation process: two new applications were developed to meet these needs, focusing on the possibility to perform advanced search and qualitative manipulation of collective intelligence and on providing guidance and support for organisational sense making.

This deliverable is organised as follows: Section 1 describes the work carried out in WP7 for the development of the final demonstrator, including details on the user studies methodology and on the focus of implementation. Section 2 introduces the recommendations inherited from the first stage development and evaluation (described in D7.5.1) and the new requirements resulting from the ongoing user studies. Section 3

introduces the Architecture of the system, with details on how it has been modified from the previous implementation described in D7.3.1 [10]. Section 4 describes the desktop and mobile applications, highlighting the improvements and difference from the previous released version. Section 5 introduces the new applications developed to meet the requirements of supporting organisational collective intelligence building through post-incident management tools. Section 6 summarises how the requirements identified in D7.1 [8], the recommendations highlighted in D7.5.1 [11] and the new requirements defined in Section 2 have been met in the final demonstrator. Section 7 concludes the document presenting a summary of the activity done. Appendix 1 presents installations instructions for the mobile and the desktop application.

Abbreviations and Acronyms

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DTO	Data Transfer Object
EIP	Enterprise Integration Pattern
ER	Emergency Response use case
ESB	Enterprise Service Bus
EXIF	Exchangeable Image File
FLO	Forward Liason Officer
FOAF	Friend-of-a-Friend
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
JAXB	Java Architecture for XML Binding
JAX-RS	Java API for RESTful Web Services
JMS	Java Messaging Service
JSON	JavaScript Object Notation
ID	Identifier
OSGi	Open Services Gateway initiative
POJO	Plain Old Java Object
RDF	Resource Description Framework
REST	Representational State Transfer
SPARQL	SPARQL Protocol and RDF Query Language
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
UI	User Interface
URI	Unique Resource Identifier
WKI	The WeKnowIt project
WKI DS	The WeKnowIt Data Storage
WP	Work package

WS	Web Services
WSDL	Web Service Definition Language
XML	Extensible Markup Language

Table of Contents

List of Figures

1. Introduction

This deliverable describes the implementation of the final prototype of the Emergency Response (ER) demonstrator: according to section B.1.3.6 Work package descriptions of WeKnowIt Annex I – "Description of Work", Deliverable D7.3.2 – "Emergency Response case study implementation", has to report the final implementation of the Emergency Response Case Study.

The demonstrator for the WKI Emergency Response Scenario has four primary aims. It should enable individuals to upload information to the WKI system about an emergency incident using mobile or desktop devices; this information should be enriched through the action of the intelligence layers present in WKI. This information should then be presented to citizens and ER personnel allowing them to understand the incident and make improved decisions on the basis of this information. Additionally it should allow the ER to perform post-incident analysis using a selection of tools. All above functionalities should be supported by the WKI platform and architecture developed within WP6.

The process of developing the WKI ER final demonstrator is shown below:

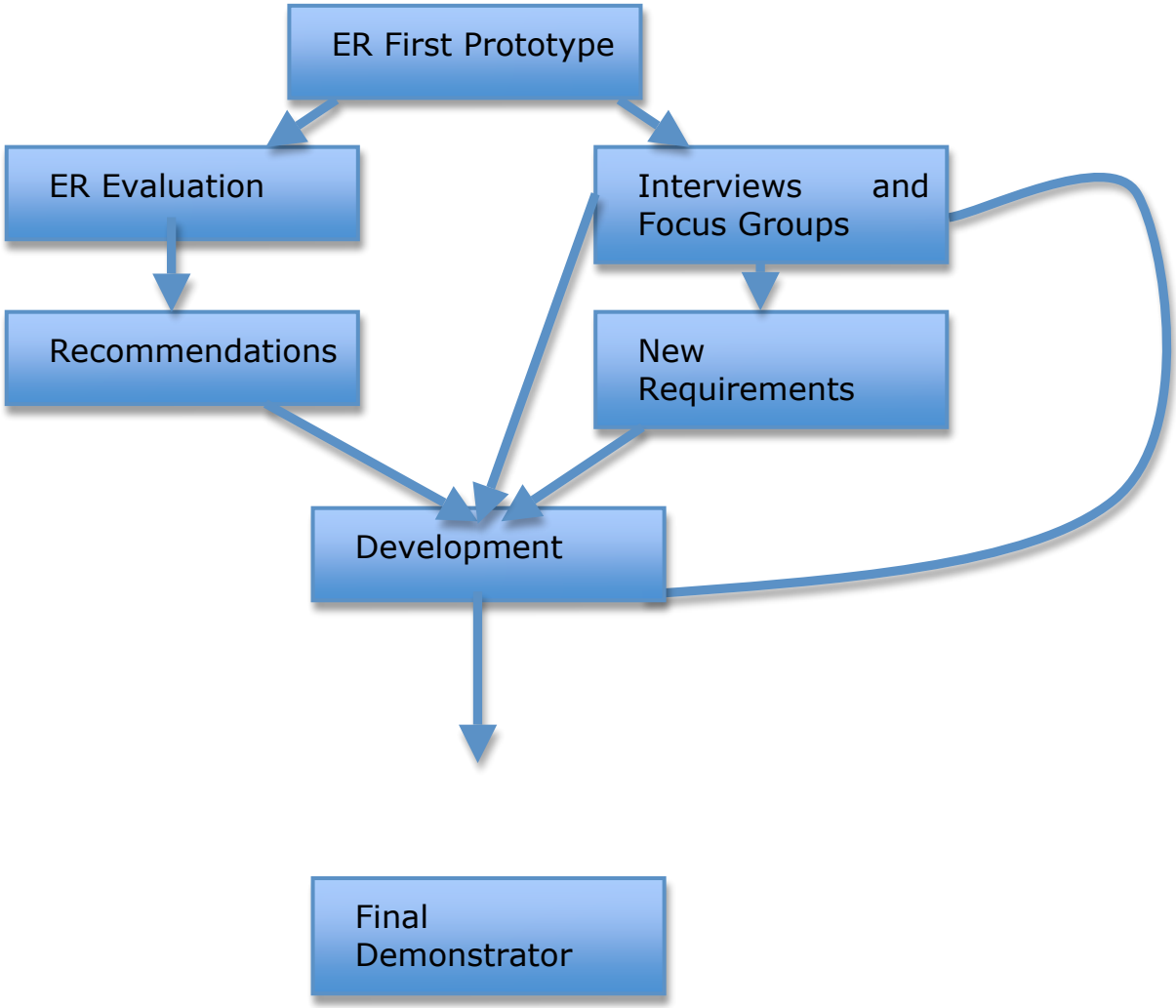


Figure 1 - Prototype 1 Development Process

The development process started from the first prototype implementation described in D7.3.1 [10] and from the results of the evaluation reported in D7.5.1 [11]. Further internal tests and user interviews and focus groups were then conducted, to better re-asses the requirements and plan for the final demonstrator development. Finally, a new version of the web application and mobile application were developed to allow users to access and interact with the system. New tools were developed for post-incident management, allowing users to search for information, perform quantitative analysis and collective incident debriefing and root cause analysis.

1.1. Focus and Aims of the Implementation

The first prototype (described in D7.3.1 [10]) demonstrated how the intelligence layers present within the WKI system contribute to Collective Intelligence processing which allows the ER system to function effectively. The final version of the ER demonstrator therefore needs to focus around the *results of the first implementation* and of the recommendations and additional requirements derived from the evaluation and user studies.

The user-centred design lifecycle is a standard cyclical methodology [5] (see Figure 2) for developing systems with the requirements of the user at the centre of the process. It is shown below:

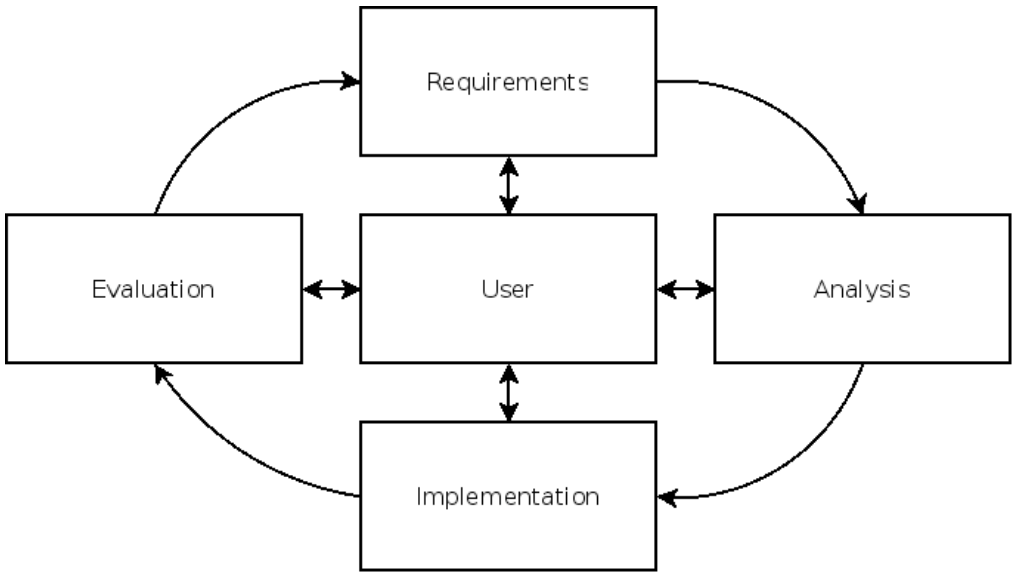


Figure 2 - User Centred Design Methodology

Thus, each stage of the user-centred design process involves both progression of development and validation with target users. Previous

work has defined the initial requirements for the ER demonstrator (described in D7.1 [8]) and the requirements highlighted after the first design-evaluation phase are reported in Section 2. Following this, the overall architecture used to implement the system is described. The implemented interfaces are then described. Finally the requirements described are revisited in order to assess how the final prototype has met these requirements.

2. Requirements for the final ER demonstrator

Following the results of the first evaluation (reported in D7.5.1 [11]) recommendations and new requirements were highlighted. These results were further validated and incremented during user interviews and focus groups with members of the Sheffield and Doncaster Emergency Response Teams – this step of the user studies was conducted in parallel with the development of the final demonstrator, to gain real-time feedback that could be very rapidly incorporated in the developed version.

2.1. Recommendations from first phase user evaluation

The results of the first evaluation show the need to improve some of the basic functionalities such as

- The tag filtering interface was counter-intuitive. The selection of a tag caused it to be struck out in the tag cloud and the corresponding images were removed from the map display. This process had two problems. Firstly, the evaluations have shown that this process was counter-intuitive for most of the participants. The majority of the participants that accessed the information through the tag cloud expected that selecting a term in the tag cloud would highlight the corresponding information in the map. Additionally this meant that if an image is attached to multiple tags, the user must spend time discovering what these tags are in order for information to be displayed.
- None of the participants made use of the evidence list part of the interface (see the screenshots in the appendix). The interface component should be restructured in order to draw the benefits out for the user.
- The ER experts indicated that they would like to be able to analyse the images in more detail than is provided by the small version presented on screen. This request is in conflict with the general feeling from both groups that the map should be the focus of the interface and should be larger than it currently is. A solution to this problem may be just to offer the ability to view the image as an overlay over the current interface.
- A further problem with the interface was that the connection between the various information displays (e.g. the tag and time displays) was relatively weak. For example, the tag widget displayed tags that are present in the WKI-DS regardless of the state of the other components. Some of the participants felt that changes in one widget should be reflected in the other widgets.

With regard to the upload interface:

- The process of tagging was often misinterpreted by the participants, thus needing simplification and clarification.
- The process of localising the image should be simplified, by making more use of the intelligent services or by providing an alternate means of stating the location of the image.
- The responsiveness of the interface should be improved.
- There was a need for technology that can assist the ER experts at identifying critical information and distinguishing between incidents (for example, clustering technologies).

2.2. New requirements from ongoing user studies

Following interviews with members of the Emergency Response Teams further requirements were highlighted

- In addition to post images about an incident, a provision should be made to post relevant information about the incident itself, giving the user the possibility to upload a subset of the information from the mobile application and then editing the report when and how needed (for example more information could be added as the situation evolves, or ER members based in the office could edit the report to add information)
- A search functionality is fundamental to
    - Overcome information overload
    - Perform post-incident analysis
- There is a need for a quick way to contact the ER or citizens that uploaded information, for example for ask for more details or more images.
- There is a need for uploading and accessing audio files.
- Whilst it is possible to visualise the information using the interface during an emergency, provision should be made for more advanced mechanism of incident analysis, that include collective investigation for an incident cause and report production.

3. Prototype Architecture

The architecture of the final ER implementation does not differ substantially from what was prepared for the first prototype and described in D7.3.1 [10]. This section recalls the architecture solution and gives information on changed and improved elements, as well as clarifies some uncertainties that were mentioned in D7.3.1 [10]. For the full information on the WKI System architecture please refer to D6.4.2 [7] and D6.1.2 [6].

The architecture consists of four different logical layers. This approach allows to achieve a clean separation of the intelligence layers and knowledge store present in the WKI system from the user interface components. Because there is no direct connection between the user interface and the underlying services both are free to change their implementation without having an effect on the other. This improves the maintainability of the system and facilitates independent development of both parts.

The WKI core system consists of a number of OSGi services running over a Fuse ESB[1]. This contains the services from each intelligence layer in addition to the data storage and knowledge base modules. The Composition Layer is then used to mediate between the core system and the user interface layer. This allows the user interface to make single requests to the Composition Layer that are then translated by this layer into multiple service requests. On top of this layer is the Translation Layer, which converts the POJOs that are generated by the core services into a format, which is more amenable to the user interface layer. Finally, the user interface layer presents information to the user and allows the user to interact with the system via the different architectural layers. The following section describes the architecture in details.

3.1. The WKI System Architecture

The architecture of the WKI System was prepared by WP6 and described in D6.1.2 [6] and D6.4.2 [7]. This section repeats and slightly updates information that were included in D7.3.1 [10]. It provides information that is necessary to understand the integration of the use case application with the WKI System. Figure 3 presents layers and components of the WKI System. The colours of components on Figure 3 mark the division between the *use-case applications* (green colour), and the *WKI System* (blue colour). Yellow vertical arrows represent messages exchanged between the WKI System and the UI components.

[1] http://fusesource.com/products/enterprise-servicemix/

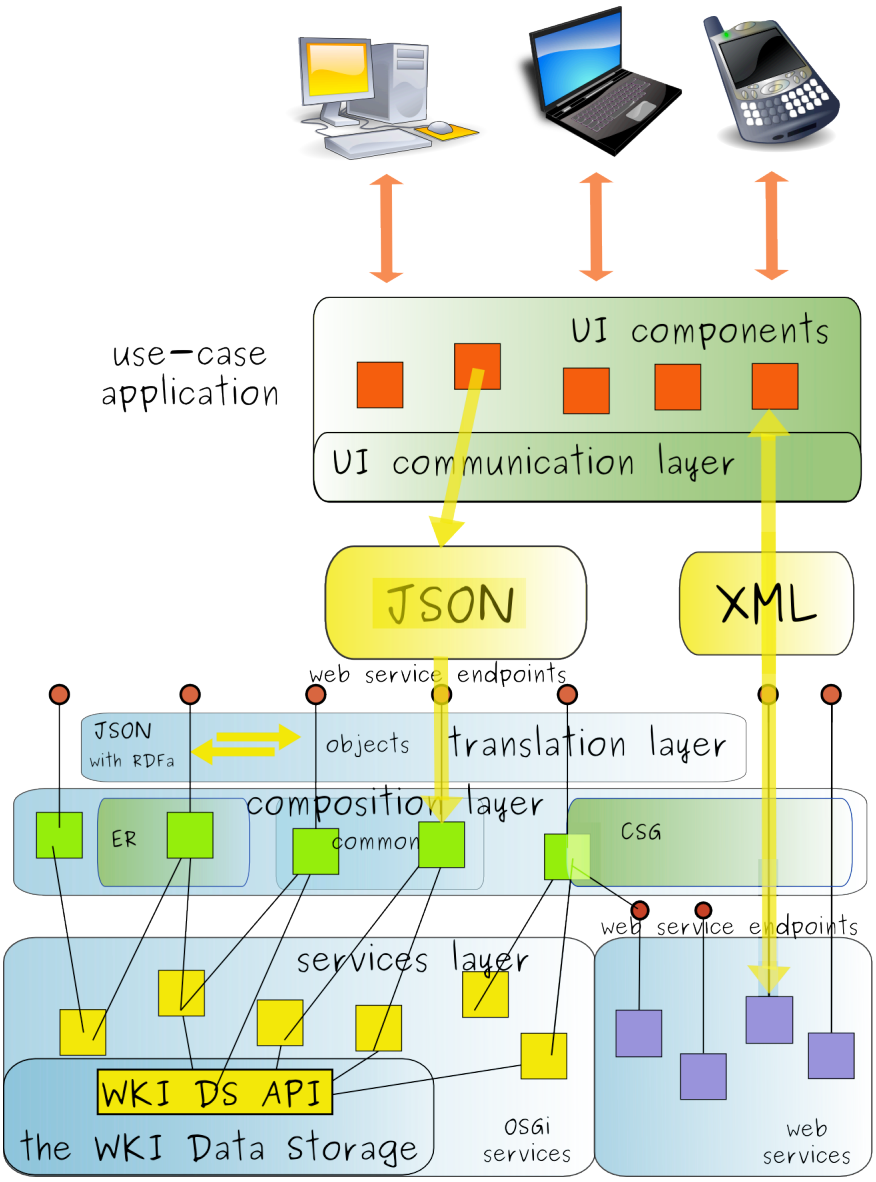


Figure 3 UI components integration with the WKI System - an overview

3.1.1. The use-case applications

The use case applications consist of:

1. UI components (represented as red rectangles on Figure 3), which are responsible for presentation of data. They can also include some business logic (even thought the real logic - especially combining of data from many services – is handled by the Composition Layer).
2. UI communication layer - responsible for making calls and receiving responses from the layers underneath (i.e. Composition Layer). It takes the burden of communication from UI components and makes them agnostic to messaging protocols.
3. Composition Layer, which acts as an intermediary between the UI components and the services of the WKI System. Yet, it is unaware of the UI components. Composition Layer (described in section 3.2)

is accessible via web services. It returns data that is later translated by Translation Layer so the components of the User Interface Layer can use them without further transformations

It is worth noticing, that UI components and UI communication layer live outside the OSGi environment provided by the WKI System, while the Composition Layer lives in this environment.

3.1.2. Messages

Messages between the WKI System and the UI components are exchanged via the HTTP protocol.

UI components of the ER use case rely on the messages in JSON[2] format. UI components of the CSG use case use different types of messages. Some services of CSG use case return XML, which is consumed directly, without any transformation, by the UI components. Such messages do not go through nor the Translation Layer, neither the composition layer.

The Translation Layer (part of the WKI System) is responsible for the translation of messages format.

3.1.3. The WKI System

The WKI System consists of:

- WS endpoints (represented as red dots on Figure 3),
- The Translation layer,
- The Services layer.

3.1.4. WS endpoints

WS endpoints can expose:

- Compositions of services,
- Single services.

The first option is preferred, as it takes the burden of combining outputs derived from many services from the UI components.

3.1.5. The Translation Layer

The Translation Layer is responsible for translating messages between formats used by the WKI System and by the UI components of ER use case. This layer is capable of translation of Java POJOs to JSON (and vice versa). In Figure 3 this layer is placed within the WKI System as one of its components. It can serve many different use-case applications as long as they accept formats that it provides. In other case, the Translation Layer

[2] http://json.org/

must be enhanced with ability of transformation to and from the required format.

3.1.6. The WKI Services

The services (yellow and violet squares) on the bottom of Figure 3 represent the low-level, atomic services provided by WPs, as described in D6.4.2 [6]. Some of the services live in the OSGi environment, while others are accessible via web services and are outside the OSGi environment of the WKI System.

Among the services, one is singled out - it is the WKI DS service. Technically, it is just one of many services, but it plays a central role in the system. The use-case application can contact with the WKI Data Storage only by using other services - the interface of the WKI Data Storage is not exposed via web services.

3.2. Composition Layer

The role of the Composition Layer is to provide an externally accessible API of the WKI System. This layer is responsible for intercepting messages from clients (e.g. use-case applications) and passing them to appropriate services of the WKI System.

D7.3.1 included description of an early version of the Composition Layer. This deliverable provides enhanced description, including changes that were introduced after D7.3.1 [10]. The implementation of this layer was updated with introduction of EIP patterns (section 3.2.1.1), preparation for usage of realiable messaging (3.2.1.3) and use of XML for services orchestration (3.2.1.2 and an example in 3.2.2.1).

Also the API of the Composition Layer was updated. This deliverable describes its latest version (section 3.2.2) with significantly extended "common" part (see section 3.2.2.1).

3.2.1. Implementation of the Composition Layer

Technically Composition Layer consists of three OSGi bundles that are deployed to the OSGi environment provided by the WKI System:

- wp7-compositionlayer-commons – described in 3.2.2.1 - contains common functionalities used by both use-case applications,
- wp7-compositionlayer-csg – contains CSG-specific functionalities; not used by the ER prototype; for information on this bundle please refer to D6.4.2 [7] where full description of REST API of the WKI System is presented,
- wp7-compositionlayer-er – described in 3.2.2.2 - contains specific functionalities the ER use-case application.

3.2.1.1. Enterprise Integration Patterns

Enterprise Integration Patterns (EIPs) are "design patterns for the use of enterprise application integration and message-oriented middleware".[3] The idea of such patterns was popularized by Hophe in 2003 [4]. The aim of the patterns is to provide good practices for implementation of asynchronous messaging architectures. They describe ways of routing messages and monitoring the health of a messaging system. EIPs are technology-agnostic, and can be implemented using various messaging technologies (e.g. SOAP, JMS, MSMQ, TIBCO[4]). Table 1 presents few examples of EIPs.

EIP	Description
Content Based Router	Routes messages to different destinations depending on their properties.
Dead Letter Channel	Processes messages that can not be delivered to their destination points.
Splitter	Treats parts of a complex message separately (as separate messages).

Table 1 - EIPs Examples

Selected EIPs are used in the Composition Layer of the WKI System. They are realised with use of Apache Camel[5].

3.2.1.2. XML Usage

Orchestration of services is described using XML files. This format has advantages over coding them using Java. It is much easier to understand XML files that resemble natural language and easier to modify them. Developer needs to know only what "objects"[6] are available (e.g. `capBean`, `dsBean` on Listing 1) and what methods each of them provides. Then, using EIP patterns any flow can be described.

3.2.1.3. Reliable Messaging

Reliable messaging system must be able to survive various problems related to unavailability of parts of the system. Composition Layer of the WKI System is ready to use JMS queues with message persistency. Messages are persisted in relational database, and in case of failure (even system breakdown) delivery attempts are resumed. This guarantees that temporary problems with message delivery will not break the communication, and messages will be eventually delivered.

---

[3] http://en.wikipedia.org/wiki/Enterprise_Integration_Patterns

[4] http://www.tibco.com/

[5] Apache Camel, http://camel.apache.org

[6] Here, by "objects", we mean WP1-WP6 services, wrappers over this services (provided by WP7), and utility classes of WP7.

It is possible to control exactly which messages (or messages channels) should be persisted. This minimizes the overhead related to additional database operations.

3.2.1.4. REST interface

Composition Layer exposes its API via REST interface. This allows clients of the WKI System to use a simple programming model (comparing to WSDL/SOAP approach). Services of the Composition Layer are annotated with JAX-RS[7] annotations provided by Apache CXF project[8]. The annotations are used to make the services available under specified URLs, and specify types (e.g. POST or GET[9]) and formats of requests and response.

3.2.2. API of the Composition Layer

The API of the Composition Layer consists of APIs of all three OSGi bundles listed in 3.2.1.

3.2.2.1. wp7-compositionlayer-common

The idea behind this module is to group common functionalities required by both use-case applications. Since D7.3.1 [10] a lot of new functionalities were implemented within this module.

Additionally functionalities of both ER and CSG use-cases were analysed, and common functionalities were moved to this module. This effort was announced in D7.3.1 [10], as the code duplication was a visible threat to project maintenance at that time. By merging ER and CSG services within this module, the cost of project maintenance was lowered by:

- reducing duplicated code,
- use of EIPs,
- replacing Java code with XML configuration files.

Package	Description
eu.weknowit.composition.layer .common	Abstract classes and exceptions used in the common composition layer.
eu.weknowit.composition.layer .common.dataflows	Interfaces and implementation of dataflows.
eu.weknowit.composition.layer .common.dataflows.beans	Beans for camel flow.
eu.weknowit.composition.layer .common.dataflows.processor	Camel processors for camel flows.

[7] http://en.wikipedia.org/wiki/JAX-RS

[8] Apache CXF, http://cxf.apache.org

[9] http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

eu.weknowit.composition.layer.common.dataflows.wrappers	Interfaces of data flows wrappers.
eu.weknowit.composition.layer.common.dataflows.wrappers.impl	Implementation of commons wrappers.
eu.weknowit.composition.layer.common.providers	Providers for REST services.
eu.weknowit.composition.layer.common.services	Composition Layer Commons Services Implementation.
eu.weknowit.composition.layer.common.utils	Utility classes.

Table 2 - Packages of the Composition Layer Common

The API of the Common Composition Layer is included in eu.weknowit.composition.layer.common.services package. Other packages include classes that constitute implementation of this API – from utility classes used, through custom Apache Camel processors, to REST services providers.

As expected in D7.3.1 [10], the growing requirements of UI layer resulted in more complex tasks being executed by Composition Layer. Because of this, direct calls made by the CL layer via OSGi registry, were partially replaced by easily configurable flows described in XML files. XML-based flows definitions - which are then executed by Apache Camel - are easier to maintain and offer some built-in additional functionalities (e.g. asynchronous calls).

An example of an XML Apache Camel flow definition file (part of file upload flow) is presented on Listing 1.

```
<route id="uploadFileDirect">
    <from uri="direct:uploaded.files" />
    <log message="The upload flow started."/>
    <to uri="bean:capBean?method=checkPermission" />
    <to uri="bean:dsBean?method=storeFile" />
    <to uri="bean:capBean?method=registerDocument" />
    <wireTap uri="direct:analyzing"/>
</route>
```

Listing 1 - Example Flow in Composition Layer

This XML file can be explained as a chain of the following commands:

- log information about the start of upload flow,
- execute `checkPermission` method of `capBean` (which represents WP4_CAP service) that will check if the user has permission to upload a document,

- store document in the WKI DS by executing `storeFile` method of `dsBean` (which calls WP6_DataStorage methods),
- assign permissions for this document to user that uploaded it, by calling `registerDocument` method of `capBean`,
- pass the control to another flow – `analyzing` – which executes methods of other services responsible for analysing the uploaded file (this flow is not presented for the sake of brevity).

This example also shows use of wire-tap pattern[10] - one of many EIPs (see section 3.2.1.1) provided by Apache Camel. This pattern causes the result (ID of stored file) to be returned to the client, and the next flow ("analysing") to be executed asynchronously.

Services of the Common Composition Layer

Table 3 presents services that are exposed by the Common Composition Layer component.

Service	Description
CapService	Allows executing CLD queries via REST calls.
DocumentService	REST service for retrieving documents from WKI-System.
HttpAccessService	Implementation of service retrieving files content from WKI DS by their ID.
HybridImageClustererService	REST service for image clustering based on both visual and tag similarity.
SparqlService	REST service for executing sparql queries.
UploadCamelService	Class exposes REST service for uploading the files. It starts the "analyzing file flow" which uses various WKI services in order to extract metadata from uploaded file.
UploadFromFlickrService	Starts analyse flow for images from flickr. For each file fetched from flickr, the UploadCamelService is executed.

Table 3 - Services of the Common Composition Layer

3.2.2.2. wp7-compositionlayer-er

ER use-case application uses mainly services from wp7-compositionlayer-common project (see section 3.2.2.1). Yet, some specific functionalities are included within wp7-compositionlayer-er project. No major changes were introduced in wp7-compositionlayer-er since D7.3.1 and the

[10] http://camel.apache.org/wire-tap.html

following description repeats what was stated in the previous deliverable [10].

This module contains the following packages:

Package	Description
eu.weknowit.composition.layer.er	API of the ER composition layer - interfaces of services.
eu.weknowit.composition.layer.er.dto	DTOs of composition layer.
eu.weknowit.composition.layer.er.impl	Implementation of ER composition layer services.
eu.weknowit.composition.layer.er.providers	Translation layer - contains providers for JSON/POJO translation (specific for ER use-case applications).

Table 4 - Packages of the Composition Layer ER

The API of the Composition layer is included in eu.weknowit.composition.layer.er package, and the core functionality is grouped in two packages – eu.weknowit.composition.layer.er.dto and eu.weknowit.composition.layer.er.impl.

DTOs of the Composition Layer

The DTOs of the Composition Layer are strictly related to the set of ontologies used in the WeKnowIt project. This is a requirement of the UI components of ER use-case, which work with JSON data format. Composition Layer, in cooperation with Translation Layer, delivers the data in the requested format.

Types	Description
ICLComponent, CLAbstractResource, CLResource, CLSiocItem, CLSiocPost	Interfaces and abstract classes for all DTOs.
	DTOs related to Tags.
CLDocument, CLFoafDocument	DTOs related to Documents.
CLEvent, CLEventNews, CLFullEvent, CLParticipation	DTOs related to Events.
CLSimpleUser, CLUser, CLUserDetails, CLUserPermission, CLAddress, CLOnlinePresence	DTOs related to Users.
CLGmlGeometry, CLGmlPoint, GeoCoordinates	DTOs related to geo localization.

Table 5 - DTOs of the Composition Layer

The DTOs of the Composition Layer are annotated with JAXB[11] annotations. This results in automatic serialization (marshalling and unmarshalling) of DTOs to/from XML during sending requests and receiving responses by web services.

Services of the ER Composition Layer

Services presented in Table 6 are available and exposed as the API of the ER Composition Layer via web services. Most of them are aimed at processing of a particular DTO. ICLSearchService differs in this aspect, as it provides an "entry" to the WKI DS allowing for execution of various search queries.

Service	Description
ICLEventService	This interface declares all methods required to add/update//retrieve event (CLEvent) including functionalities like "return all events that are related to this one".
ICLResourceService	Service provides methods to operate on resources (CLResource).
ICLSearchService	This service provides specialized methods for search in the WKI Data Storage.
ICLTagService	Interface contains method for processing tags (CLTag).
ICLUserService	Service provides operations connected with users (CLUser), like addition of friends, profile update etc.

Table 6 - Services of the ER Composition Layer

The methods of the services are prepared in such way, that the UI components can make direct calls to receive the data required by a particular element of the user interface. For example, the home page of the WKI System displays latest public events. Such events can be retrieved from the WKI DataStorage with one request - via getLatestPublicEvents method of ICLEventService. Thus, one of the requirements of the Composition Layer (to make UI layer free of the business logic related to manipulation of data retrieved from different services) is fulfilled.

3.2.3. Workflows

For the first prototype of Emergency Response Scenario (described in D7.3.1 [10]) some workflows were implemented within the Composition Layer. They presented "how data is passed between the core WKI services

[11] http://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding

and the various users that interact with the system in the Emergency Response Scenario".

The following workflows were discussed:

- Creating a New User
- Logging In
- Upload Image
- Tag Image
- Making Images Public
- Commenting

For the final implementation of the system these workflows were revised and update. The most significant changes were introduced to the "Upload Image" workflow:

- it is now a part of a more generic "Upload File" workflow,
- being of general nature, it is now a part of Composition Layer Common,
- it does not block the client; the ID of uploaded file is returned immediately, while other time-consuming operations are performed asynchronously.

3.3. Translation Layer

The Translation Layer mediates between the composition layer and the user interface layer. The purpose of this layer is to transform the standard java objects that are used in the composition layer into the JSON data that is shown at the user interface level.

The Translation Layer is responsible for translating messages between formats used by the WKI System and by the UI components of ER use case. This layer is capable of translation of Java POJOs to JSON (and vice versa). This layer is under development and subject to the requirements of a specific use case.

3.4. User Interface Layer

The User Interface layer has been implemented using the Ruby on Rails[12] framework using the JRuby[13] interpreter. This framework enables the rapid development of the user interface framework and has built in support for simple access to REST services. The foundation of the user interface is implemented as HTML and CSS pages, with each page corresponding to a fundamental action within the user interface workflow

[12] http://rubyonrails.org/

[13] http://jruby.org/

(see below). On top of the basic framework, Javascript is used to enhance the user experience and, additionally, AJAX support allows the interface to connect to the core WKI services. JRuby was chosen as a language since it will allow for future versions of the interface to support more component based widgets, such as those developed within the JMaki[14] framework.

3.5. Integrated Services

The following table illustrates which WKI services are integrated (or will be integrated) in the final Emergency Response Demonstrator. For each service we present the name, what functionality it enables in the prototype, and how it is visible in the User Interface (or how it impacts it).

[14] https://ajax.dev.java.net/

WKI Service Name	Functionality	Interface Visibility
WP1_UsersMessaging	This service enables of messaging exchange amongst users. This function enables the integrated mechanism of an instant messaging facility	This service is used to send messages from the Desktop Interface and receive them on the smartphone application.
WP2_SemanticPhotoQuery	Retrieving Flickr photos that best match the domain ontology concepts for specific time intervals	Incorporates relevant images if available
WP2_Text_Classification	To determine the similarity between a given text (set of texts) and some pre-defined language models relating to different categories. The predefined language model is created as part of the off-line text classification process, using a sample of classified training data provided by the (WP7) application providers. A	Tag suggestions are presented in both desktop and mobile interface when a new document is uploaded.

	is constructed from a collection of example texts relating to that category (e.g. reviews and not-reviews). The modelling tool creates a collection of category models which can be used by the classification service to classify a text.	
WP2_TagNormalization	The list of tags that have been assigned to each resource are matched to one or more domain ontology concepts. Each matching is accompanied by a weight coefficient that shows the degree of the tag-concept relation. For the appropriate mapping between tags and formal descriptions, external sources of knowledge (such as WordNet, Wikipedia, etc) are exploited. Furthermore, string	Backend integration, depends on data quality, no visible effect in the interface.

	processing, matching and comparison functionalities are employed. The direct benefit of this process is that it is going to yield interoperability to our dataset and allow for performing reasoning operations. Moreover, it is expected to enhance the results of the provided visual analysis in terms of landmarks or points of interest identification.	
WP3_LocalTagCommunityDetector	Given some input tag, the goal of this service is to identify a collection of tags that form a community around it. A community of objects is defined with respect to a graph of objects. In our scenario, we consider a tagging system where users tag resources (e.g. questions, answers, pictures or	Tag suggestions are presented in both desktop and mobile interface when a new document is uploaded.

	whatever we want to support). Based on the tag co-occurrences we can create a network of tags that are closely connected with each other and less connected with the rest of the network.	
WP3_Lexical_Spam_Detector	Detects spam from user input	Used when checking titles, description and tags for an uploaded image, both from mobile and desktop interface
WP6_DataStorage	This service provides API for storing/retrieving data from the WKI Data Storage. This API covers all components of the WKI Data Storage - triple store(s) and other databases, and file storages.	Used to store data, no immediate impact visible in the interface.
WP1_Attention_Streams	This service learns from the documents accessed by a user and builds a model of attention, to deliver personalized	Used in the main desktop interface to personalise the exploration of content.

WP1_Attention_Streams	This service learns from the documents accessed by a user and builds a model of attention, to deliver personalized content.	Used in the main desktop interface to personalise the exploration of content.
WP1_Contextual_Model	This service takes into account the context of interaction to deliver personalized interaction.	Used to deliver a device specific interface to the use
WP2_SpeechIndexing	This services indexes speech.	Integrated when speech is uploaded to the server.
WP2_SpeechTagging	This service allows tagging speech recordings.	Integrated when speech is uploaded to the server.
WP2_Text_Annotation	Automatically annotates entities in the text, using some predefined annotation models.	Used to geo-tag the text so that they can be mapped.
WP2_Text_Geocode	Provides a list of nearest geo-locations to the provided latitude and longitude values.	Allows the user to select the correct or human interpretable address from a

Table 7 - Integrated Services

4. ER Interface Design and Functionalities

In this section, the implementation of the final interface is described with details for the desktop and the mobile application. In Section 5 new applications for collective post-incident management are presented.

Following the evaluation results and the new requirements the demonstrator desktop interface has been completely redesigned, to offer a more consistent framework in which incidents, images and audio files can be uploaded and visualised. In the following sections a brief overview of the main changes is presented.

4.1. Upload Interface

The upload interface has been redesigned to support new functionalities such as the possibility to upload reports about an incident by filling in a form, both from the mobile and the desktop interface and the possibility to upload audio files (see Figure 4).

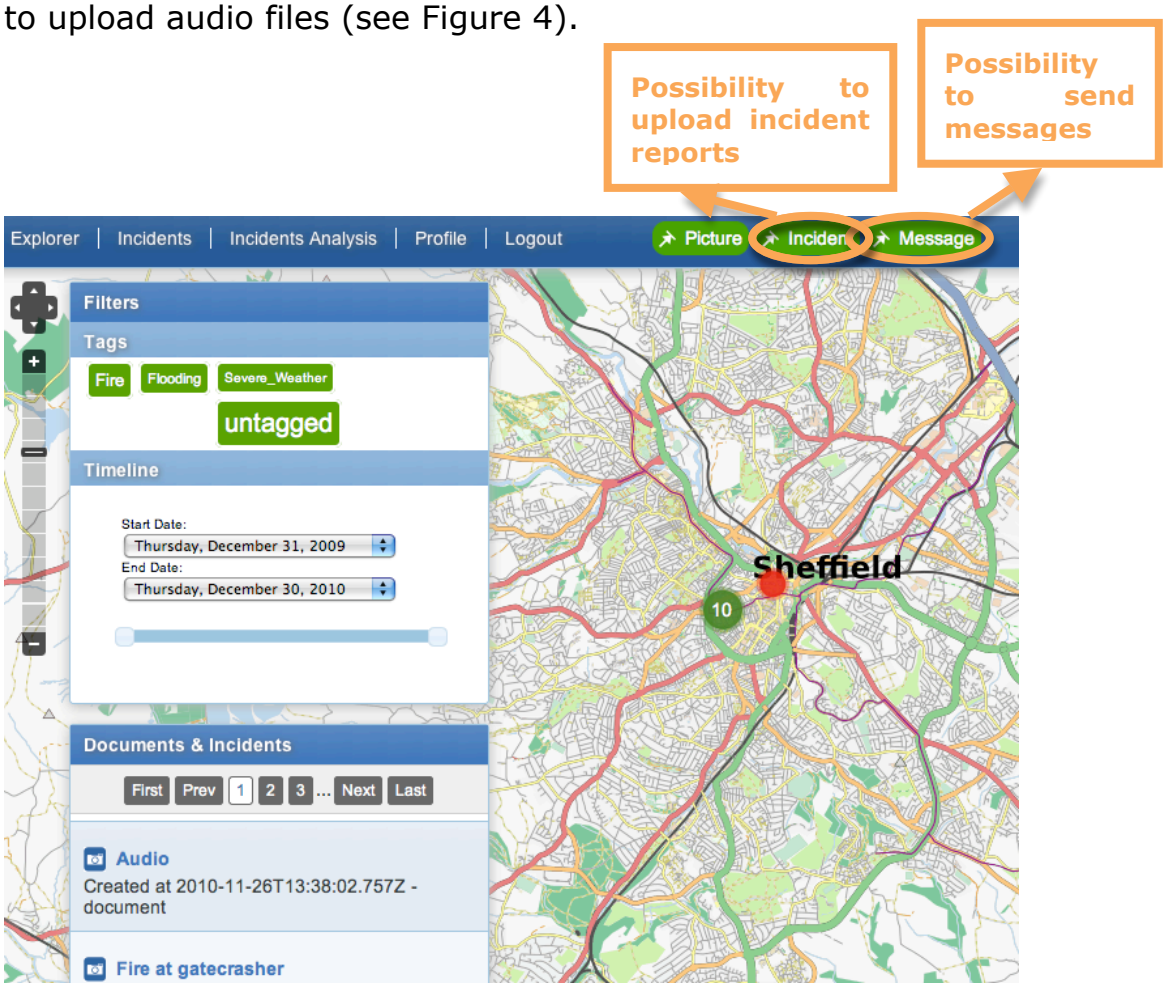


Figure 4 - New upload interface

For screenshots and information about the new upload features please see Sections 4.3.3 and 4.4. The process of tagging has been simplified: a new service from WP2 has been integrated to automatically detect possible tags from the title and the description of an image – the user can simply select one of the suggested tags clicking on it or insert a new tag (see Figure 5). The localisation of images I now happening through the EXIF data or GPS location – in absence of a given location the user can select a location by clicking on a map.

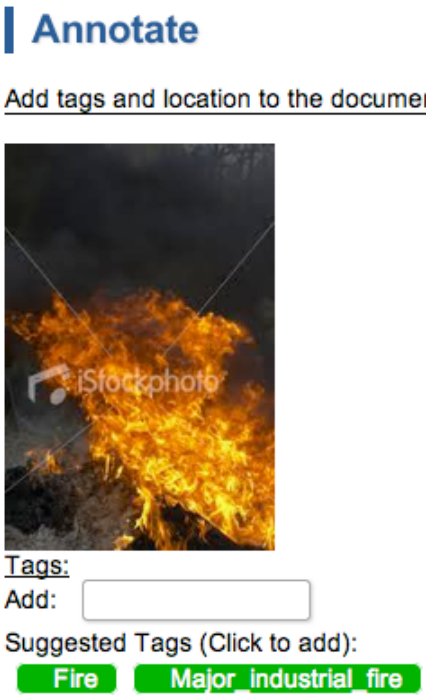


Figure 5 - Automatic Tag suggestion

Geolocation and reverse geolocation services have also been integrating in the upload process for what regards the incident reports, offering users a quick way to select a location for the incident (see Section 4.4.2.7).

4.2. Access Interface

The main geographical interface has been redesigned to be context-sensitive and to support consistent visualisation through the multiple filters available: the filters are now synchronised between each other and with the main interface, so to provide a holistic view of the data.

The data is clustered accordingly to location and type (see Figure 6) and when clicking on a cluster a layer is imposed on the interface that shows the list of documents (incident reports, images or audio files) or, if there is only one document, the document itself (see screenshots and details in Section 4.3.2.2)

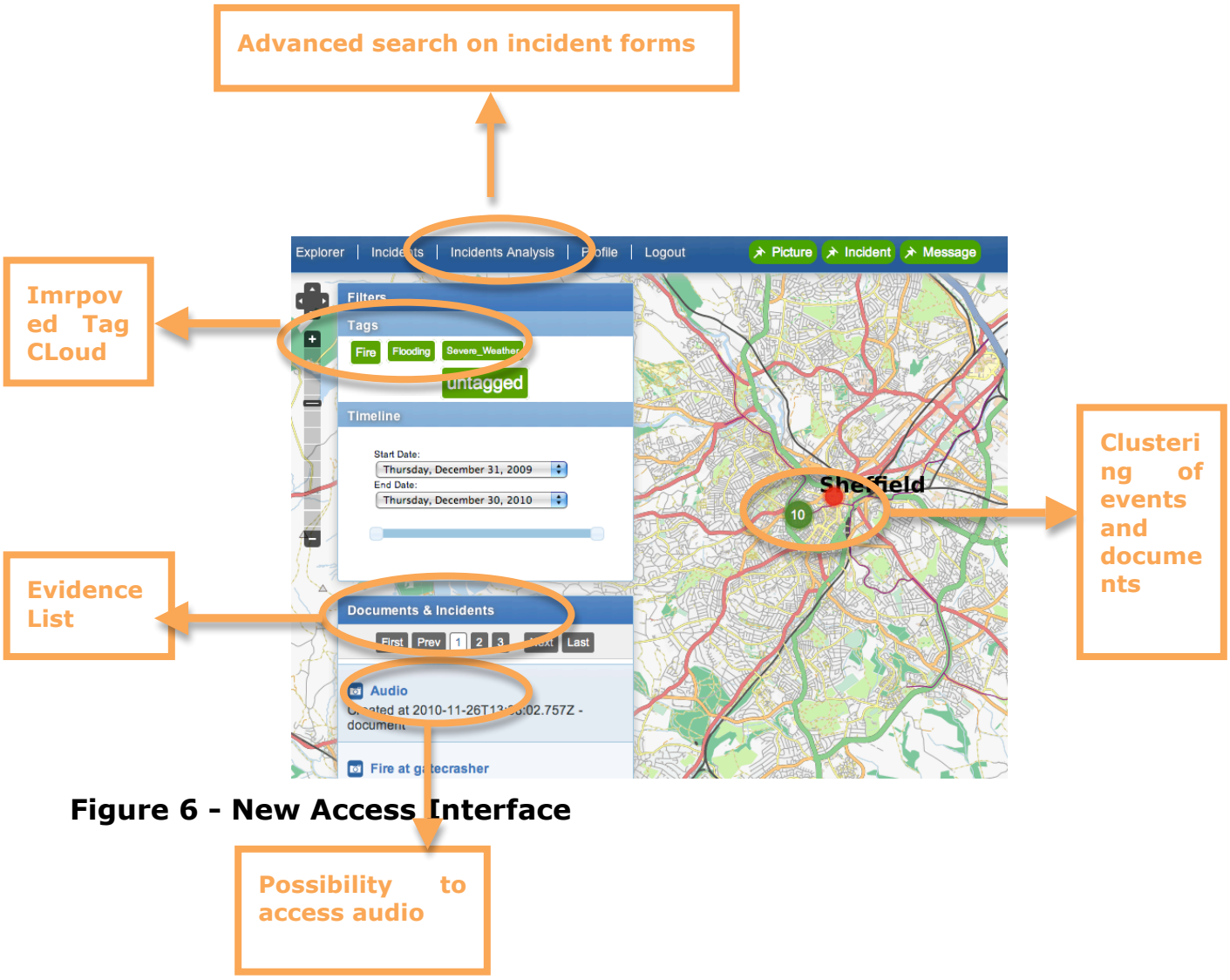


Figure 6 - New Access Interface

The tag cloud for browsing documents has been improved: it now works as standard tag cloud, selecting a tag immediately selects only the matching documents (see Figure 6). The evidence list panel is now part of the filtering widgets provided on the left hand side of the page and provide coherent filtering and visualisations with the other widgets (see Figure 6).

An alternative simplified visualisation is provided using a list of incidents, expandable for more details (for screenshots and details see Section 4.3.2.3).

Search functionalities are offered both in a simplified way to search amongst the incident list and in an advanced way to perform searches and quantitative analysis over the incident reports (for screenshots and details see Section 5.1).

A messaging service is now available to contact users on their smartphones, providing the WKI application has been installed (for screenshots and details see Section 4.4.2.4).

4.3. Desktop Interface

4.3.1. WeKnowIt ER login page

The login page presents the user with the possibility to login into the system or to register as new user. The user can log in using openID or using with a traditional username/password combination.

A link is provided to download the mobile application for Android Smartphones (see Figure 7).

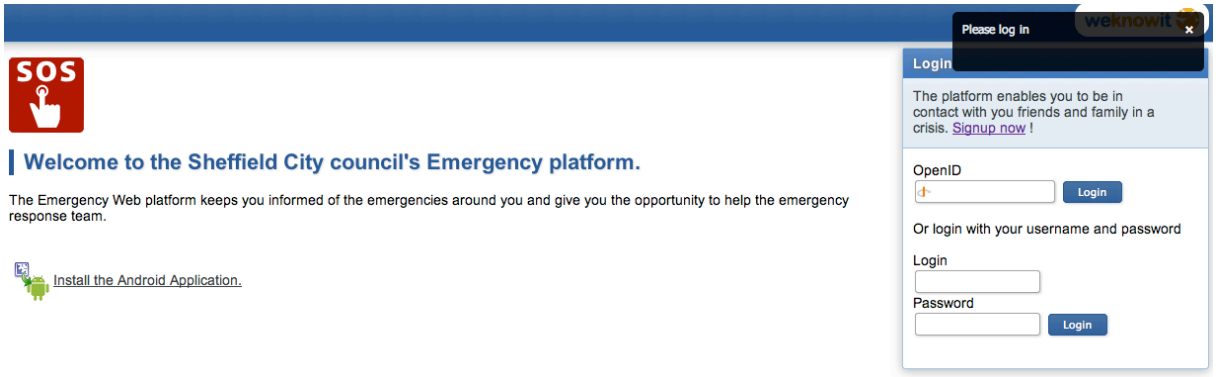


Figure 7 - WeKnowIt ER Login page

4.3.2. Accessing Data

After logging in the user is redirected to the main WeKnowIt ER view, the Explorer, that is the interface for browsing uploaded data, using multiple filtering dimensions. There is the possibility to switch between different applications by using the toolbar, that contains links to the incident view, the incident analysis tool (see Section 5), and the links to upload picture, incident reports and to send messages to the users (see Figure 8).

Figure 8 - Navigation toolbar

4.3.2.1. Explorer View

The Explorer is meant to be the main navigation modality in WeKnowIt, as it offers a holistic view of the situation combined with filtering widget to narrow down the information displayed.

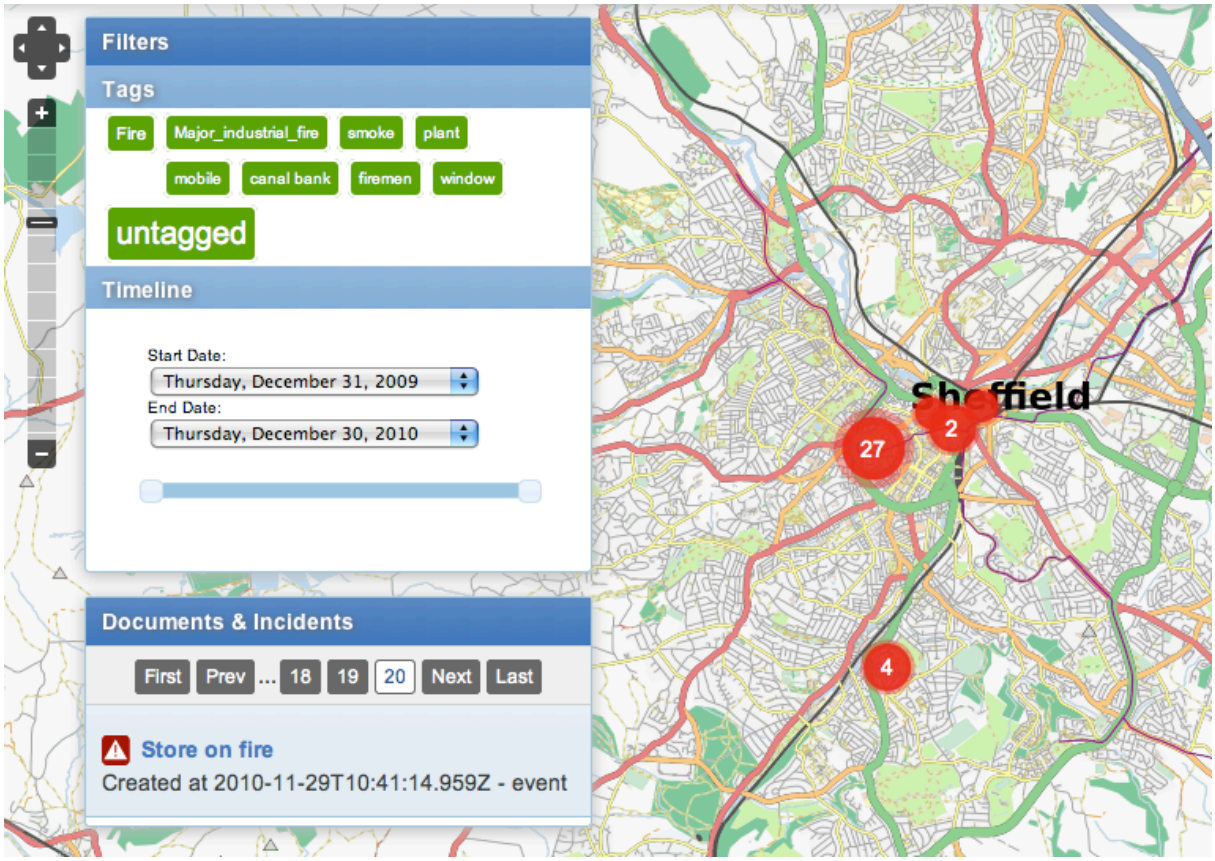


Figure 9 - WeKnowIt ER Explorer

The explorer interface main dimension is geographical, with all the available data displayed on a map. The data (images, incident reports, audio files) are clustered accordingly to time and location on the map. Each cluster is represented as a red dot, of increasing dimensions accordingly to the number of documents collated. To double-code this

important information, the number of documents is displayed inside the red dot (see Figure 9). When a cluster is clicked, a list of possible documents is displayed (see Figure 10), that can then be visualised using the document preview function (see Section 0).

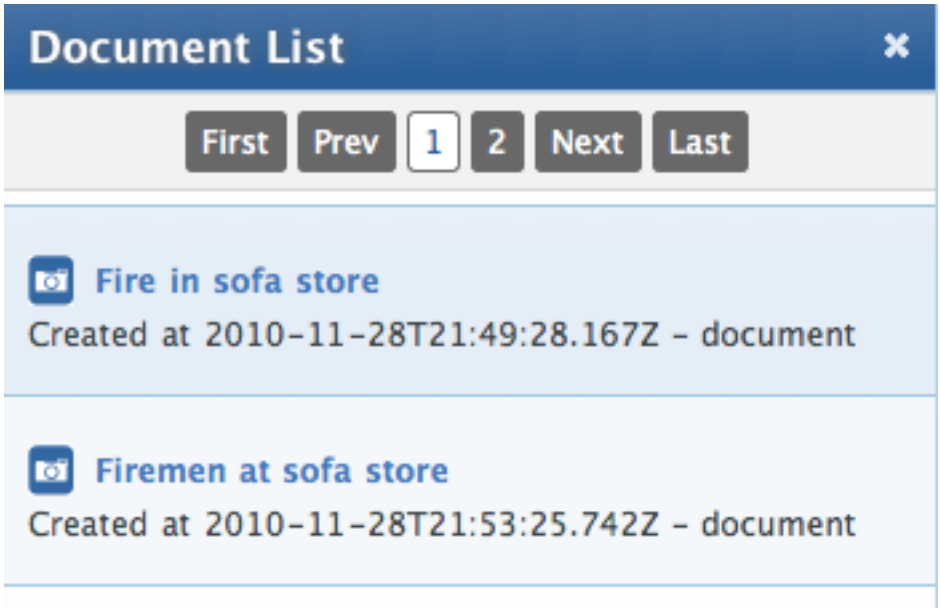


Figure 10 - Document List

On the left hand side of the interface the filtering widgets are presented.

The first filtering widget allows to choose tags from a tag cloud; the second allows to scroll or select dates from a timeline and the third is the evidence list, that is the list of documents and incidents meeting the desired conditions.

The whole explorer interface has been redesigned to provide cohesion and coherence to the navigation experience. The explorer is now context-sensitive and supports consistent visualisation through the multiple filters available: the filters are now synchronised between each other and with the main interface, so to provide a holistic view of the data. When a filtering condition is selected, the interface changes to display only the matching data (see Figure 11).

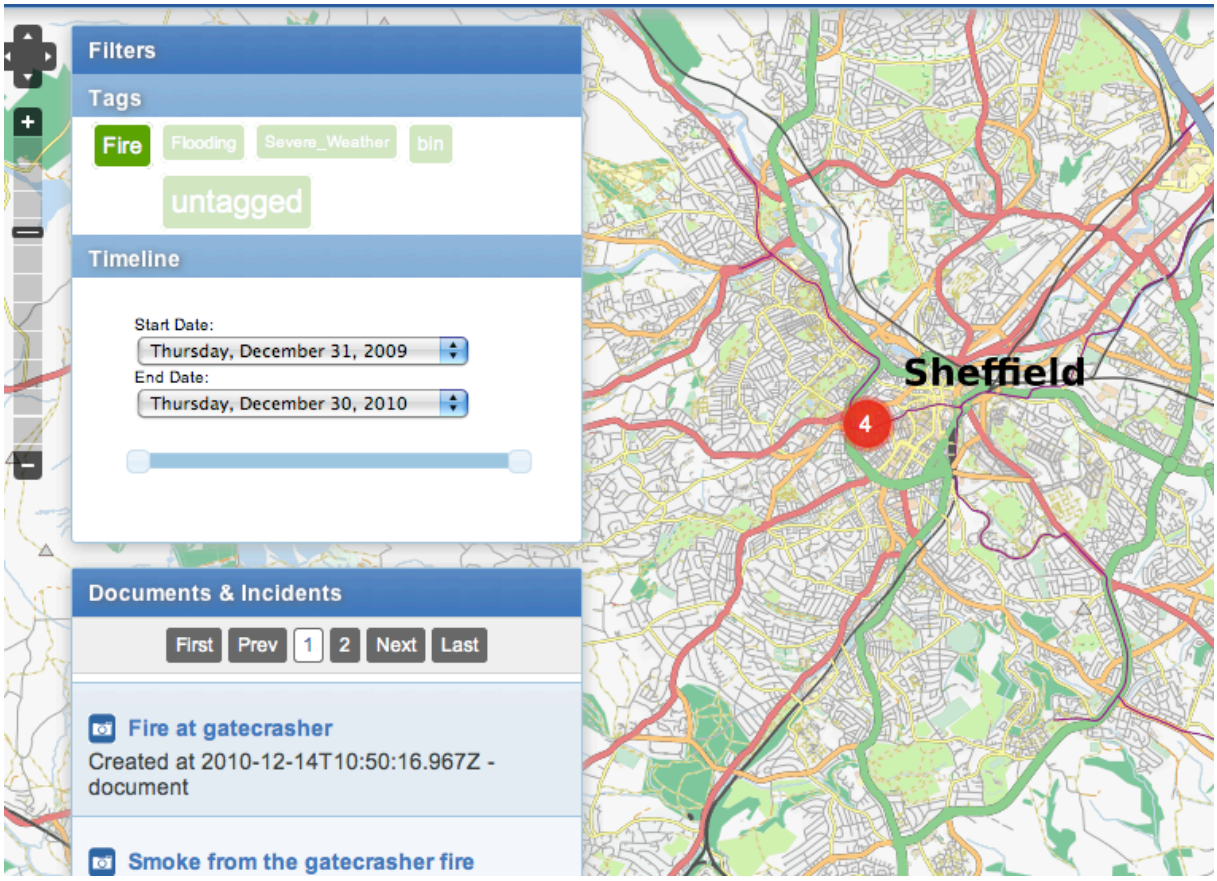


Figure 11 - Example of filtering widgets behaving in a consistent way – the tag fire is selected and only the relevant documents are displayed in the geographical interface and in the filtering widgets.

An added functionality is the possibility of switching the map layers used to visualise the information (see Figure 12).

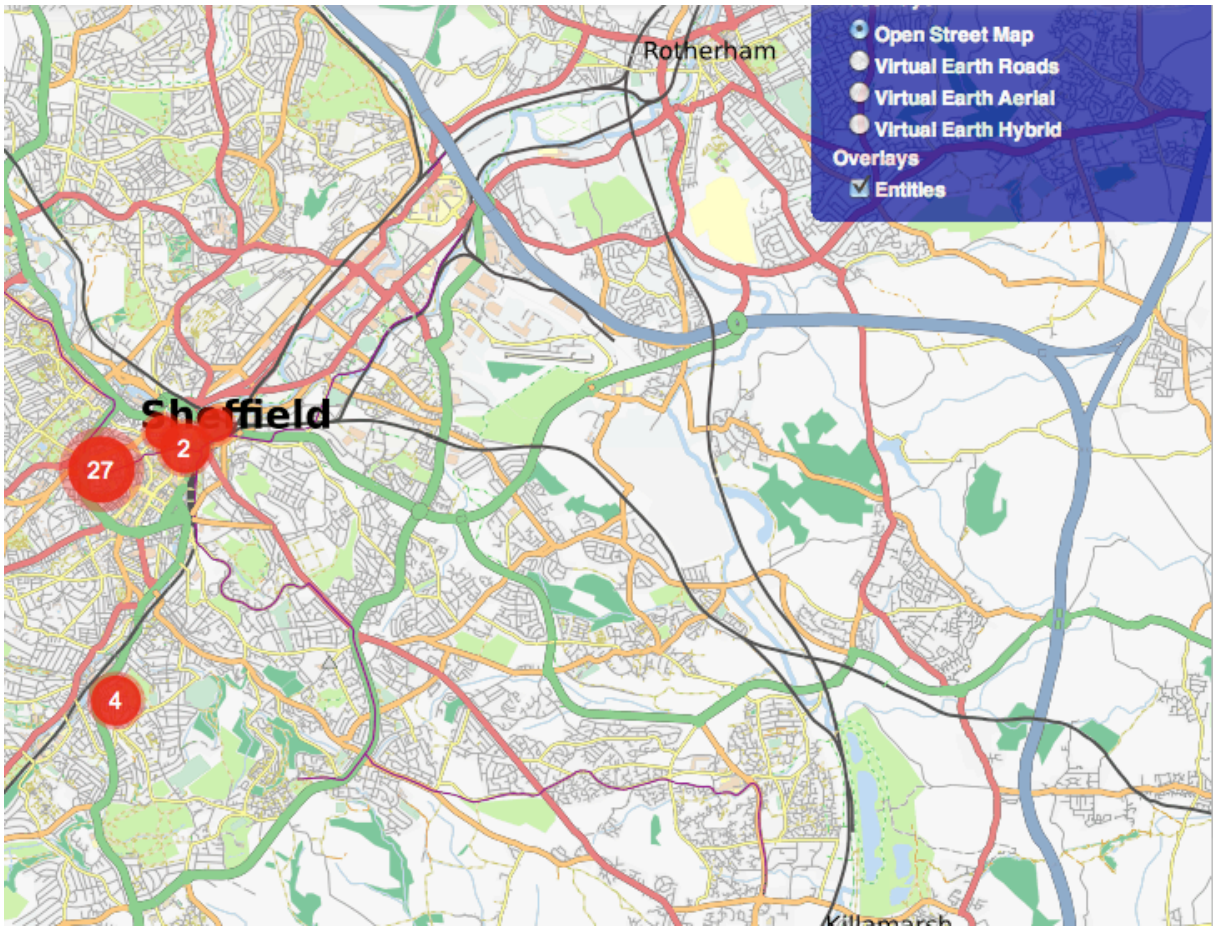


Figure 12 - Choosing the map layer

The pre-defined map layers uses OpenStreetMap as it contains advanced information about schools, hospitals etc that have been considered important by the ER Team.

It is though possible to switch to Virtual Earth Roads, Aerial or Hybrid, to match different information requirements and provide the right view and the right time. For example when assessing the road situation a simple view like Virtual Earth Roads could be helpful (see Figure 13); when instead having to find more information about a building and how the building is structured (for example how many floors for evacuation purposes) a view based on Virtual Earth Aerial could be preferred (see Figure 14).

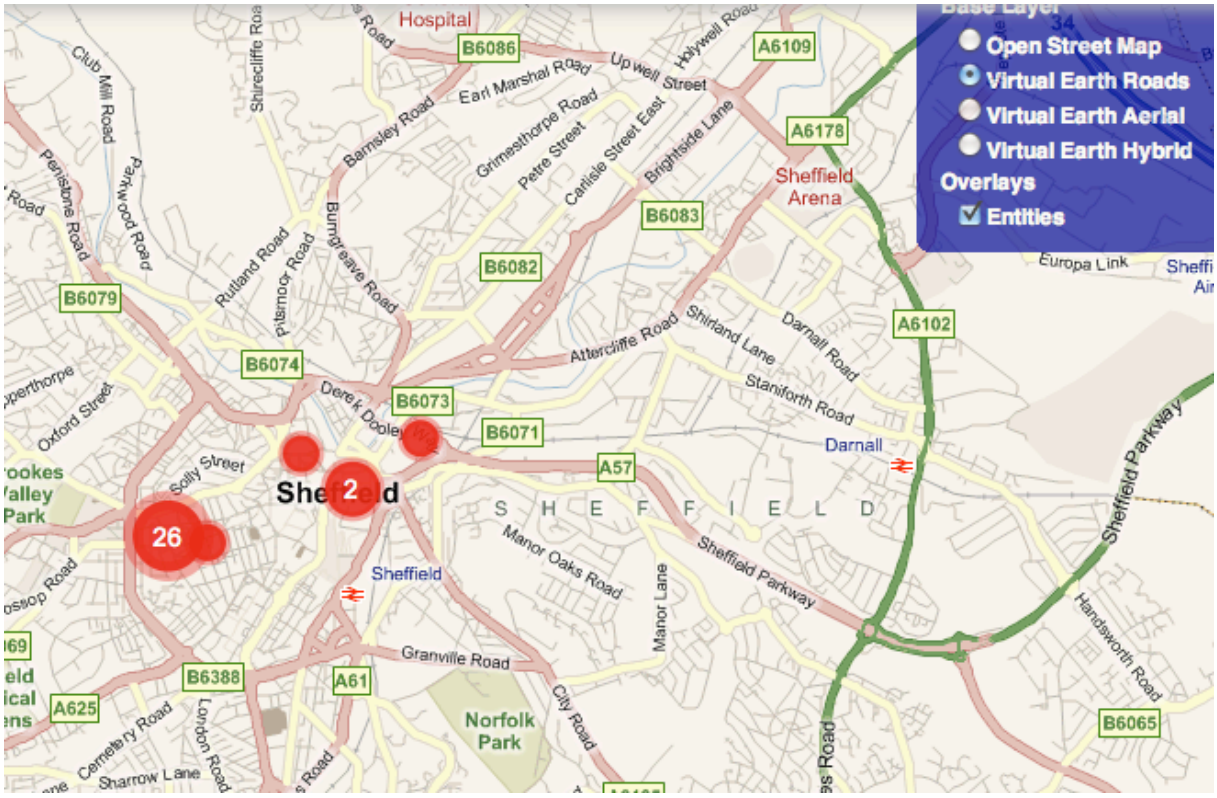


Figure 13 - Virtual Earth Roads view

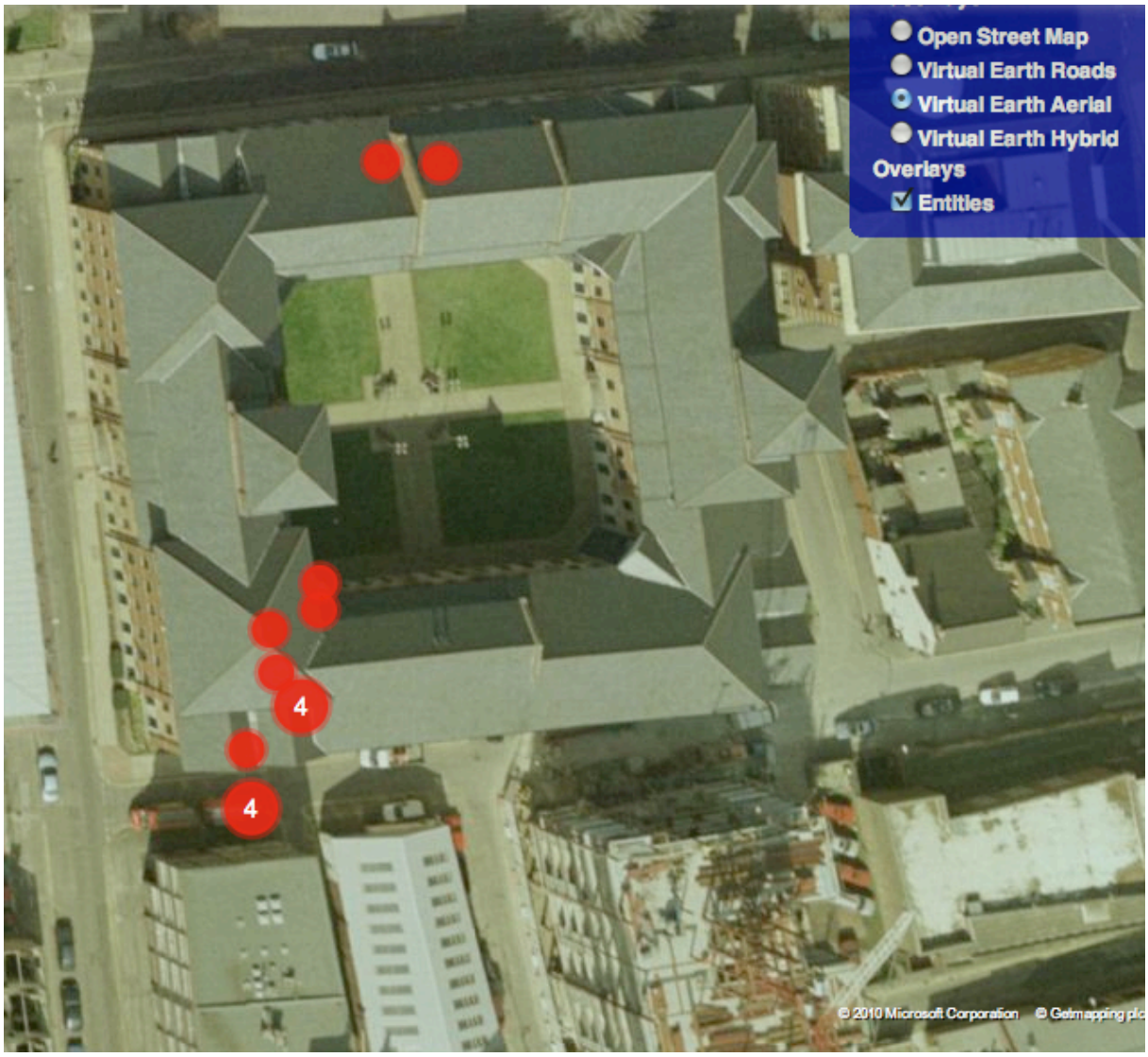


Figure 14 - Virtual Earth Aerial View

4.3.2.2. Document Preview

The aim of the document view is to provide a way to quickly analyse the uploaded document without loosing the navigation context and the possibility to visualise other documents. To achieve this, when clicking on a cluster or on a document link in the evidence list, a layer is displayed over the interface (see Figure 15).

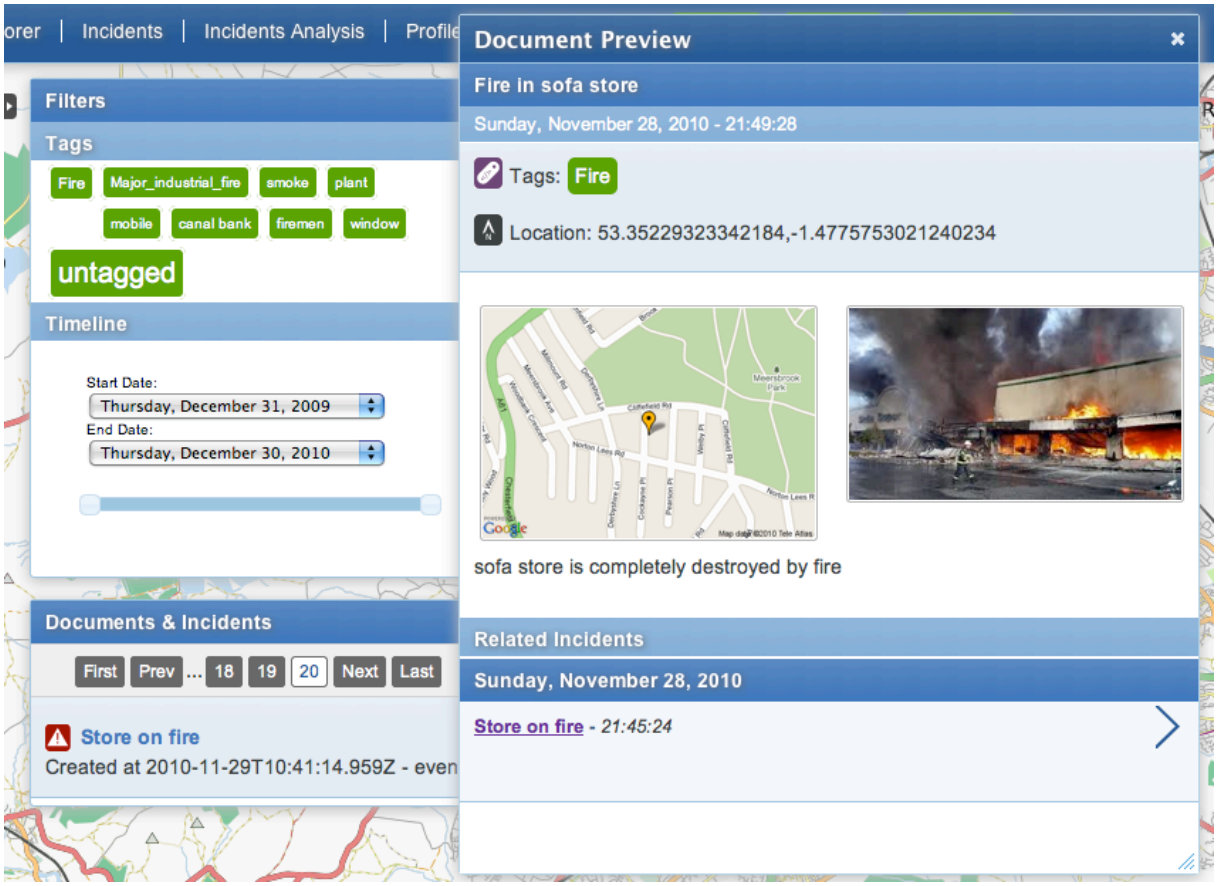


Figure 15 - Document Preview

Figure 15 shows an example picture uploaded to the WeKnowIt ER about a fire in a sofa store: the image is displayed alongside its location on a map – the tags ad the date and time are also visualised.

When a document is opened, the related ones (if existing) will be visualised at the bottom. For example in Figure 15 there is the possibility to view the related incident "Store on fire" by clicking on the link – the details about the incident will then be displayed (see Figure 16).

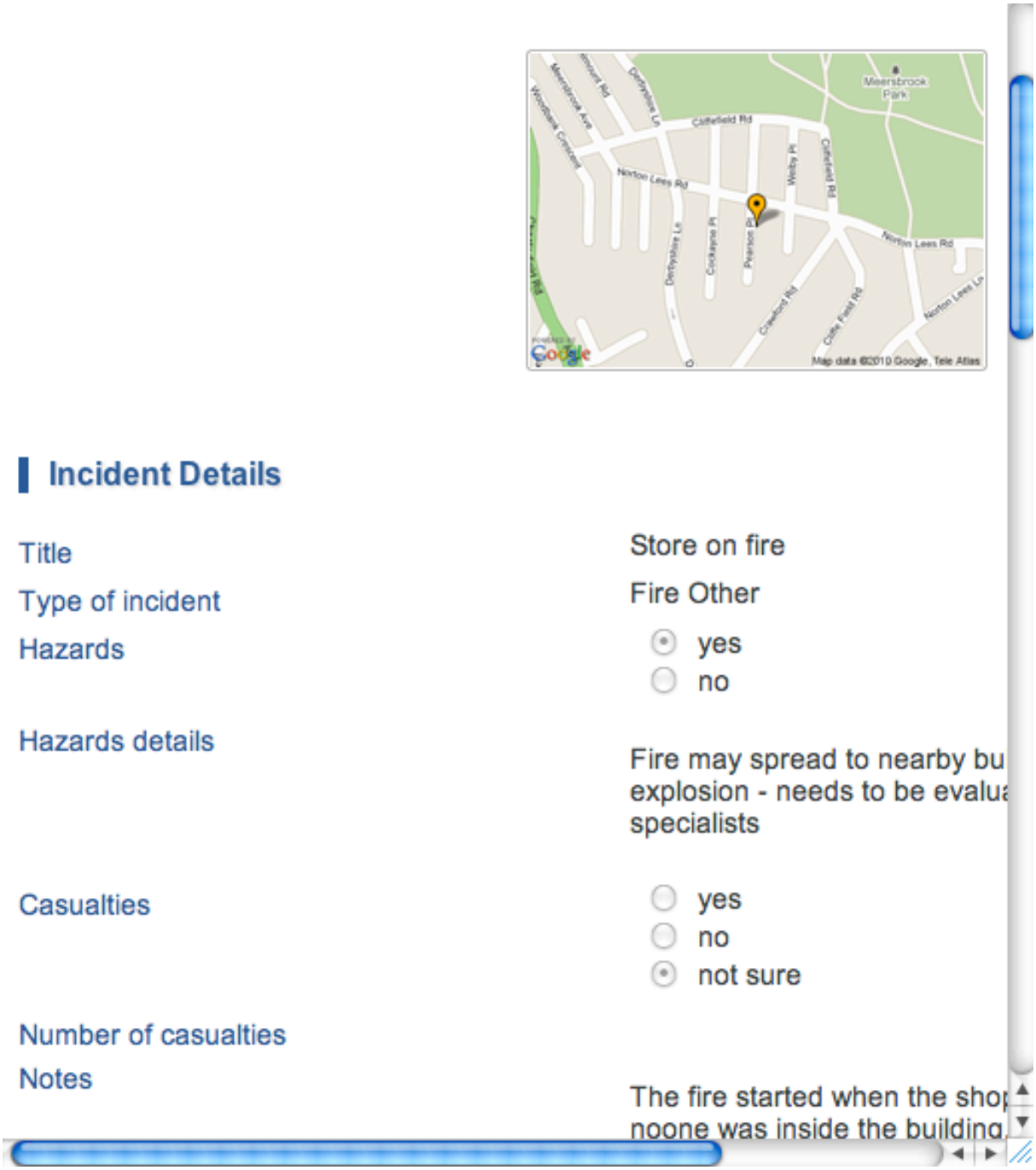


Figure 16 - Incident details

4.3.2.3. Incidents View

The incident view is a very simplified interface for browsing document in chronological order. They are display in list format, and the user can click on the desired document to view the full details (see Figure 17).

This view has been created to meet a requirement derived from the second stage user studies, where the ER team members expressed the desire for a simple chronological view of the events to use in alternative to the Explorer View.

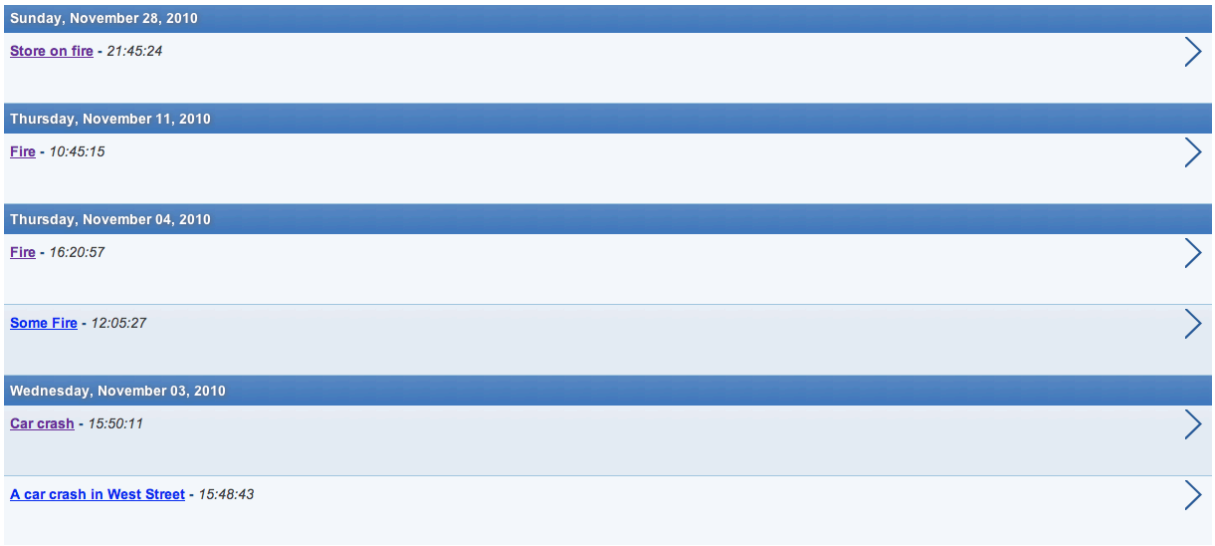


Figure 17 - Incidents View

In this modality a basic search functionality is available, to filter down the number of incident displayed accordingly to the keywords inserted.

4.3.3. Upload Data

The process for uploading images has been redesigned to streamline it and make better use of the intelligent features.

4.3.3.1. Upload Images

When an image is uploaded from the desktop interface the user is asked first of all to insert a title and a description (see Figure 18). This information is used by WP2 to extract information for two purposes:

- suggest possible tags
- extract geolocation (for example if a location name is inserted in the description it can be used for automatically geolocating the image).

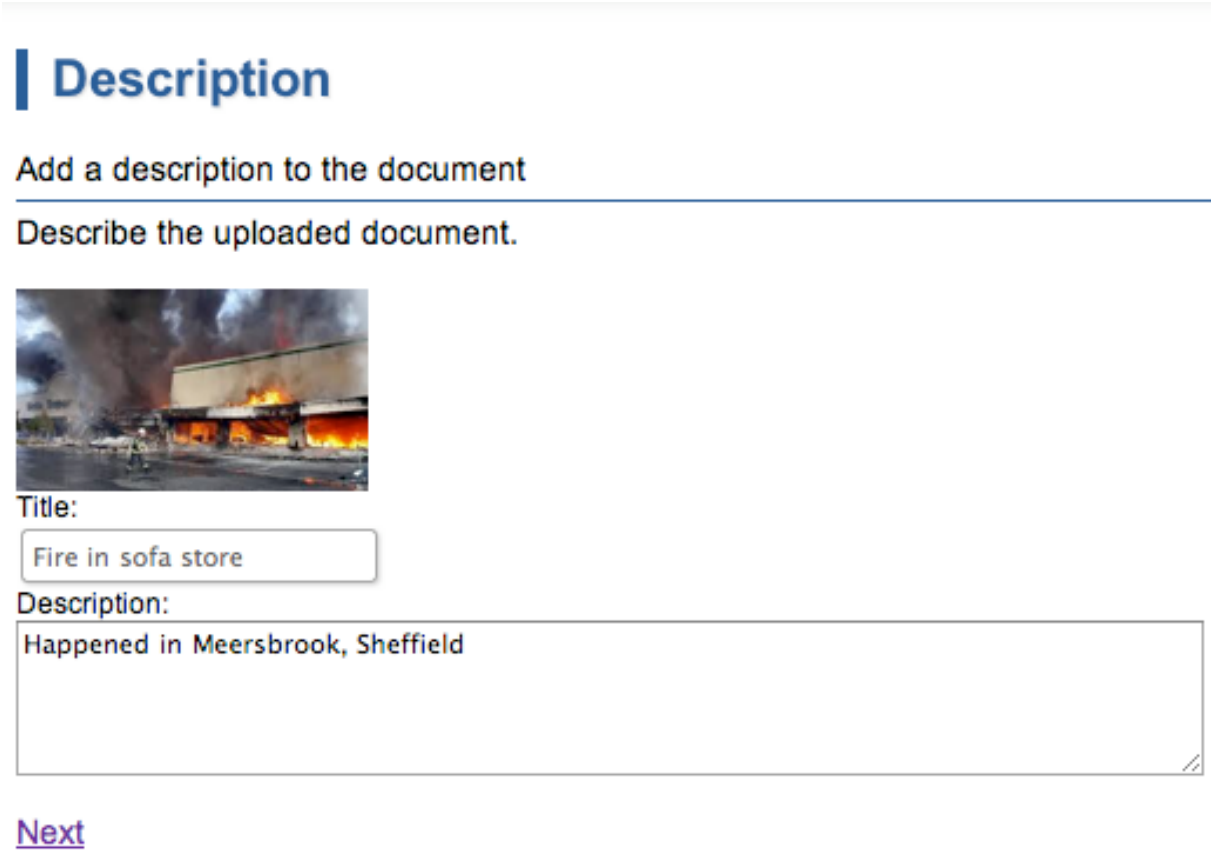


Figure 18 - Upload an Image Description

The images EXIF data are also extracted to help geolocating it. If no location information is available, a map is presented to the user to choose the image location.

4.3.3.2. Upload Incident Form

To meet the requirements expressed from ER members during user interviews and focus groups to be able to upload detailed information about an incident a form has been designed to collect the data. This functionality is directed to the organisational user, as only ER team members will be allowed to add a report about an incident and edit it.

The form includes a set of questions that are currently used by the ER Team members in an informal way to gather information when going to the emergency scene. The form has been validated with ER team members and after the first release new questions were added to better capture the incident details.

As the whole WKI system is based on semantic objects that interrelate, the form has been designed using K-Forms [1], an application initially developed in the OAK Group (University of Sheffield) for knowledge acquisition and sharing at the point of its creation. K-Forms uses semantic technologies (invisible to the users) to collect and store the data. When a

new form template is created, K-Forms seamlessly translates it into an explicit ontology (see Figure 4); the template concepts and fields and associated constraints are translated into OWL statements. Data input into forms based on those templates is transformed into RDF triples that can be searched using standard query languages such as SPARQL and SERQL.

The data collected using K-Forms can then be searched and shared using K-Search [2], a hybrid search application described in Section **Error! Reference source not found.**.

The form for collecting data is depicted in Figure 19.

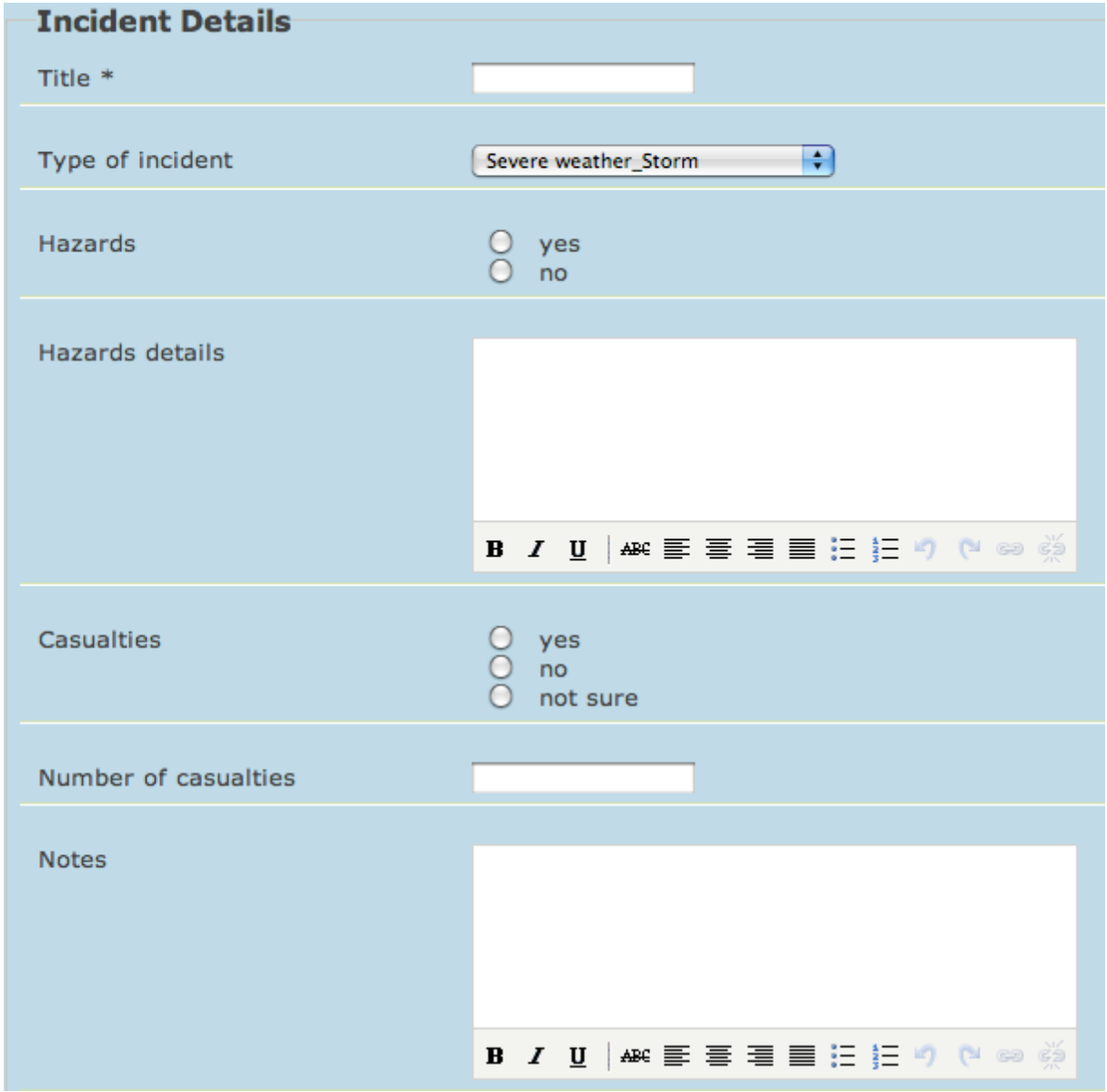


Figure 19 - Incident report form

After submission the form can be re-edit when needed to update the details, by simply clicking on the "Edit Report" link near a visualised form. This functionality is particularly useful to ER members, as often incidents evolve in time and new information is continuously available. The staff on the emergency scene could update the form when and of needed, and the updated data would be immediately visible to the office-based personnel.

A customised mobile version has been created for the mobile application, that takes into account the screen limitation and offers a quick way to geolocate information, as described in Section **Error! Reference source not found.**.

4.3.4. Messaging System

The messaging system allows ER users and the public to send short messages between desktop and mobile users. The intended purpose of the messaging system is for ER workers to send short messages to FLOs but it can be useful for regular users of the application as well.

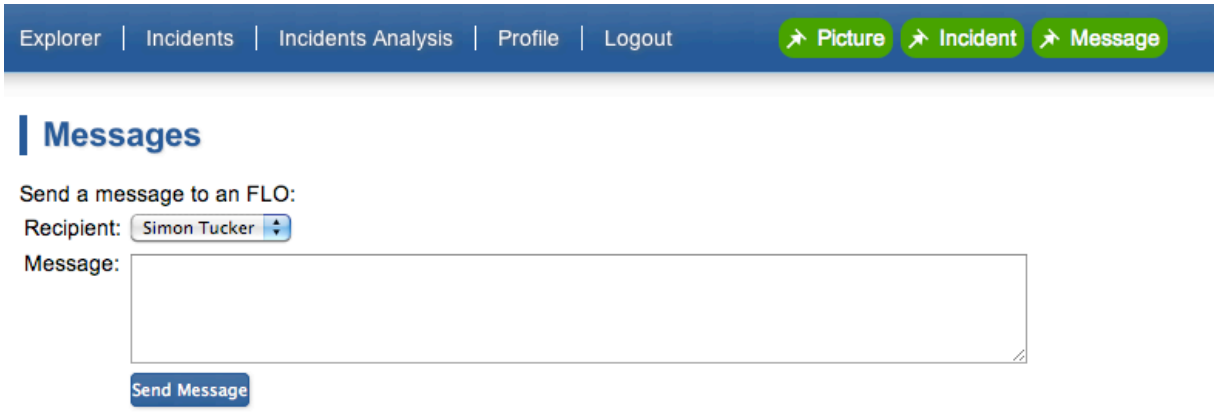


Figure 20: Messaging System

The user selects a recipient from the drop down list. Thus for an ER workers the recipient list could be a list of all the available FLOs or for a public user it could be a list of their contacts. The message is then added in the box below the recipient and clicking on "Send Message" delivers the message to the user. There is currently no interface for reading messages in the desktop system as the intention of the system is to deliver messages to mobile users at the scene of the incident. The mobile users are delivering information in the regular way and so there is less need for a fully two-way communication system.

The delivery of the message is shown for the mobile application below.

4.4. Mobile Application

The WeKnowIt demonstrator mobile application has undergone significant development since the previous version (see Deliverable D7.3.1 [10]). The previous mobile application used a simple native application to perform the capturing of an image and subsequent uploading of that image to the WeKnowIt system. This enabled the application to remain lightweight, and therefore straightforward to implement on different hardware platforms and still maintain a relatively high level of functionality overall. For the second iteration of the application we focused on building a more integrated native application whilst still keeping the connection with the external website in order to promote cross hardware functionality.

Whilst there are several frameworks which aim to bridge the gap between the rich web experience we expect on our desktops and the type of experience we can access on mobile browsers we did not feel that these

frameworks were of a level of a maturity suitable for use within WKI. Most significantly, at the time of development such frameworks were only able to support access to a small number of the sensors and functionalities present in most mobile smartphones. Thus the frameworks can access the location of the user via GPS but are unable to record audio or present natural interfaces for dealing with map data.

Given this problem we focused on implementing a demonstrator of how WKI functionality can be easily included in mobile applications by opening up selected functionality via the Desktop Demonstrator. Since it is undesirable to allow the outside world direct access to the WKI Composition Layer an extra layer of functionality was added to the desktop interface which acts a bridge between the Composition Layer and the mobile application. Security of this layer is handled by the mobile application which ensures that users have a legitimate account before giving them access to the functionality present in the WKI system.

This approach means that the native application is collecting and sending data to the WKI system but is able to present and gather that information using the sensors which are available to native application designers thereby combining the vital functionality. Whilst the bridge was designed with the specific application in mind it forms an API on top of which other applications can be developed and integrated within the WKI system.

4.4.1. Protoype Implementation of WKI Mobile Application

We focused on the Android platform for the WKI Mobile Application prototype. This decision was largely a result of the platform being conducive to radid prototyping and that it was a platform that already had some degree of penetration within the Sheffield City Council. Whilst the prototype described in D7.3.1 [10] was implemented for both iphone and android platforms, we focused solely on the Android platform for this version of the prototype in order to demonstrate the use of the Composition Layer bridge described above. The features and structure of the mobile application are described below.

4.4.2. Structure of the application

The mobile application replicates some of the functionality available from the desktop application as well as including some functionality which is unique to the mobile domain. In addition, the mobile application includes some of the intelligent services from WP1 which allows the mobile application to communicate its context to the WKI system and so the system is able to react accordingly.

4.4.2.1. Login

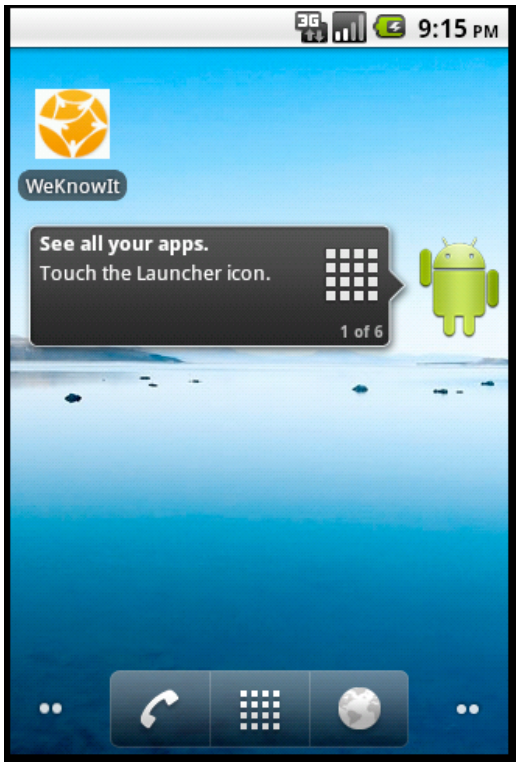


Figure 21: Application Screen

Upon starting the application for the first time the user is asked to login. They can do this by entering their opened in the box shown in the figure. Upon doing so the opened is validated and the user is taken to the main menu. If the opened is not validated an error message is shown and the user is asked to correct their details.

Figure 22: Login Screen

If the openid is validated then the WKI system will return the user id to the mobile application. This user id is then stored internally and used by the functions described below in order to ensure that a) requests are coming from legitimate users and that b) the user information is stored alongside any information that is provided to the WKI system. This also acts as a method of testing whether the user of the mobile application needs to login. Thus if the user logs in, then the next they access the WKI application they are taken directly to the menu screen until they select the log out option from the menu.

Figure 23: OpenID Entry Screen

4.4.2.2. Main Menu

The main menu gives the user access to all the functionality available within the application. In addition, the user is also able to log out of the application, returning to the login screen described above.

Figure 24: Main Menu Screen

4.4.2.3. Uploading Images

The upload of images follows a similar process to that described in D7.3.1 [10] but has been greatly simplified on the basis of the sensor information available to the user. In the previous version of the demonstrator the interface presented to the user was the same as that used in the desktop version of the application. For this version we have developed a specific version of the upload process for mobile devices. The upload process for mobile devices is simplified and accounts for the user context.

The process begins by the user taking a picture using a standard image capture interface. The user selects the button to take the picture and it is then automatically uploaded to the WKI system.

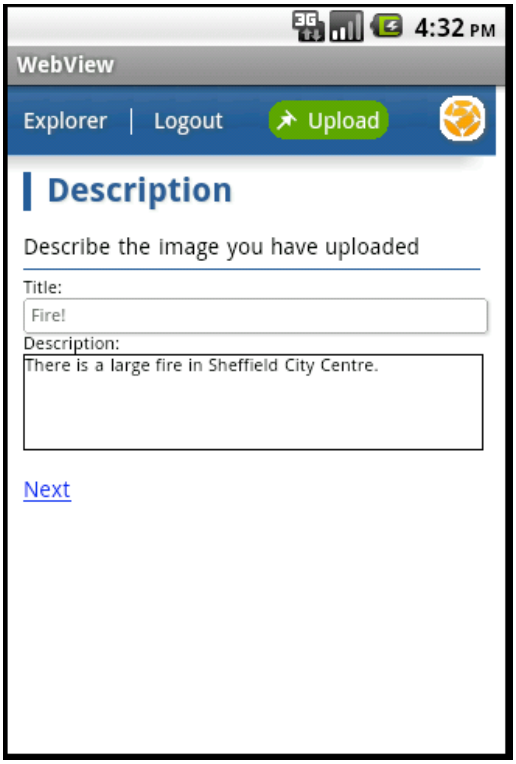


Figure 25: Describing uploaded image

Following this the user is given the opportunity to add a title and a description to the image. They do this in a screen similar to that shown in the desktop version but the interface has been designed to provide a better fit to the mobile screen.

Figure 26: Annotating the image

Once the user has entered the title and description information the mobile device sends the system the current battery level of the device. The WKI system then invokes the contextual modelling and uses this service to determine if the users battery level has reached a critical state. If the system detects that the power level is critical then it cuts the upload process short and automatically attempts to tag the image on the basis of the given title and description.

Thus the user is able to either conserve their battery life in times of emergencies or maximise their remaining power in order to add more image to the WKI system. If the user has enough remaining battery power then the system moves to a screen which localises their image and allows them to tag the image.

The user is able to tag the image on the basis of the text they entered in the previous screen. In addition, on the basis of GPS information taken from the phone sensors the image is localised and the nearest street name is shown to the user. If the phone was unable to localise the image then a map will be shown which will allow the user to manually locate themselves if they wish. If an unlocalised image is uploaded to the WKI system then it is automatically attached to an incident on the basis of it's temporal connection to that incident. This is the final screen for the upload process, after selecting next the image information is fixed and the user is returned to the initial menu screen.

4.4.2.4. Reading Messages

The user is also able to access their message from their mobile phone. The messaging system is lightweight and is primarily designed for workers in the command office to send messages to FLO although it can be used for other purposes. The user gets notified of any messages within the message bar of the phone. Thus the user is able to receive notifications when they are not using the application – the phone vibrating when the message arrives. The notification is displayed in the standard location within the phone.

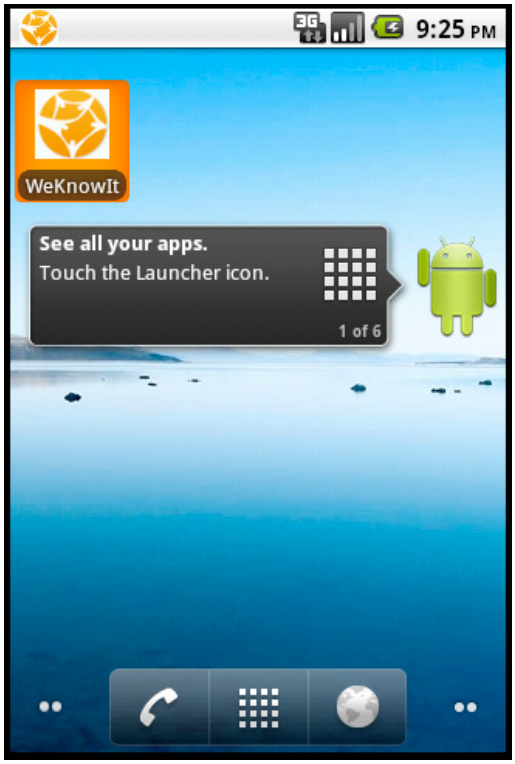


Figure 27: Message Notification (See top left of figure)

Upon selecting the notification, the user is taken directly to the message screen where they are able to see any unread messages displayed in read. The user is expected to read new messages straight away and thus there is no ability to organise or delete messages. This is intentional as the user will have more complex and structured messaging systems outside of the WKI system – the messaging system is designed to support very lightweight messages directed towards specific users. Once the user has selected the message they are able to see the full message text and the message is marked as being read.

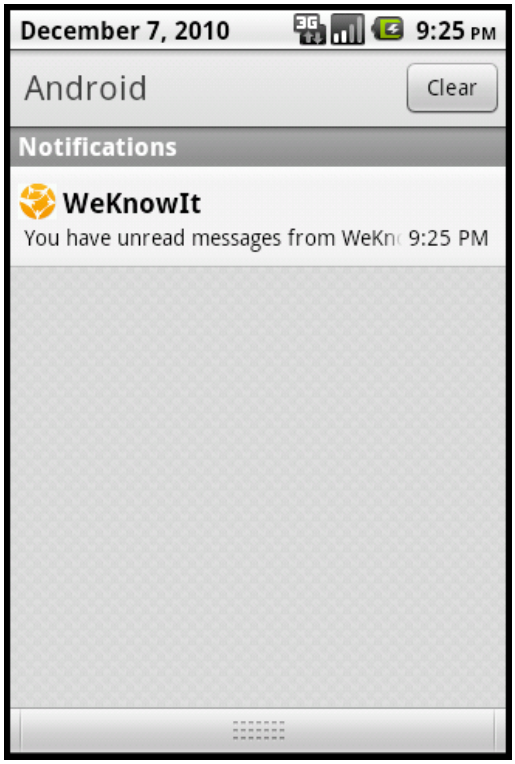


Figure 28: Message Notification (Full)

The mobile device polls for new messages every minute – thus ensuring that messages will be delivered within an average of 30 seconds.

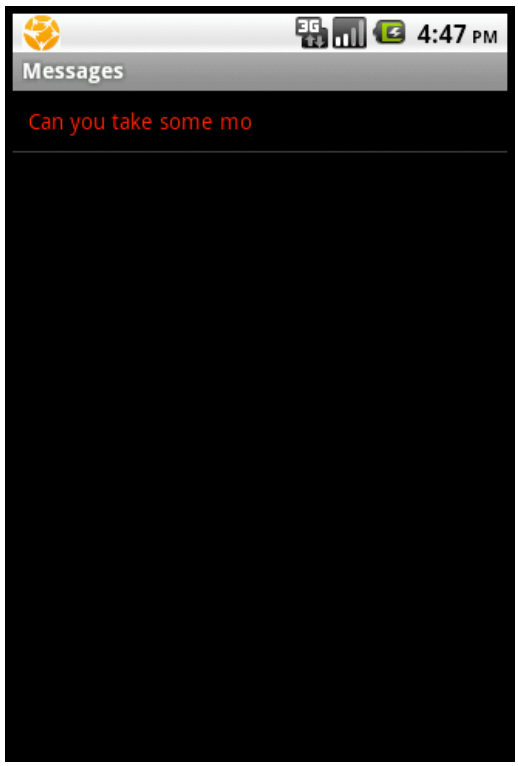


Figure 29: Message List

The user can then select an item in the message list to read the full message.

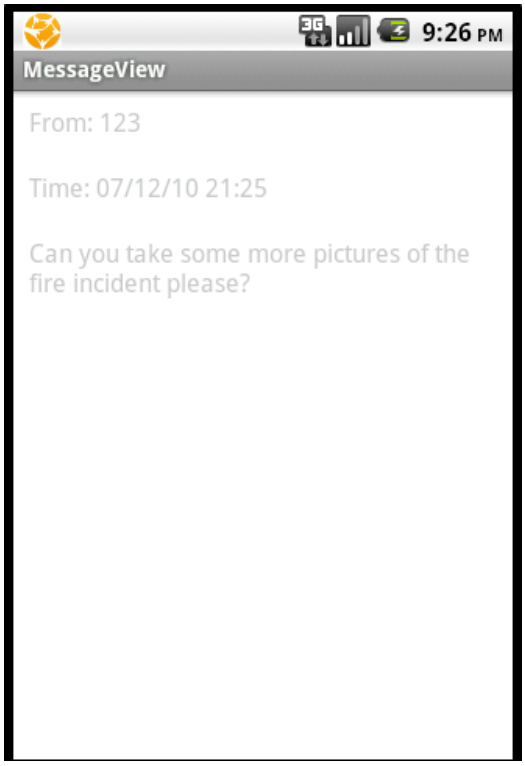


Figure 30: Sample Message

4.4.2.5. Seeing information in my location

The user is also able to see information relevant to their current location.

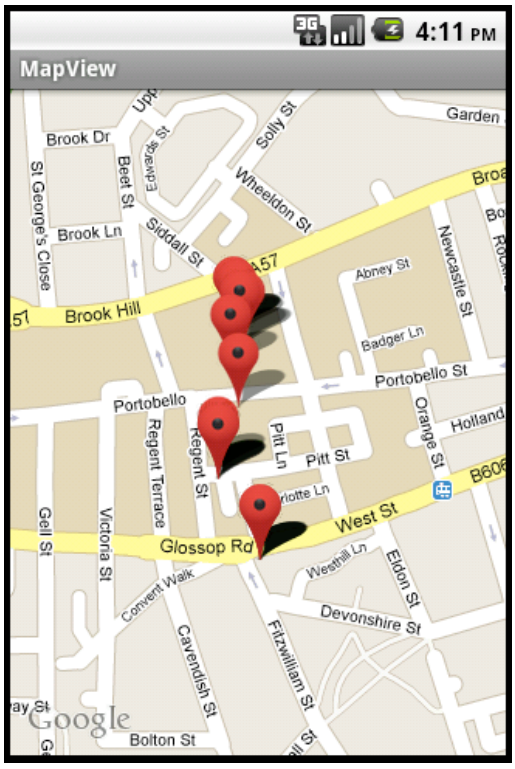
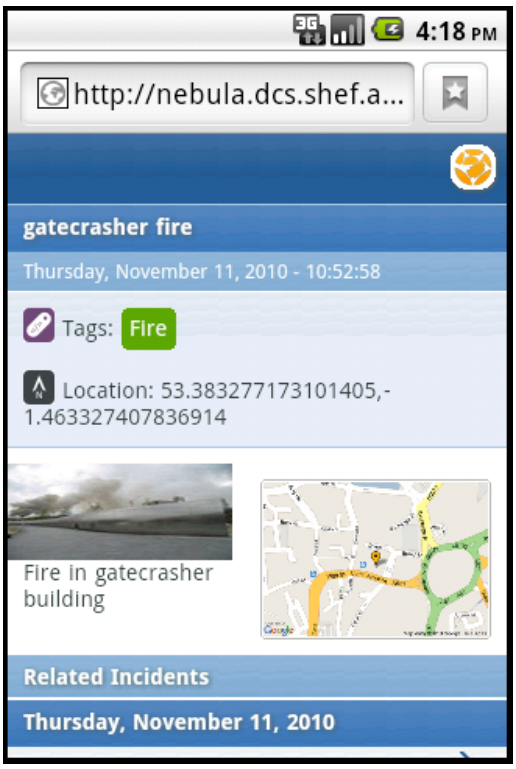


Figure 31: Map Display

The user's location is updated and the map adjusted to account for this. In addition new information is added to the map as it is added to the WKI system. Thus the user is always aware of the incidents and information which is relevant to their current location. The users are able to select any of the map pins to see more information. The information screen is taken directly from the website and therefore will mirror the display of information that the user expects from the desktop website.



4.4.2.6. Speech Upload

Selecting the speech upload button the user is able to record a short piece of audio.

The user selects record, speaks into their phone as normal. They then press stop to store the audio and upload to transmit it to the WKI system. The audio is uploaded and automatically tagged on the basis of speech recognition run over the speech recording

Figure 32: Speech Recording Screen

4.4.2.7. Add Incident

Finally the user is able to add an incident to the WKI system. Typically this functionality would be invoked by an FLO at the scene of an incident.

On top of standard multimodality-driven changes such as reduced size of the form, formatting changes to make the form more legible etc., the location information about an incident has been collected using a semantic typed form field that has a reverse geocoding service associated. A new field type, called location, has been created to define all the location fields in the form (Location, Access, Egress, RVP) – when the fields are displayed a reverse geocoding service (provided by WP2) is automatically called. This service takes as input the geolocation of the device (obtained using the geolocation properties of Mozilla in HTML5) and returns a series of geolocation expressed in humanly understandable terms: for example instead of returning the latitude and longitude of the location 5 possible street name surrounding the location are returned. The user can simply choose one of the available options (if accurate) by clicking on it (see Figure 33).

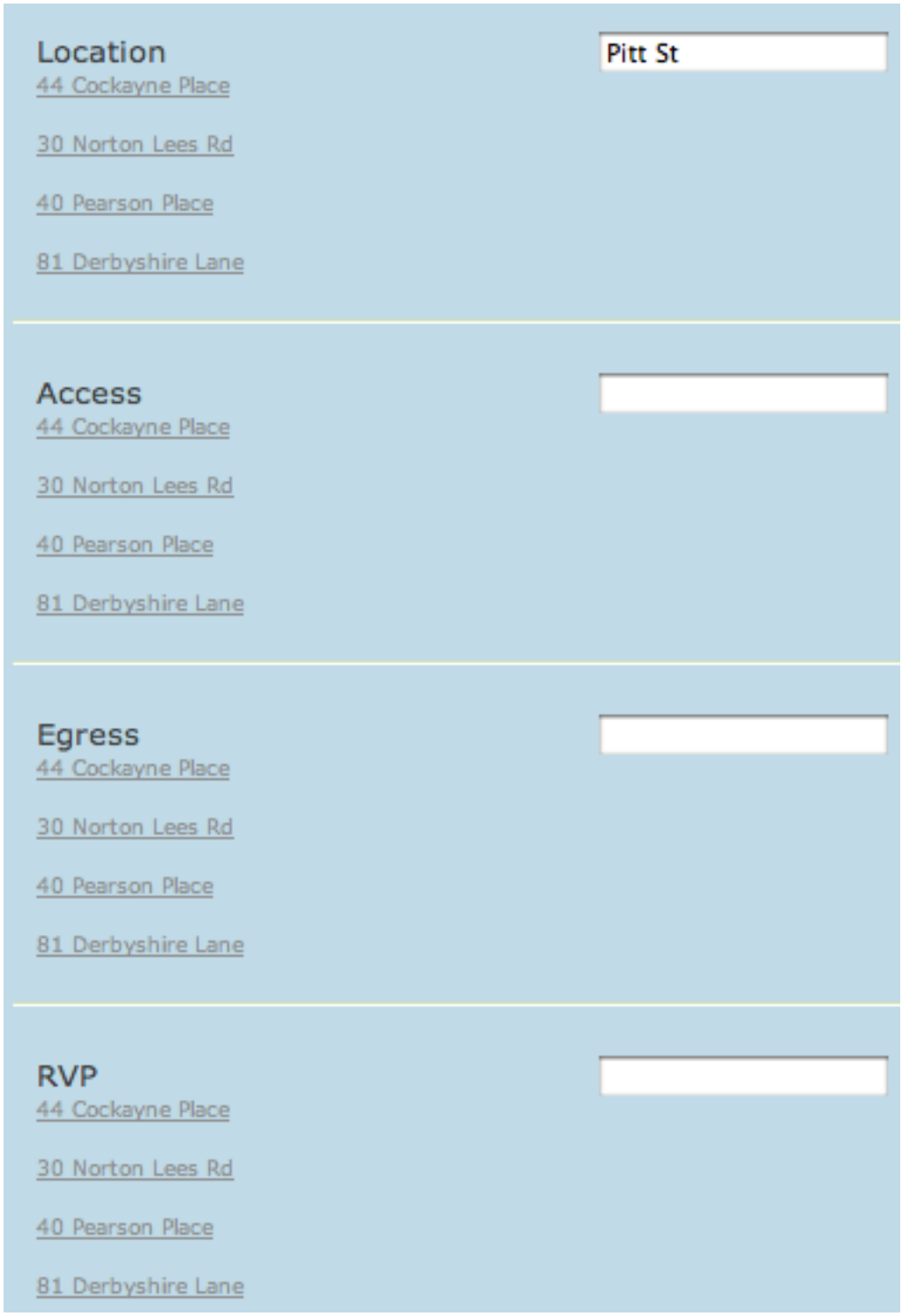


Figure 33 - Geolocation of incident form

This service is particularly useful for a mobile application as the limitations in size and the need to submit the incident details quickly call for an easy way to determine the location. The user can always insert manually a location if the suggestions are not accurate.

5. Post-Incident Management

An important functionalities highlighted by members of the ER team during interviews has been the possibility of reviewing data about an incident after it has happened both for debriefing and research purpose.

To support these requirements two different applications have been developed:

- Advanced Search and Data Analysis
- Post-Incident Management Box

In the following sections the two applications will be described.

5.1. Advanced Search and Data Analysis

This functionality allows users to search for legacy incident report forms and perform quantitative analysis, automatically generating graphs and charts over the results.

In order to implement this functionality an application previously developed in the OAK Group has been reused, called K-Search [2]. K-Search is a semantic tool for documents and knowledge retrieval and sharing that supports multiple, de-centralised, dynamic knowledge communities by enabling access to knowledge stored in multiple repositories (e.g. triple stores) with multiple ontologies.

The ontology associated to the incident report form created with K-Forms (see Section 4.3.3.2) is made available to K-Search, enabling hybrid search over the data (see Figure 34).

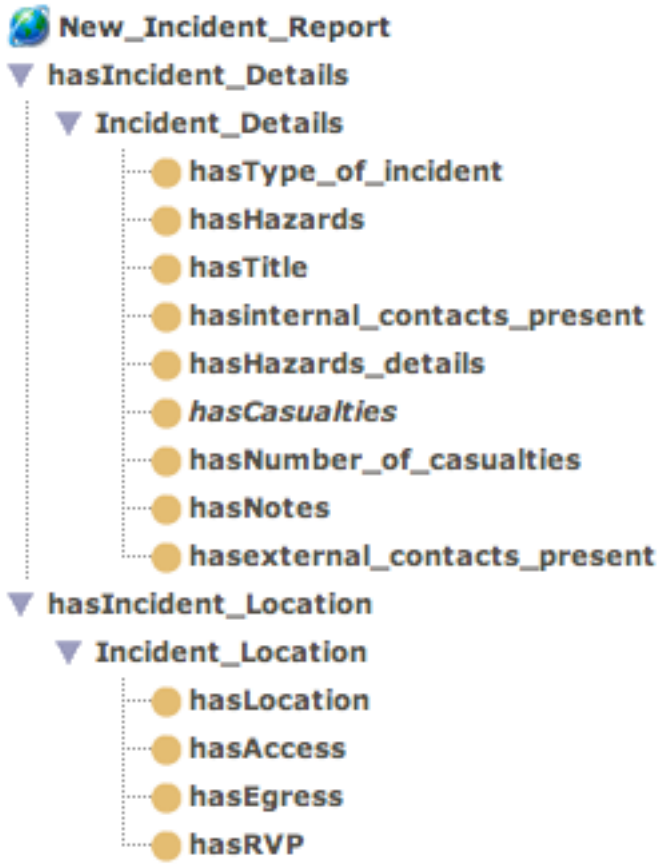


Figure 34 - Incident Report form ontology

Hybrid Search (HS) combines the flexibility of keyword-based retrieval with the ontology and its reasoning capabilities, making a synergistic use of both strengths. In HS, users can combine within the same query: (i) ontology-based search; (ii) keyword-based search and (iii) keyword-in-context based search. Keyword-in-context searches for keywords only in the text annotated with a concept in the ontology; in case the document has been created with K-Forms, it searches the content of the form field values associated to the concept.

The user can perform advanced searches by using all the concepts of the ontology, i.e. all the fields of the incident report form. This enables complex queries such as "Show me all the incident in Sheffield that are related to Fire and may have had casualties" (see Figure 35).

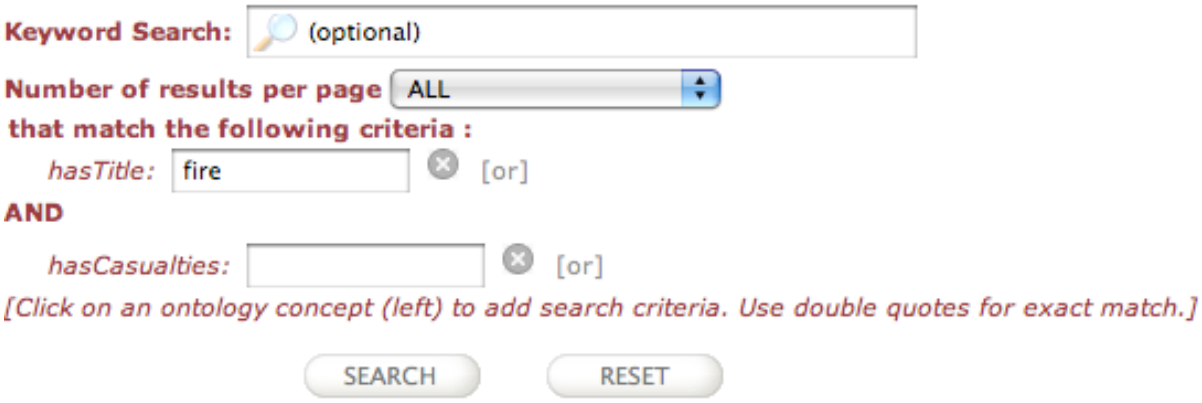


Figure 35 - Specification of search parameters

When the results are visualised (see Figure 36) it is also possible to perform quantitative analysis of the data, in the form of pie or bar charts.

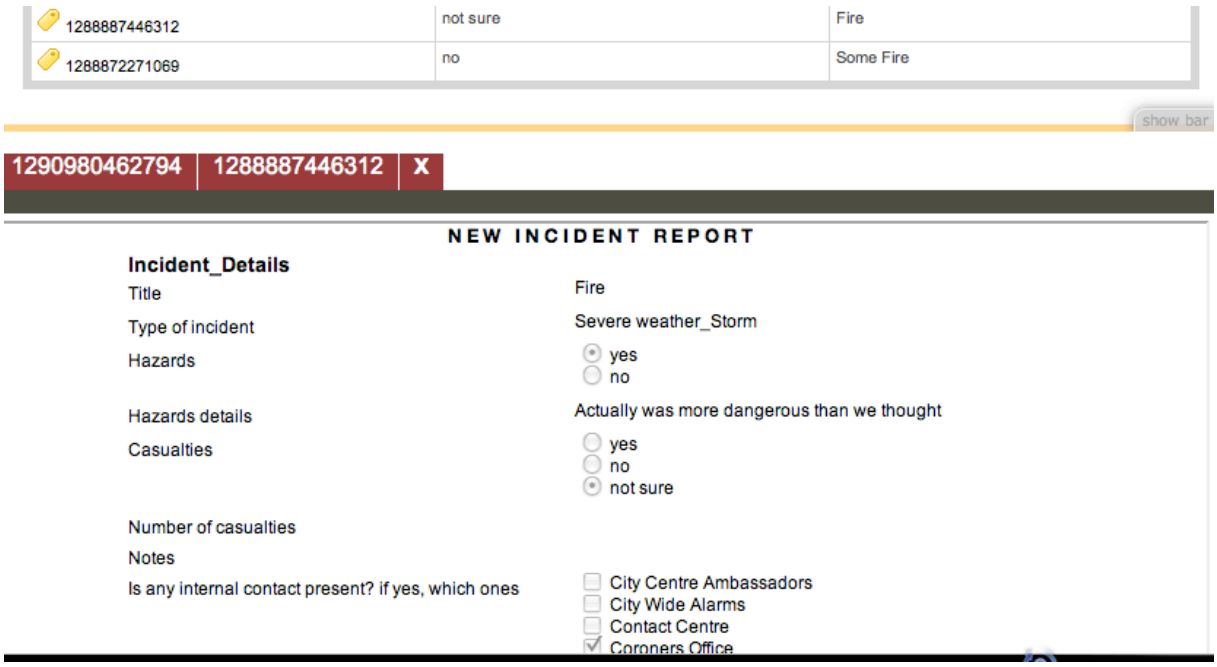


Figure 36 - Advanced Search Results

The user can simply specify the desired type of chart and the concepts to be used as dimensions (see Figure 37) and the chart is automatically produced (see Figure 38). It is also possible to export the chart as image or as CSV file to import in external documents.

Figure 37 - Select chart preferences

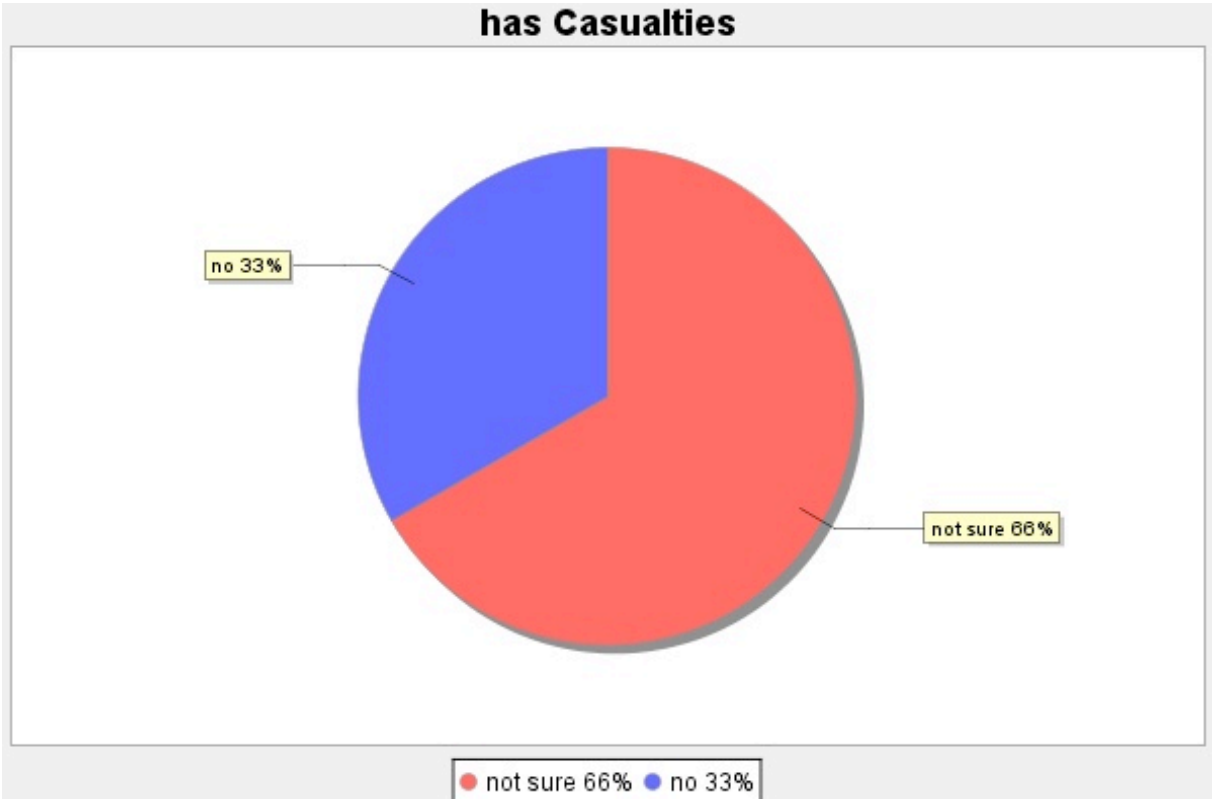


Figure 38 - Chart example

5.2. Post-Incident Management Box

After an incident happens, a more sophisticated type of analysis is related with debriefing and learning lessons, and it is typically focused around investigation to ascertain the root cause of an event. To investigate the root cause of an event, all the available evidence is analysed and reasoning is performed over the data. Often this task is shared by a group of experts; in the case of ER a group of ER team members is selected by the ER Team Manager to investigate the incident.

This is an example of using collective intelligence for organisational purposes, as the collaborative work performed by the members of the team is an investigation over the collective intelligence input that produces a collective intelligence enriched output: the input from ER members and users during an incident is enriched by the investigation and analysis process and turned into collective intelligence.

To support this sense making activity a tool previously developed in the OAK Group for sense making activity [3] has been adapted to the ER domain.

The Post-Incident Management Box offers the possibility to perform a root cause investigation of an incident, taking into account all the information available in the WeKnowIt ER application and giving the possibility to add new information.

The Post-Incident Management Box provides guidance through the root cause investigation process via several steps, each providing a different view on the current situation and related information.
Each step corresponds to a tab on the user interface (see the top of Figure 39: status, search, browse analysis and closure).

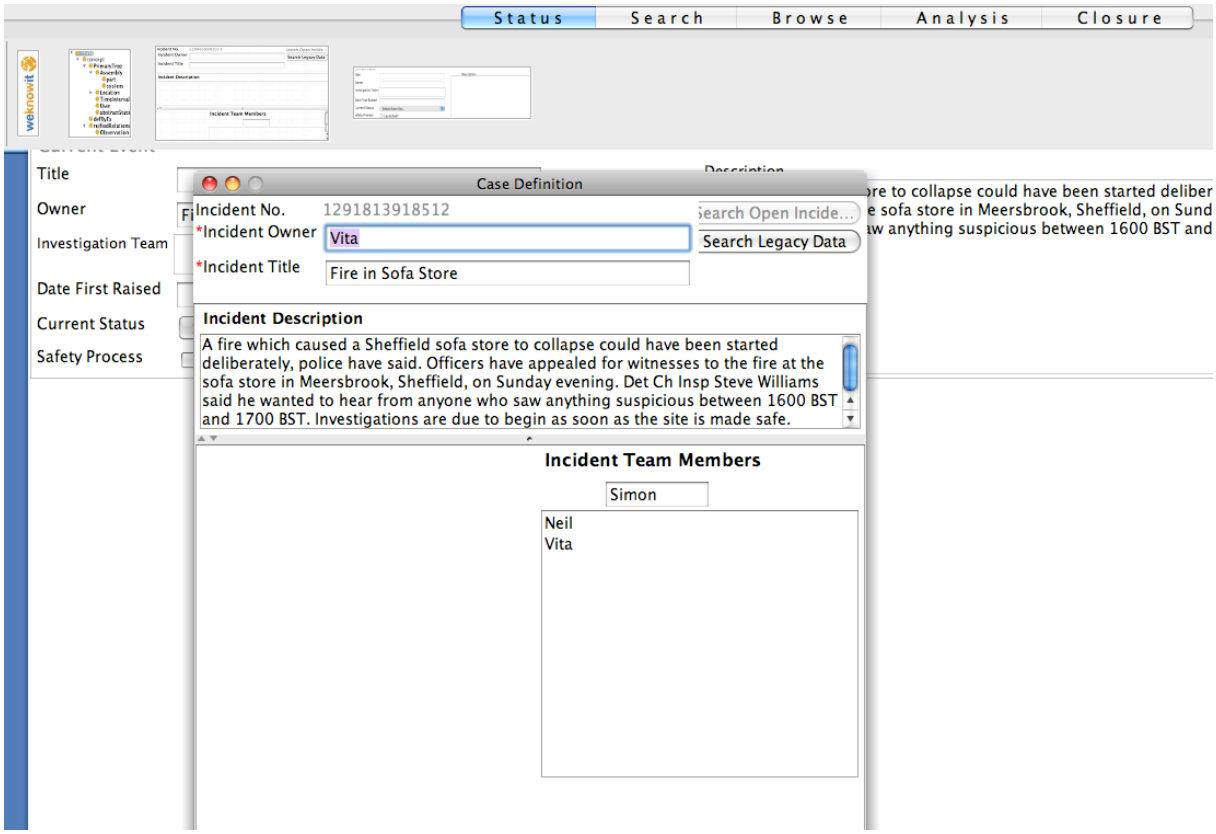


Figure 39 - Post Incident Management Box

The sense making process always begins by defining the purpose of the investigation – this can be done in the Post-Incident Box interface by

using the Case Definition Form. The status of the investigation is recorded using interactive tables (see Figure 39).

The search and browse steps offer the user the possibility to access external tools to perform searches, collect evidence and store it in the clipboard area. The search tab allows the user to open the Incident Analysis Search system (see Section 5.1) in a browser window, perform searches and drag the desired results in the clipboard. This is particularly important as it offers a way to bookmark data and share them with other colleagues working on the same case. The browse tab allows to open the WeKnowIt application in a browser window and again drag any desired content in the clipboard for future reuse and sharing.

The browse tab also allows users to create an maintain a workbook for the incident: the workbook automatically records all the bookmarked images and document placed in the clipboard and also allows to record additional comments, conversation etc. Evidence in the form of linked documents from the clipboard can be attached to any new comment/conversation by simply dragging and dropping it in the Attach box (see Figure 40). This again supports recording the organisational memory whilst helping users in rationalising and organising it.

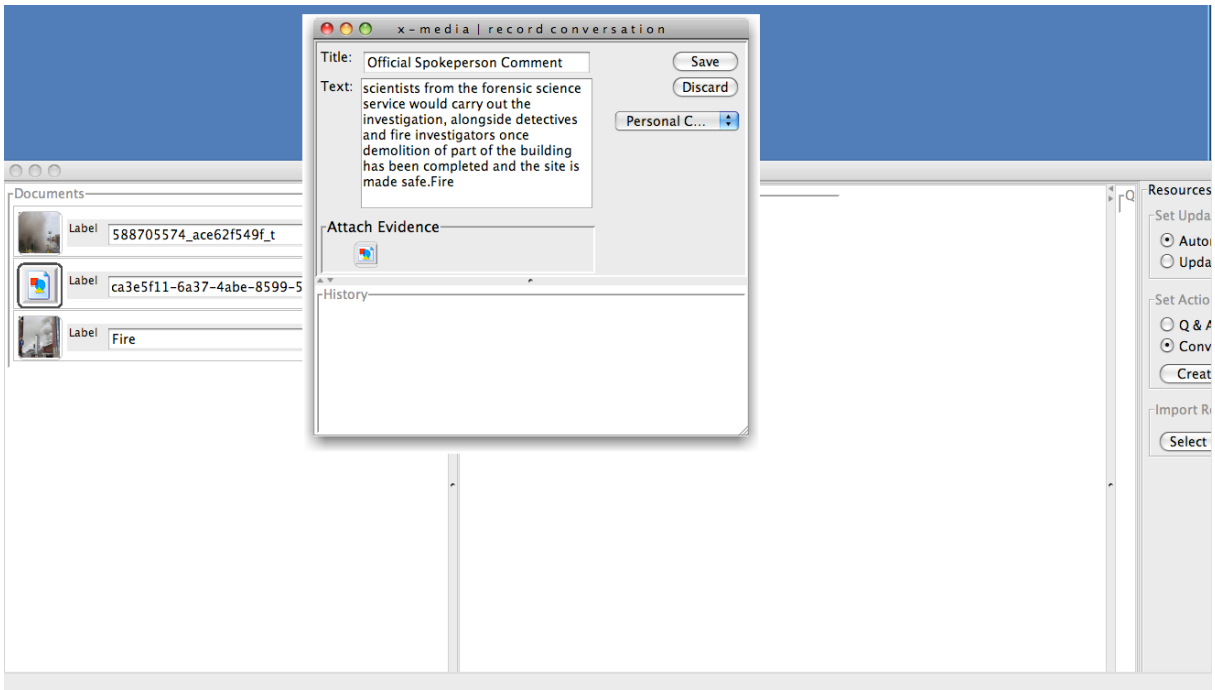


Figure 40 - Workbook

It is also possible to import resources from HD, in case a team members has some pictures or resources that have not been uploaded to the WeKnowIt ER interface.

While search is an activity carried out only when the user needs to collect new evidence, analysis, which is the core of sense making, is sustained

throughout the process. User understanding is built up during the systematic analysis of evidence and counter-evidence, and on the evaluation of which hypotheses explain the phenomena observed.

The Post-Incident Management Box captures this rationalisation process in the interactive analysis tree (see top, Fig. 2) that helps users formulate new and analyse existing hypotheses. The system supports users in building the analysis tree from suggestions in the domain ontology, focusing on the relation hasCause.

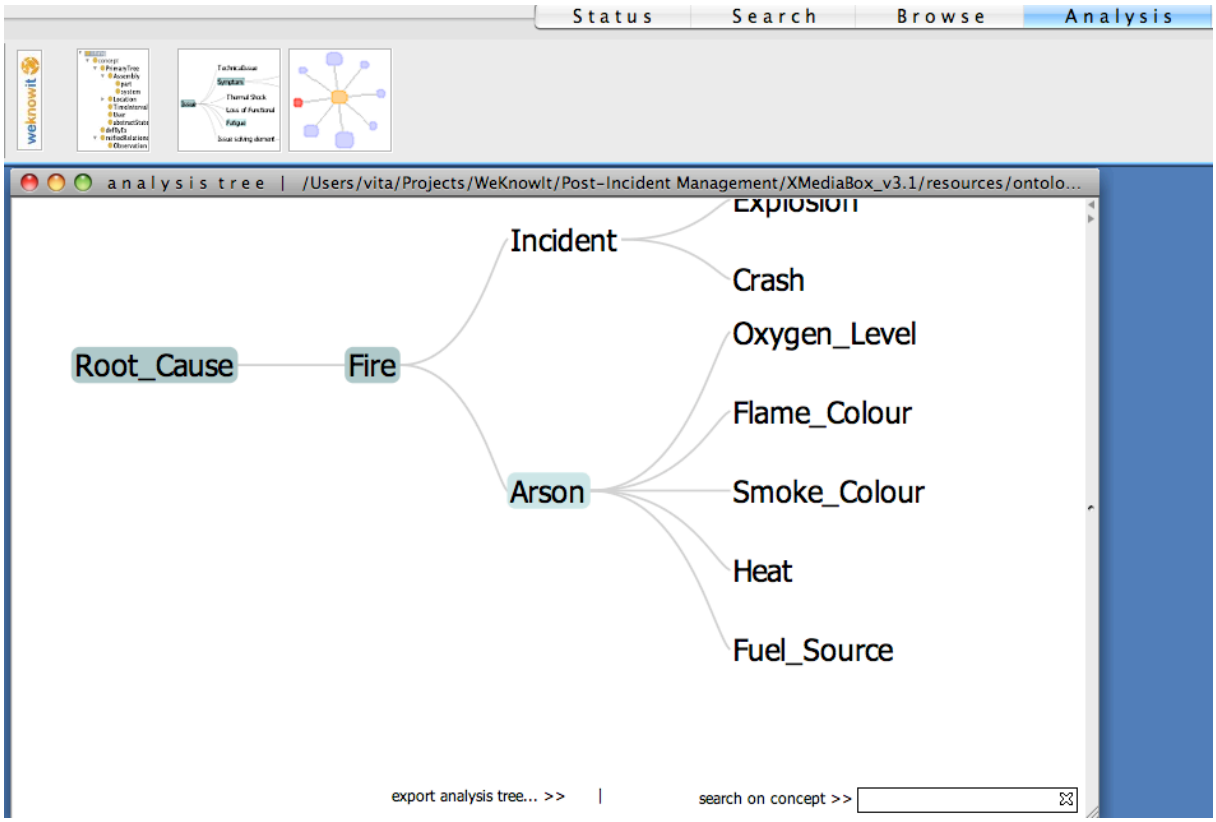


Figure 41 - Analysis tree

New concepts can be added to extend the ontology as and when needed, by using the appropriate option on the right-click menu.

Relevant documents, snippets of evidence and comments retrieved from the searching and browsing applications or created within the system may be attached to each (cause/hypothesis) node in the tree during the sense making process, creating an individual perspective on the hypotheses being investigated and the resulting knowledge space (see Figure 42).

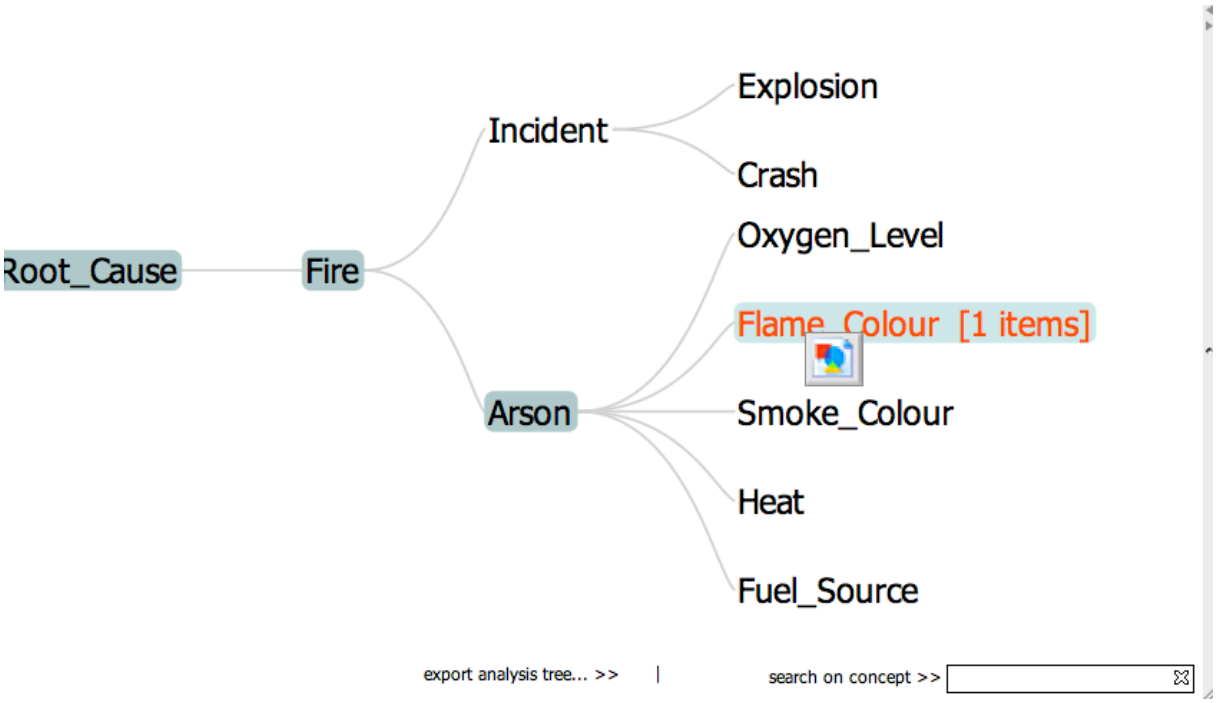


Figure 42 - Analysis tree with attached evidence

This view helps also pattern spotting, helping users determine which branch in the investigation contains the more valid hypothesis and which others need further investigation. Any node in the tree can be marked as "Root Cause" or discounted as root cause, offering a way to quickly reason on the data.

While the knowledge tree provides a structured view of the knowledge space for the current analysis, the knowledge graph provides a more complete overview of the relationships between the hypotheses (concepts of interest) and the evidence (see Figure 43). The knowledge graph serves as an alternative, less restricted view on the knowledge space, showing interconnections between evidence that may not be easily identified otherwise. It further supports the discovery of existing, but not yet identified knowledge during exploratory analysis, by providing a means for browsing the contents of the knowledge repository.

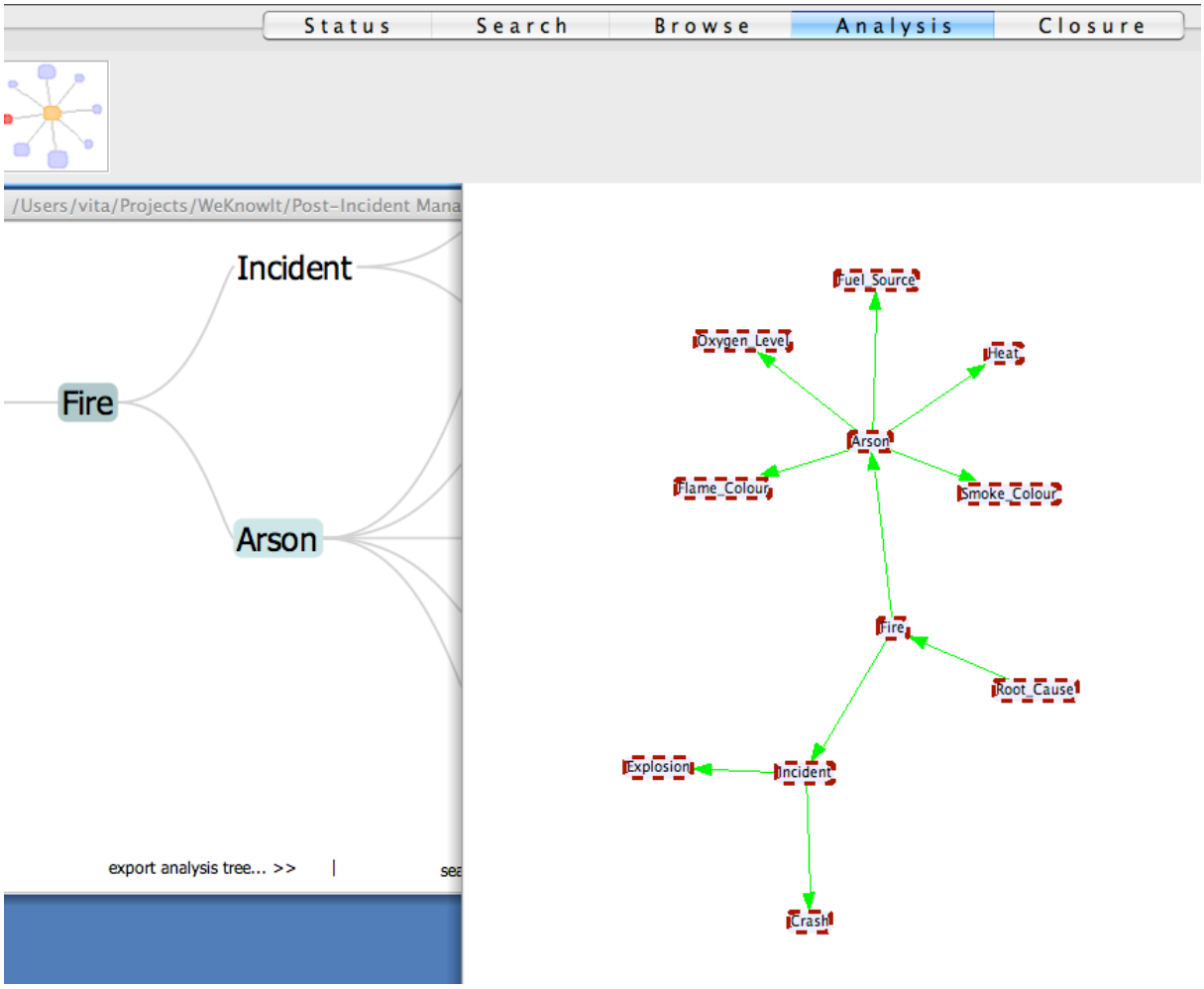


Figure 43 - Knowledge Graph

When the investigation is finished the closure tab offers a form to collect the results of the investigation, with fields such as Containment action summary, Solution Action summary, Lesson learnt (see Figure 44). The full investigation can then be exported as a report in HTML (or XML) format, including a picture of the analysis tree and the links to the additional documents used to prove evidence. This is particularly useful as it allows to create a copy of the investigation for reporting purposes, whilst still maintaining the knowledge in the session of the Post-Incident Management Box. The report can be updated when new knowledge is added or parts of the investigation are modified.

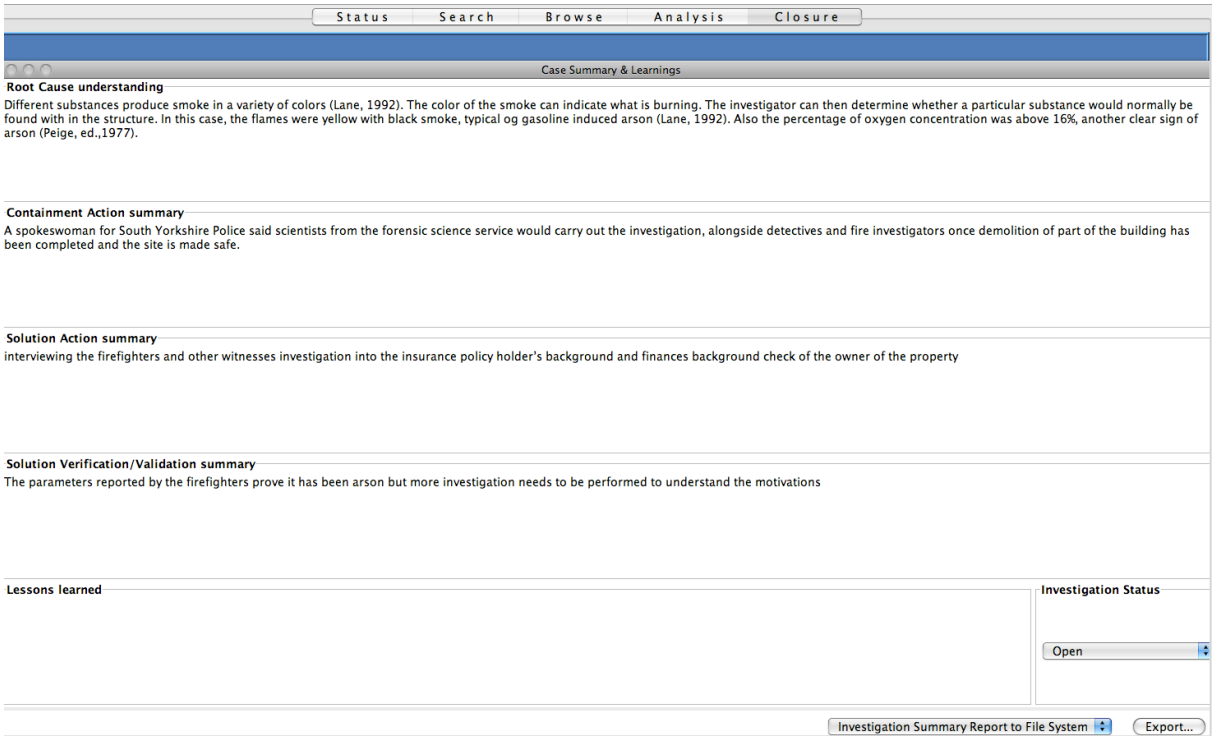


Figure 44 - Closure Report

6. Emergency Response Requirements

6.1. First stage Requirements

Deliverable D7.2 [9] outlined the requirements of the first prototype for the emergency response use case. These highlighted both functional and non-functional requirements that the first prototype demonstrator must meet. This section describes how the implemented final demonstrator meets the functional requirements described in that document.

ID	Description	Priority	How was achieved	Notes
UR-EF1	*Registration*	H	It is possible to register and access personalised functionalities.	
UR-EF2	*Support for multiple users and visualisations*	H	It is possible to filter info geographically and by time. It is possible to cluster documents	
UR-EF3	*Geo-located visualisation*	H	The information is geolocated and reverse geocoding services are available to facilitate the interaction.	
UR-EF4	*Information Control*	H	Information can be flagged as private.	implemented but not in the interface
UR-EF5	*Multimodal Interface*	H	A mobile and desktop a implemented	
UR-EF6	*Search functionality*	H	The system supports basic and advanced search.	
UR-EF7	*Content Upload*	H	Images, Audio files and Incident Report forms can be uploaded	

UR-EF8	*Tagging functionality*	H	It is possible to tag images, reuse suggested tags and browse using a tag cloud.	
UR-EF9	*Automatic gathering of metadata*	H	Services for tag suggestions have been integrated. EXIF data i.e. location are automatically extracted and assigned. Also geocoding services are used: for example if inserting street name in description of an image it will be recognised and the image will be geolocated using the extracted location.	
UR-EF10	*Login System*	M	Login is available using userID and Password or OpenID.	
UR-EF11	*Feedback*	M	Possible in the CURIO ontology and architecture.	Implemented but not in the final interface as not very relevant for ER domain.
UR-EF12	*Automatic gathering of relevant information*	M	Information has been automatically retrieved from external sources, extracted and stored in CURIO format so to be visible in the interface. Also a service has been created that allows to retrieve relevant images on-the-fly.	The historic information can be archived as available data set.
UR-EF13	*Support for big screen visualisation*	M	Possible as the interface is built in a scalable and flexible way.	
UR-	*Recommendations*	M	Tag suggestions are provided as well as	

EF14			similar images.	
UR-EF15	*Geo-location*	M	Geolocation is automatically extracted from images and from the device the user is using and associated to incident forms.	
UR-EF16	*Personal Network browsing*	M	N/A	Not relevant for ER domain.
UR-EF17	*Identify key people in a social network*	M	N/A	Not relevant for ER domain.
UR-EF18	*Communication Prioritization*	M	It is possible to switch communication priorities and visualisation using a parametric interface.	
UR-EF19	*User Profiling*	M	Implemented	
UR-EF20	*Messaging* The users shall be able to send and receive personal messages (i.e. email, texts) from/to other registered users.	L	It is possible to send messages to users on their smartphones (providing they have WKI application installed). The mobile will check for messages in background and alert the user when a new message arrives.	
UR-EF21	*Information Filtering*	L	Various filters are provided for information.	

6.1. Second stage recommendations and requirements

Following interviews with members of the Emergency Response Teams further requirements were highlighted

ID	Description	Priority	How was achieved	Notes
R1	*Improving Tag Filtering Interface*	H	The interface for tagging was improved by offering a list of suggested tags for the user to choose from or the possibility to enter own tags. On the browsing side The tag cloud was Implemented Following traditional Interaction paradigms.	
R2	*Evidence list should be restructured*	H	The evidence list is now part of the contextual filtering widgets and acts coherently with other filters. Also it displays different type of documents (incident reports, images, audio).	
R3	*Improve image visualisation for analysis*	H	Images from the map or the evidence list are now visualised as overlays, thus allowing to better analyse the images whilst keeping the visualisation context.	
R4	*Improve the connection between the various information displays*	H	The information filtering widgets have been redesigned to be context sensitive and offer a coherent view over the information. Modifications to one of the widgets are reflected in all the others, thus offering a holistic view.	

R5	*Improve image localisation*	H	The image localisation automatically extracting EXIF data or extracting location from image description
R6	*Improve Interface responsiveness*	H	The system has been redesigned to improve performances.
R7	*Introduce clustering technology*	H	A clustering algorithm has been implemented to group together documents in the geographical view.
UR-1	*Upload and edit Incident Information*	H	It is now possible to upload incident details using a form both from the desktop and the mobile application. Uploaded forms can be edited.
UR-2	*Advanced Search functionalities*	H	It is now possible to perform advanced searches over uploaded data.
UR-3	*Possibility to contact users on their smartphones*	H	It is now possible to message users on their smartphones.
UR-4	*Possibility to upload audio files*	M	It is now possible to upload audio files
UR-5	*Possibility to perform root cause analysis*	M	A new application has been developed to support root cause analysis investigations.

7. Conclusion

This document has described the implementation of the final demonstrator for the Emergency Response scenario. The Emergency Response scenario is exploring how Collective Intelligence can support individuals at the scene of emergency incidents and the organisations that are dealing with the incident.

The development of the demonstrator followed a standard user-centred development process, in two iterative stages. Following the first implementation a user evaluation was performed (reported in D7.5.1 [11]) and then the second stage involved two parallel processes of system development on one side and of user studies on the other. User interviews and focus groups were held every 15 days to both review the recommendations from the first stage evaluation and provide new requirements and feedback on the new prototype, in such a timely manner to be able to incorporate it in the final demonstrator version. This coupled with internal testing and review resulted in re-assessed requirements, recommendations and new requirements, that were addressed by the final demonstrator.

The goal of the demonstrator for the Emergency Response scenario is to allow both individuals and organisation to gather and upload information about an emergency (from mobile or desktop devices) offering advanced functionalities to enrich this information and turn it into Collective Intelligence.

The collected Collective Intelligence is then available for searching and browsing, using a multi-dimensional information space, that allows filtering the data based on location, time, tags etc. Services from Wp1-5 offered the technical basis upon which the final demonstrator was built, providing intelligent services that help and guide users during the upload and access processes, thus actually creating intelligence from information. The final demonstrator integrates services from the Personal, Mass, Social and Media Intelligence layers to collect and analyse inputs and produce a Collective Intelligence output.

On the basis of this collective intelligence, the ER personnel are able to make more informed decisions as to how to deal with the emergency situation. The ER personnel are also able to contact the members of the team that are on the incident scene to ask for more information or more details.

Whilst it is very important to upload and access Collective Intelligence during an emergency attention should also be paid to the importance of this intelligence in a debriefing phase. After an incident happened, it is often the case that ER personnel (or other forces) perform investigations about the incident, whether to solve an issue, introduce improvements, manage the consequences etc. All this activities can be summaries as

post-incident management, and support is required to reuse the Collective Intelligence collected during an incident.

To achieve this goal new applications have been provided for post-incident management:

- A new search application for advanced searches and quantitative data analysis over the uploaded information.
- A collective intelligence application for collaborative debriefing and investigation of incidents.

These new application become particularly important as they build upon the intelligence acquired and use the human active contribution and the cooperation between individual to create new intelligence that could be useful in case of future incidents. For example it is typical to create lessons learnt documents after an incident, that can be reused to better plan actions in future incidents or to prevent them. Using the tools provided by WeKnowIt it is possible to create this intelligence in a reusable semantic form that can get uploaded in the WeKnowIt repository and is available for search and reuse.

The development of the demonstrator has also shown that the WKI system developed within WP6 combined with the Composition Layer provides a suitable framework for building WKI applications.

8. References

1. R. Bhagdev, A. Chakravarthy, S. Chapman, F. Ciravegna, V. Lanfranchi, 2008, Creating and Using Organisational Semantic Webs in Large Networked Organisations. In Proceedings of the 7th International Semantic Web Conference, Karlsruhe, Germany.
2. R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi and D. Petrelli, 2008, Hybrid Search: Effectively Combining Keywords and Semantic Searches. In Proceedings of the 5th European Semantic Web Conference, Tenerife.
3. A.-S. Dadzie, V. Lanfranchi, D. Petrelli, 2009, Seeing is Believing: Linking Data With Knowledge. In the Special issue of the Information Visualization Journal on Human-Centered Information Visualization, 8:3, pp.197-211.
4. G. Hophe, B. Wolfe, 2003, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional, ISBN-13: 978-0321200686
5. J. Preece, Y. Rogers, H. Sharp, 2002, Interaction Design: Beyond Human-Computer Interaction. New York:Wiley
6. WeKnowIt – D6.1.2 - Identification of architecture elements and relations, version 2
7. WeKnowIt – D6.4.2 - Integration of WP 1-5 tools and common components version 2
8. WeKnowIt – D7.1 - Consumer and Emergency Response Use Case Initial Requirements
9. WeKnowIt – D7.2 - Emergency response and consumers' social group case study design and specification
10. WeKnowIt – D7.3.1 - Initial Emergency Response Case Study Implementation
11. WeKnowIt – D7.5.1 - Consumer and Emergency Response Use Case first evaluation report

Appendix 1 - Installation Instructions

Requirements:

Before starting the installation of the WeKnowIt system, you need to make sure that you installed the following tools successfully:

- The WeKnowIT UI server files
- Java 1.6 (**http://www.java.com/en/download/index.jsp**)
- Jruby (**http://jruby.org/**) + Jruby on Rails 2 (using Jruby)
- Jruby Gems
- WeKnowIt OSGI platform
- Tomcat (**http://tomcat.apache.org/**)
- Sesame (the Open Workbench and the SPARQL endpoint, **http://www.openrdf.org/**)
- Kforms + KSearch
- MySQL (for the User Interface, **http://www.mysql.com/**)
- PostgreSQL (for the OSGI, **http://www.postgresql.org/**)

Installation:

Before starting and configuring the WeKnowIt server, you need to start the WeKnowIt OSGI platform (cf previous delivereable) and install tomcat, Sesame (**http://www.openrdf.org/doc/sesame/users/ch02.html**) Kforms and MySQL on the same server if possible since it reduces the communication latency between the different system.

JRuby and Ruby on Rails:

After installing the previous tools, you need to configure JRuby. In order to do so, you need to make sure that Java is correctly installed and that you downloaded correctly JRuby on the official website. Depending on the platform, you can either install the Windows package or install it from source (if you decide to do it this way you will need to set the path variables correctly). For more information you can consult the documentation on the official website (**http://jruby.org/getting-started**).

After installing JRuby, you should be able to access it with a command line console by typing "jruby". Depending on the version you can check that the installation is correct by typing the following in a console window:

```
$ jruby -v
jruby 1.5.1 (ruby 1.8.7 patchlevel 249) (2010-06-06 f3a3480) (Java HotSpot(TM) 64-Bit Server VM 1.6.0_20) [x86_64-java]
```

Table 8 - JRuby installation check

You should check that JGem is also installed:

```
$ jgem -v
1.3.7
```

Table 9 - JGem installation check

If everything is correctly installed, you need to install JRuby on Rails. You can perform this task using ruby gem:

```
$ jruby -S gem install rails
JRuby limited openssl loaded. gem install jruby-openssl for full support.
http://wiki.jruby.org/wiki/JRuby_Builtin_OpenSSL
Successfully installed activesupport-2.1.1
Successfully installed activerecord-2.1.1
Successfully installed actionpack-2.1.1
Successfully installed actionmailer-2.1.1
Successfully installed activeresource-2.1.1
Successfully installed rails-2.1.1
6 gems installed
Installing ri documentation for activesupport-2.1.1...
Installing ri documentation for activerecord-2.1.1...
Installing ri documentation for actionpack-2.1.1...
Installing ri documentation for actionmailer-2.1.1...
Installing ri documentation for activeresource-2.1.1...
Installing RDoc documentation for activesupport-2.1.1...
Installing RDoc documentation for activerecord-2.1.1...
Installing RDoc documentation for actionpack-2.1.1...
Installing RDoc documentation for actionmailer-2.1.1...
Installing RDoc documentation for activeresource-2.1.1...
```

Table 10 - Ruby on Rails installation

The version number might vary, but you need to make sure that the installed version is the second version of Ruby on Rails.

The WeKnowIt UI Server:

After the installation of JRuby and Ruby on Rails you have to install the WeKnowIt server. First, you need to put the files of the server in a directory (lets call it <*WKI_UI_SERVER*>).

Before starting the server, you need to makes sure that all the dependencies are loaded, the easiest way to do it is to use the following command from the <*WKI_UI_SERVER*> directory*:*

```
$ rake gems:install
```

Table 11 - Gems install check

That command should install all the rubygems dependencies that are necessary for running the application. If you get an error saying that rake is missing, you can install it with the following command:

```
$ jgem  install rake
```

Table 12 - Rake install

This section closes the installation of the main WeKnowIt system. The next step is to configure the server and to start it.

K-Forms+K-Search

For installing K-Forms+K-Search Web application:

1. Create the ENVIRONMENT_ROOT directory. This will contain both the tomcat instance serving the web applications and the data it operates on: e.g.
   ENVORONMENT_ROOT/TOMCAT_DIR/webapps....
   ENVIRONMENT_ROOT/data/....

2. Install Tomcat 6.0 inside the ENVIRONMENT_ROOT directory (we should have a standard version of Tomcat which we use for this). It is recommended to use Tomcat core zip instead of windows installer if you expect to run multiple instances of tomcat on same machine, e.g. a webserver. If you only want to run one instance at a time (e.g. your personal laptop, then it is ok to use installer)

3. Deploy the following web apps to tomcat inside ENVIRONMENT_ROOT\TOMCAT_HOME\webapps: k-forms (create wars via Hudson from build.xml), k-cas, solr, openrdf-workbench, openrdf-sesame

4. Create ENVIRONMENT_ROOT\data directory

5. Create ENVIRONMENT_ROOT\data\solr-index directory containing any necessary index files and/or directories

6. Start hsqldb server:
   java -cp ../lib/hsqldb.jar org.hsqldb.server.Server -database.0 "file:ENVIRONMENT_ROOT\data\user_db\know_users" -dbname.0 know_users

7. Set the following tomcat server VM args
   -Dsolr.data.dir="ENVIRONMENT_ROOT\data\solr-index\data"
   -Dsolr.solr.home="ENVIRONMENT_ROOT\data\solr-index"

8. Start the tomcat server

Steps to change port numbers (for HTTP and HTTPS):

Default ports for K-Forms release currently is HTTP:9090 and HTTPS:8080. To change this for your installation, follow these steps:

1. Open Tomcat/conf/server.xml
When editing this find, please make sure you update the correct line, there are lot of lines with examples which are actually commented out. Changing those will have no effect.
Look for line that looks like this:

```
<Connector port="9090" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8080" />
```

Table 13 - Tomcat port commands

Update the port from 9090 to your new HTTP port number: e.g.:

```
<Connector port="6060" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8080" />
```

Table 14 - Tomcat port change

To update your HTTPS port, look for line that looks like this:

```
<Connector port="8080" protocol="HTTP/1.1" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS"
keystorePass="changeit"/>
```

Table 15 -Tomcat HTTPS port commands

And update the port to new port number, e.g. 6061:

```
<Connector port="6061" protocol="HTTP/1.1" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS"
keystorePass="changeit"/>
```

Table 16 - Tomcat HTTPS port change

2. Open Tomcat/webapps/k-forms/WEB-INF/classes/k-search.properties
Look for lines:

```
solr_url=http://localhost:9090/solr k-forms_url=http://localhost:9090/k-forms/
```

Table 17 - SOLR port

And update the ports to your new HTTP port:

```
solr_url=http://localhost:6060/solr k-forms_url=http://localhost:6060/k-forms/
```

Table 18 - SOLR port change

3. Open Tomcat/webapps/k-forms/WEB-INF/applicationContext-security.xml
Look for lines:

```
<logout logout-success-url="https://localhost:8080/k-cas/logout"/>
<beans:constructor-arg index="0" value="https://localhost:8080/k-cas"/> <beans:property name="service"
value="https://localhost:8080/k-forms/j_spring_cas_security_check"/>
<beans:property name="loginUrl" value="https://localhost:8080/k-cas/login"/>
```

Table 19 - Tomcat security port

Update with your new HTTPS port:

```
<logout logout-success-url="https://localhost:6061/k-cas/logout"/>
```

```
<beans:constructor-arg index="0" value="https://localhost:6061/k-cas"/> <beans:property name="service" value="https://localhost:6061/k-forms/j_spring_cas_security_check"/> <beans:property name="loginUrl" value="https://localhost:6061/k-cas/login"/>
```

Table 20 - Tomcat security port update

Configuration:

After the installing the WeKnowIt Server, you need to configure the Database, the Sesame Repository and set the address of the KForms server.

MySQL:

Before starting, you need to install MySQL. After the installation, you need to modify the database configuration file (*<WKI_UI_SERVER>/config/database.yml*) for matching your configuration (replace the information with you system configuration):

```
#database.yml
development:
  adapter: jdbcmysql
  host: <HOST>
  database: eresponse
  username: eresponse
  password: <PASWORD>
  port: <PORT>
```

Table 21 - MySQL configuration

The next step is to create the initial databases and tables. For doing this operation, you need to run the Ruby n Rails script that create the initial tables. From the *<WKI_UI_SERVER>* directory, run the following command:

```
$ rake db:create
```

Table 22 - Initial db tables creation

SESAME:

Before starting the server you will need to configure SESAME and KSearch. In this paragraph, we explain how to create a new repository using OpenRDF Workbench and configure it.

Before starting, make sure that you have installed SESAME and the Open Workbench successfully and that it is already running (**http://www.openrdf.org/doc/sesame/users/ch02.html**).

By default the server shoul be running on the port 8081, if you have installed SESAME and Open Workbench using the default directory, you should be able to access it using the following URI: **http://localhos:8081/openrdf-workbench** .

In the user interface, you have to perform the following actions:

1) Select New Repository
2) Select "Native Java Store" in the dreop down menu
3) In the ID field, type the name of your repository. For instance, "WKI".
4) The title can be left blank
5) Press next.
6) In the next section click create.

After creating the repository, you should see the following page:

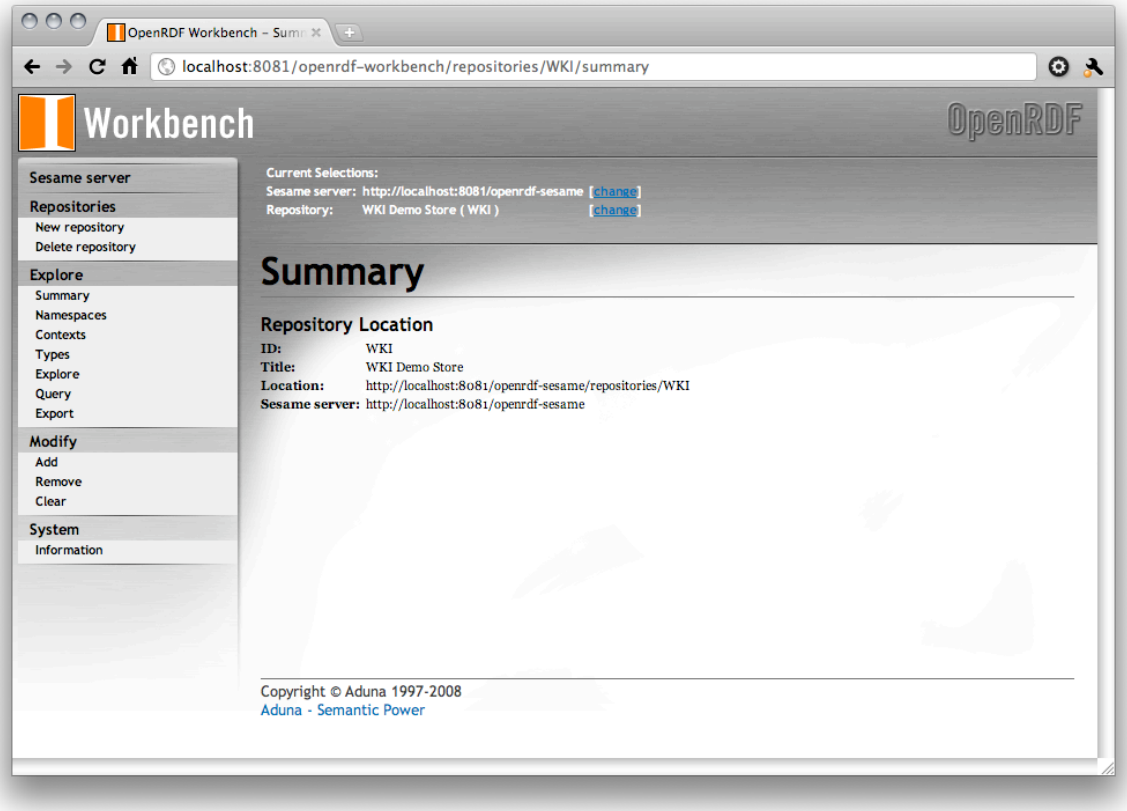


Figure 45 - OpenRDF workbench

In this page you need to copy the address of the repository (location). This URI will be required by the WeKnowIt server (we refer to this URI as *<SESAME_REPOSITORY>*).

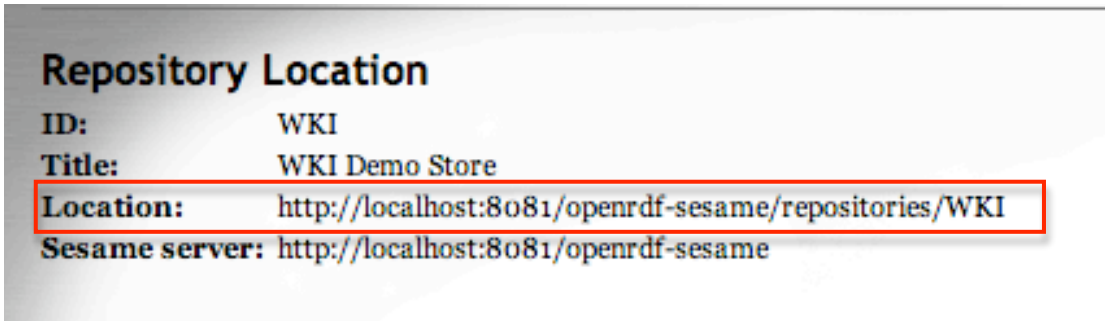


Figure 46 - Repository location

KForms+KSearch

You need to have the KForms repository installed on a server in order to use the WeKnowIt system.

You need to know the URI of the root of the Sesame repository used by the KForms server. This URI should look like: http://localhost:8081/openrdf-sesame/repositories/k-store (we refer to this URI as <*SESAME_KFORMS*>)). By default, the URI of the K-Forms server is fixed.

After creating the SESAME repositories, you need to configure it. In order to do it, you have to edit the following file: *<WKI_UI_SERVER>/config/config.yml*

```
#config.yml
development:
  rdf-repository: < SESAME_REPOSITORY>
  kforms-repository: < SESAME_KFORMS >
```

Starting the Server:

When the configuration is finished, you can start the server by typing (you must be in the <*WKI_UI_SERVER*> directory first):

```
$ jruby script/server
=> Booting WEBrick
=> Rails 2.3.8 application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2010-11-26 12:20:49] INFO  WEBrick 1.3.1
[2010-11-26 12:20:49] INFO  ruby 1.8.7 (2010-06-06) [java]
[2010-11-26 12:20:49] INFO WEBrick::HTTPServer#start: pid=1756075494 port=3000
```

As mentioned by the script, you can decide to detach by adding "-d" before starting the server.

Mobile Application

The android application is supplied as an application file and can be installed on a suitable phone using the standard method. The uploading application is compatible with versions greater than 1.5 of android.

The application can be download from the main WeKnowIt ER site at the address (see Figure 47):

http://nebula.dcs.shef.ac.uk/weknowit/download/android

or locally at the address:

http://localhost:3000/weknowit

If you access the page with an Android phone you can install it directly by clicking the link displayed on the page (you may need to allow your phone to install unsigned application first.).

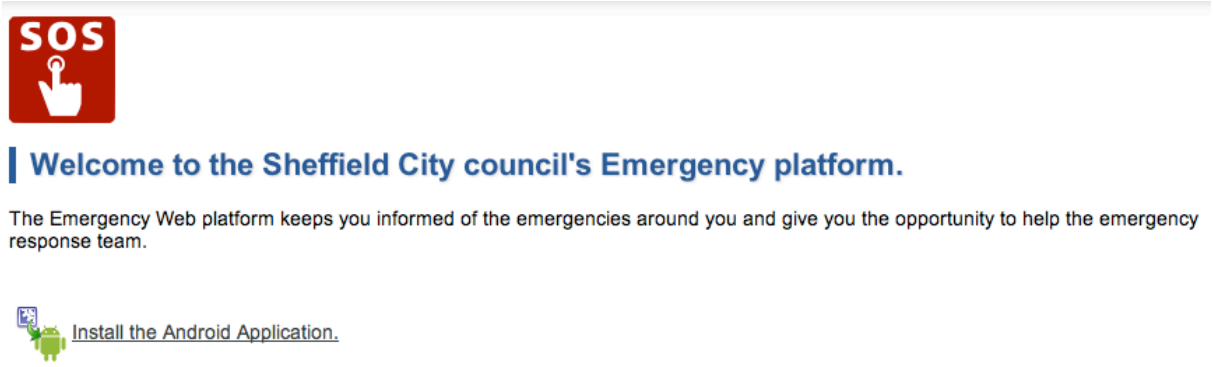


Figure 47 - Download Android application

The download file can then be executed directly on the Android phone, as shown in Figure 48.

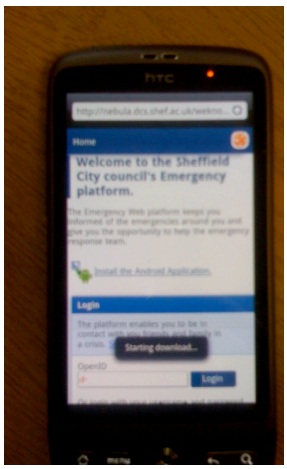


Figure 48 - Installing the WeKnowIt Android App from a smartphone

When the installation package is downloaded it can be executed directly just by clicking; the smartphone will ask the user's permission to use the internet and track the user's location (Figure 49) – if permission is granted the applications is installed.

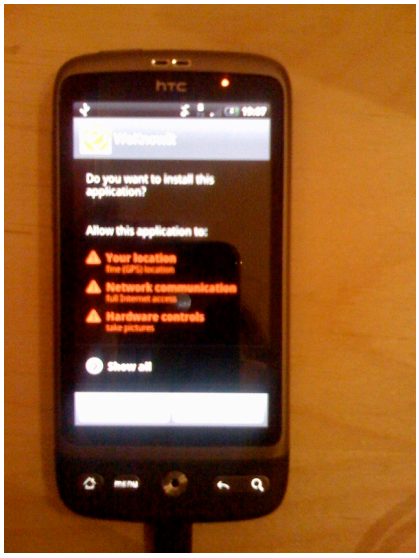


Figure 49 - Privacy settings for WeKnowIt Android App