



VATDIS web mapping

A report on the application of open standards and open architecture in geospatial interoperability for emergency management

Christiaan Logtmeijer, Paolo Isoardi, Bastiaan Schupp & Jean-Pierre Nordvik

EUR 23071 EN 2007

The Institute for the Protection and Security of the Citizen provides research-based, systems-oriented support to EU policies so as to protect the citizen against economic and technological risk. The Institute maintains and develops its expertise and networks in information, communication, space and engineering technologies in support of its mission. The strong cross-fertilisation between its nuclear and non-nuclear activities strengthens the expertise it can bring to the benefit of customers in both domains.

European Commission
Joint Research Centre
Institute for the Protection and Security of the Citizen

Contact information

Address: Christiaan Logtmeijer
E-mail: christiaan.logtmeijer@jrc.it
Tel.: +39 332 78 6727

<http://ipsc.jrc.ec.europa.eu/>
<http://www.jrc.ec.europa.eu/>

Legal Notice

Neither the European Commission nor any person acting on behalf of the Commission is responsible for the use which might be made of this publication.

***Europe Direct is a service to help you find answers
to your questions about the European Union***

**Freephone number (*):
00 800 6 7 8 9 10 11**

(*) Certain mobile telephone operators do not allow access to 00 800 numbers or these calls may be billed.

A great deal of additional information on the European Union is available on the Internet. It can be accessed through the Europa server <http://europa.eu/>

JRC 42703

EUR 23071 EN
ISBN 978-92-79-08110-1
ISSN 1018-5593
DOI 10.2788/6125

Luxembourg: Office for Official Publications of the European Communities

© European Communities, 2007

Reproduction is authorised provided the source is acknowledged

Printed in Italy

Summary

This report provides an overview of the web mapping activities carried out in the VATDIS action in 2007. These web mapping activities aimed at integrating the work done for the Orchestra, Floodsite and Preview IP into one single demonstrator.

In the main section of this report an outline is given of the reasons for integrating the projects and the contents of this integration. The report should be seen as a summary of the technical choices made in order to accomplish this integration. The annexes on this report help the more technical reader to understand the design of the demonstrator and to duplicate it for its own purposes.

1. Introduction	5
Objective	6
Structure of the document	7
2. Interoperability in emergency response	8
Use Case	8
Footprints and emergency planning zones	11
3. Demonstrator design	14
Proposed approach and software components	17
PostGIS	19
Geoserver	19
Mapbuilder	19
52 North web processing service	20
Axis	20
Tomcat	20
4. Implementation	21
Routing Web service	21
Creating a web service for a chemical dispersion model	24
Grabbing of spatial data	26
Possibility of calling all of the above within Google Earth	28
Comply with INSPIRE/OGC standards for interoperability, i.e. wrap services in SOAP	28
Demonstrate the possibility of extension with semantics	29
5. Result	30
Working demo	30
Outline of the tool	30
Creating footprints of hazards	30
Google Earth	35
SOAP	39
Semantics	42
6. Summary and Conclusion	45
Implementation:	45
Added value created	46
Difficulties encountered	46
Conclusion	47
7. Resources	48
8. References	49
Annex 1: Sequence diagrams	50
Annex 2: Used data	55
Annex 3: Routing service	56
Annex 4: Dispersion model	57
Annex 5: Google Earth	60
Annex 6: WPS code snippets	65
Annex 7: VATDIS and SOAP	75
Annex 8: Setting up Joseki	78

1. Introduction

Information technology could help us cope with hazards such as floods, chemical incidents and terrorist threats. However IT systems that are incompatible with each other make it hard for involved actors to collaborate and manage such risks effectively. In recent years developments have taken place that addressed these issues of collaboration. These developments have resulted in a number of technologies and standards that, once adopted and adhered to, should improve the interoperability between IT systems.

Geospatial Interoperability¹ is the ability for two different software systems to interact with geospatial information. This is a key issue in the VATDIS Action and in the integrated projects it has contributed to in 2007. These were: the ORCHESTRA, PREVIEW and FLOODSITE Integrated Projects, respectively on INSPIRE-compliant software architecture for risk management, on earth-observation for man-made risk mitigation in the context of GMES, and on evacuation management in the context of the new Flood Directive.

INSPIRE (Infrastructure for Spatial Information in Europe) is the name of a European directive on a spatial data infrastructure, a major undertaking of the European Commission to come to an infrastructure in which harmonized data will become available at large scale with the aim of improving risk management within Europe. Orchestra's initial aim is to liaison with this initiative and to explore what INSPIRE could mean in practice.

GMES (Global Monitoring for Environment and Security) is an initiative for the European Commission and the European Space agency that aims to implement information services that deal with the environment and security. PREVIEW (Prevention Information and Early Warning) liaisons with GMES by providing a number of these services from a number of risk domains, among which the man-made risk domain in which JRC-ISPC has a role by contributing to the development of a model for the atmospheric dispersion.

The aim of the EU Flood Directive is to reduce and manage the risks that floods pose to human health, the environment, infrastructure and property. According to the EU Flood Directive, Member States will have to reduce flood risk for those areas where the risk is deemed significant. This is done by determining the extent of flood risk (through flood hazard mapping and flood risk mapping). Consequently, objectives for flood risk reduction have to be established, as well as the measures that will be taken to reach these objectives. Floodsite contributes to this new flood directive by providing best practices, guidelines and models to assess flood-risk and demonstrate best practices in flood risk management.

The activities carried out within the VATDIS action for these integrated projects are the following ones:

- Flood evacuation management and routing in case of flash-floods, coming from Floodsite
- Facilitate faster and more precise dispersion forecasting for emergency responders (for use in chemical incidents), Preview

¹ Geospatial Interoperability (GI) Return on Investment Study Report , NASA, April 2005
<http://gio.gsfc.nasa.gov/docs/ROI%20Study.pdf>

- A pilot-system that implements an open service-oriented architecture (INSPIRE and other open standards compliant) for the evaluation of the vulnerability of trans-boundary road network to multi-hazards, Orchestra

The central theme within these three activities is emergency planning and response which provides an opportunity to integrate the results of these activities.

Objective

The objective of this study therefore is to demonstrate this integration by drafting out a use case on interoperability of geospatial data and information systems to support emergency response and risk management through open service-oriented architectures and software standards.

The main open standards that will be applied are the Open Geospatial Consortium² (OGC) standards³. These are de facto standards developed for the web-based dissemination and processing of geospatial data. These standards prescribe e.g. how maps can be published on the web (WMS) or how the data can be exposed (WFS) or how remote servers perform geospatial operations (WPS).

With the introduction of standards like these, the possibility arises to effectively work in a distributed manner. In such a distributed environment, which is called a service-oriented-architecture (SOA), information systems are able to 'talk' to each other and are able to provide services; packages of business processes. The OGC standards provide the glue between the diverse business processes and integrate these processes by effectively supporting geospatial interoperability.

Within the context of this Orchestra, Preview and Floodsite interoperability study, the management of emergency situation was chosen as a demonstration case for our work. In emergency situations emergency response organizations are confronted by a stream of information regarding the situation they must deal with. At a command centre, this is usually integrated in a command and control tool which integrates many sorts of information; including information about the hazard, the disaster area, resources, and procedures. Much of this information is of a geographic nature or needs to be analyzed in a geographic context.

Actors involved in an emergency response need to exchange this information in a time constrained situation. Often multiple organizations must work together and sometimes in an ad-hoc manner. Therefore data exchange and integration must be streamlined. This makes emergency response a good demonstration case; if interoperability can be demonstrated in an emergency situation, it will probably work as well in more routine risk management processes.

Given the above objective, foremost goal of this project is to demonstrate a working example and to document how it works in practice, including some of the choices that were made to achieve this, and to obtain better knowledge of the issues that need to be resolved to achieve further improvements.

An important limitation to the work described in this document is that it only concerns a first demonstrator of the basic concepts of interoperability, which does not go beyond some

² <http://www.opengeospatial.org/>

³ <http://www.opengeospatial.org/standards>

primary functions required in emergency response. It cannot be seen as an operational tool at this point in time. There are several issues that are not yet covered here, but which are required to be addressed before these concepts reach maturity and become operational in emergency management.

Such issues include the dependability of the services which must be highly dependable while some may even prove safety critical. This means there must be a high level of assurance that services deliver according to specification whenever required. Secondly, there are issues of rights management and confidentiality in real world applications which must be managed. Third, many organizations in emergency response already depend on existing command and control tools and are governed by local or national regulations that impose limits on how they can use new ways of working.

Structure of the document

This document has the following structure: The next chapter describes the use-case in more detail, and discusses the various elements of the different projects that this study is based upon. In chapter three the architecture and implementation of open standards is described. Chapter four describes the main challenges and problems encountered during the implementation of this study. The annexes provided with this study clarify in more detail and code snippets on how some essential parts of the demonstrator were built.

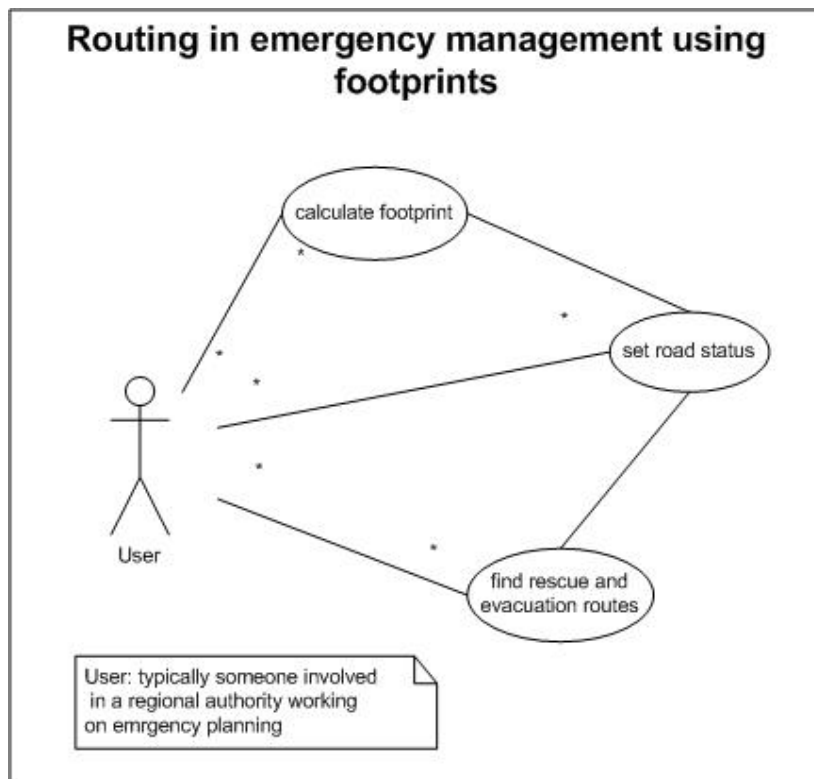
2. Interoperability in emergency response

Use Case

The use case that is introduced here builds upon the routing model developed for the Orchestra IP and feeds this routing model with the output of scientific models that determine the geographic extent of disrupting events, these outputs come from Floodsite and Preview on respectively flash flood forecasting and chemical dispersions.

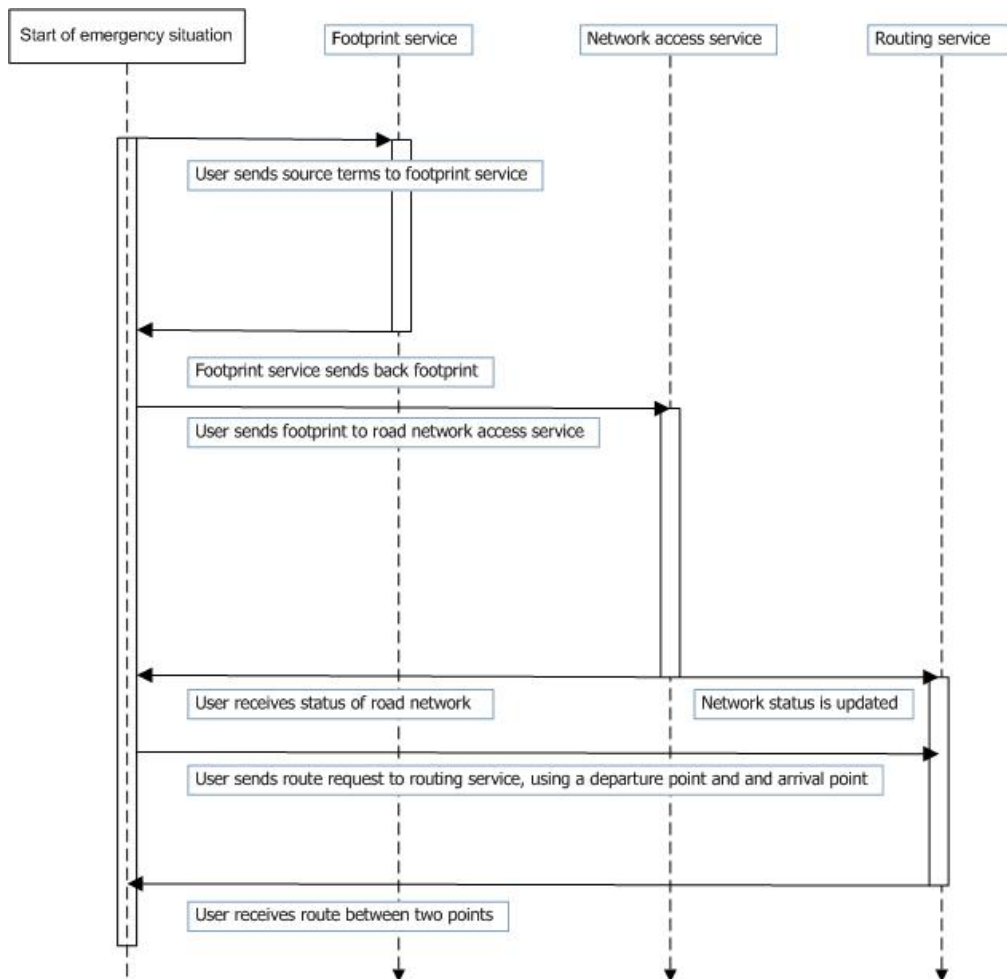
This use case makes use of several independent services; exactly how a user interacts with all these services can be outlined by the following:

1. An emergency situation starts and the user obtains knowledge on the source terms of the disaster that causes the emergency situation and sends these to a service that calculates a footprint of that disaster. The user retrieves this footprint back.
2. After having received the footprint of a disaster the user determines the most relevant elements of the disaster and outlines an emergency planning zone based on the footprint and his insight.
3. The user combines this footprint with the road network and requests a service to set the road status of the roads concerned in the disaster planning to unavailable.
4. The user uses this road status to further plan the emergency situation by e.g. planning routes and planning emergency zone exits, finding close by facilities like fire brigade stations, police stations and hospitals.



2-1 Use case on routing for emergency management

Figure (2-1) gives an overview of the relationships between the several steps mentioned above. The several steps and actions to be taken are strongly related to each other. Figure (2-2) shows how each step is involved in the sequence of providing routing information around footprints of disasters and what information is sent in each step to each of the separate elements of the pilot.



2-2 Sequence diagram on routing for emergency management showing the sequence and dataflow

To reach the objectives of this study two services are being used: one on the transport of dangerous goods and another one for submerged roads.

The first case that we present is a case on the transport of dangerous goods, this is a scientific model developed for the Preview IP. In this model the footprint of a chemical release is calculated.

The same functionality is then used in another context to demonstrate how one single web services can be used to meet different requirements. For this we present the Gard pilot; a pilot study developed mainly by the Ecole Nationale Point et Chaussées (ENPC)⁴ that demonstrates the practical use of (near) real time flood forecasting. It is the intention to extend this to provide routing for emergency services.

The third functionality extends the work of the combination of the other two by exploring the possibilities of semantics and knowledge base querying in order to establish scenario's for emergency situations in the context of the transport of dangerous goods.

⁴ Floodsite partner in Floodsite task 17, task on management for flood emergencies.

These underlying use cases are written out in more detail in annex I. The following list sums up from which IP's the elements of the use-case come from.

- Orchestra: routing
- Preview: impact area of chemical atmospheric dispersion
- Floodsite: forecasting of submerged roads in case of flash floods
- Semantics: establish incident scenario's for the transport for dangerous goods.
- Footprint: grabbing of external data using WFS

These will be brought together in a single use case that deals with emergency management.

Within this use case special attention is dedicated to the role of the network, the rationale of emphasizing the role of the network comes from approaching emergency management as a problem of connectivity⁵. Resources are located in one location but need to be transported to other locations; people are at risk and need to be evacuated; civil protection services need to know how to reach sites of hazards.

In order to accomplish this transport or exchange, several other types of connections need to be activated or used. Over these connections a variety of items can be exchanged over space and time, ranging from (already mentioned) persons, to data, to rescue services, to supplies etc.

A very specific role is ascribed to network analysis. Network analysis provides an input in the decision making process on the best use of the existing network. This network analysis can be performed on physical networks, like road networks and electricity networks, but also on cyber networks and communication networks. The main goal of these network analyses is to understand connectivity and how a loss in connectivity affects core activities.

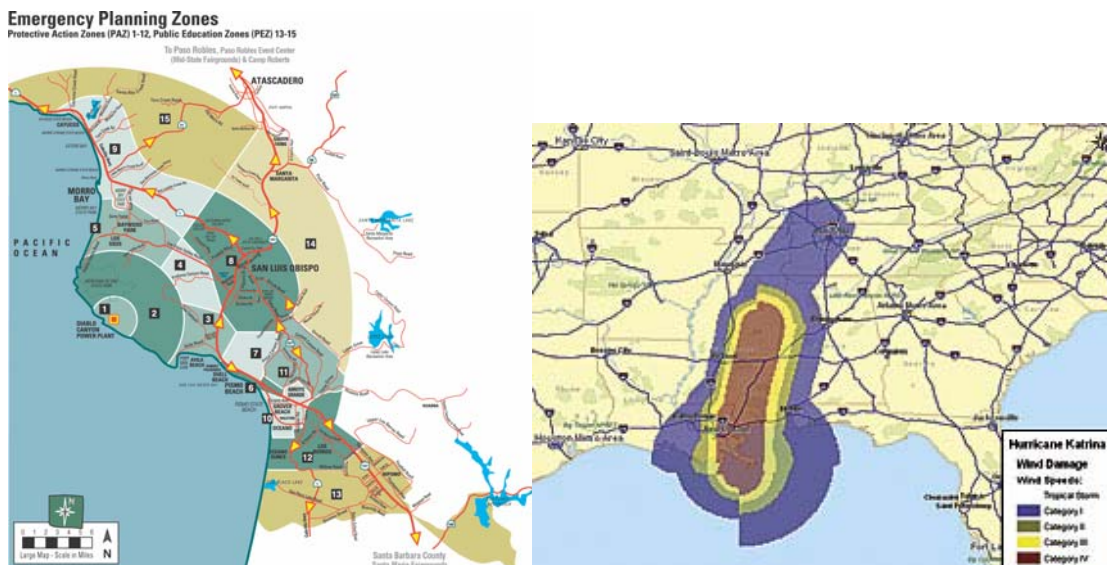
How well connected two entities are to each other determines for a large part the coping capacity of these two entities for an event that upsets parts of a connection. In this way connectivity becomes a very important aspect of vulnerability and therefore of decision making with respect to emergency event management. This provides the rationale for this study to focus on the role and availability of the road network.

⁵ Floodsite technical Note T17-07-05, "User requirements in flood evacuation management"

Footprints and emergency planning zones

By focusing on the availability of a road network some sort of generic format is need for disaster impact areas. This generic format is supposed to inform the user on how a disaster or event is geographically related to the road network. In order to deal with this issue in a more general way the concept of footprints is introduced here.

In the practice of emergency planning, civil protection authorities often work by delineating a zone around the place of occurrence of a disaster⁶. This zone is The Emergency Planning Zone (EPZ), a designated area to be used for anticipatory protective action or in the event of an evacuation⁷. Two examples of an EPZ can be seen in figure (nr)⁸



2-3 Examples of footprints found on the internet

Both figures display an event, either a planned or a foreseen nuclear leak, or the path of a hurricane over a territory. What comes back in this figure is some distribution of intensity resulting in different measures of mitigation in different areas

The EPZ itself is an important part of the communication between the various civil protection authorities involved in the response as it determines where what kind of action is needed.

An EPZ is typically delineated by knowing a disaster's footprint. This footprint is a collection of contour lines that, according to a predefined scheme, describe the hazard by its location, parameters and extent. Then rules and regulations, or practical constraints determine what the practical extent of the disaster is. One could imagine one of the following situations:

1. An incident occurs in a factory that uses a toxic chemical for one of its process. As a consequence an amount of this toxic gets dispersed in the atmosphere. The toxicity is only regarded as life threatening once it reaches a density that is higher than 1 mg per m³. As a result of modelling or measuring (preferably both) contour lines of density are established. And

⁶ http://www.esri.com/mapmuseum/mapbook_gallery/volume18/state8.html

⁷ Definition of UK Atomic Energy Council: <http://www.aec.gov.tw/english/emergency/article.php?n=03>

⁸ Source: <http://www.slocountyoes.com/emergencyplanning/epz.html>

these are drawn on a map. This serves as an input into further decision making. The contour line of 1 mg/m³ is considered as the EPZ of the hazard.

2. Due to severe weather one of the rivers in a mountainous area overreaches its embankment and the surrounding areas get flooded. Depth measurements are done and depth contour lines are quickly established. Due to the nature of the available rescue equipment, an inundation level of 20 cm is regarded as the upper limit for going out with the 4x4. Therefore the inundation level of 20 cm is regarded as the footprint of the disaster. Therefore an EPZ is established which shows the parameter and extent of inundation with a depth of 20 cm.

Or, taken from a real world example on emergency planning around a nuclear facility:

3. The Emergency Planning Zone (EPZ) is a designated area to be used for anticipatory protective action or in the event of an evacuation. The size of the zone is related to the type of nuclear reactor, the size of the population near the power plant, the topography and the meteorological conditions. Based on various internationally recognized standards with regard to radiation levels and other parameters, we have established the EPZ's of our No. 1, No. 2 and No. 3 nuclear power plants at 5 km from the plant. Within the EPZ's it is required that preparedness work be done on a continuing basis.

In all three examples the recurrent element is the distribution over time and space of a phenomenon and the delineation of an EPZ, based on a relevant intensity of the event. The relevant intensity itself is then determined by practical reasons, rules and regulations, within the distribution of a phenomenon over time and space. When this zone is defined, the emergency responder will use it to plan various operations within and outside that zone, and to plan resource allocation.

Within an emergency response, the various involved civil protection authorities that carry out the various operations in an emergency situation (like fire-brigade, police forces and ambulance services) each need to be informed about the EPZ. When this communication, and more specifically web communication, between diverse civil protection authorities takes place a need arises to standardize the format in which a disaster's footprint is published.

This need can be fulfilled by what this study calls a Geographic footprint:

Geographic Disaster footprint: A common format that describes the geographic extent of an impact or the consequences of hazards be it natural or man-made. A footprint is published in a standardized format to facilitate its use in a variety of software environments that is as wide as possible, regardless of operating platforms and allows for data interoperability.

This Geographic footprint is published in a format that contains information about the location of the impact, the shape of the impact, its size, intensity iso-lines, and potentially its evolution over time. It is described by (a) geometric and (b) thematic components. Geometric components are e.g. location, shape, and extent. The thematic component is e.g. type of disaster and intensity of a disaster.

By combining these two types of components, one could describe disaster footprint by its intensity and the distribution of that intensity over space. Knowing this, an emergency responder can estimate a situation, delineate an EPZ and determine where emergency response is needed.

Due to its generic design, this object, the geographic footprint, then facilitates the creation of services that can generically act on such kinds of footprints, including routing tools, tools that generate locations for e.g. roadblocks, and vulnerability mapping.

This study further continues with the idea of reducing disasters to footprint and its relationship to the road network and to demonstrate how this thinking helps in geospatial interoperability for emergency response.

3. Demonstrator design

From the use case the requirements for the demonstration of geospatial interoperability can be derived. This use-case then dictates the following basic functionalities:

1. A way to send or retrieve information about a disaster and to see the footprint of that disaster.
2. a way to calculate a footprint
3. Have this footprint interact with the road network to see which parts of the road network are available or not.
4. Perform routing services between a point of origin and a point of destination.
5. display all of the results on a screen

This in essence demands that the following basic functionalities must be incorporated into a demonstrator:

- An internet based map viewer that visualized geospatial data
- A way to describe a source term and send it to the footprint service
- Store geospatial data
- Functionalities to perform geospatial operations.

These are the basic requirements of a demonstration on interoperability, in order to ensure geospatial interoperability the choice is made to adhere to certain standards.

As discussed before: “Geospatial Interoperability⁹ is the ability for two different software systems to interact with geospatial information. Interoperability between heterogeneous computer systems is essential to providing geospatial data, maps, cartographic and decision support services, and analytical functions. Geospatial interoperability is dependent on voluntary, consensus-based standards... These geospatial standards are essential to advancing data access and collaborations in e-Government, natural hazards, weather and climate, exploration, and global earth observation.”

The main organization to work on these standards for geospatial data is the Open Geospatial Consortium (OGC¹⁰). This organization has specified a number of standards in order to promote and facilitate geospatial interoperability. OGC has named and specified four basic web services, all these web services are HTTP based standards to ensure client and sever independency.

Web services¹¹ “are Web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients“. The design choice for web services comes from a desire to make the functionalities open to use (for potentially everybody) and to be able to re-use them in different applications. The architecture chosen for those is the so-called service-oriented architecture. Service-Oriented Architecture (SOA) is an architectural style that supports service orientation and is adopted in for example the

⁹ Geospatial Interoperability (GI) Return on Investment Study Report , NASA, April 2005

<http://gio.gsfc.nasa.gov/docs/ROI/20Study.pdf>

¹⁰ <http://www.opengeospatial.org/>

¹¹ <http://java.sun.com/webservices/>

INSPIRE implementation¹² as well. The following sections give an overview of these four basic web services.

WMS: Web Map Service. Displays geographic data as images (maps). A WMS (or Web Map Server) allows for use of data from several different servers, and enables for the creation of a network of Map Servers from which clients can build customized maps.

The OGC website¹³ says the following about the WMS specification:

When client and server software implements WMS, any client can access maps from any server. Any client can combine maps (overlay them like clear acetate sheets) from one or more servers. Any client can query information from a map provided by any server. While programmers need to write code to implement the specifications, end users can take advantage of products that include them to publish and access geospatial information. If end-users all implement the same standard, WMS, they will all work together.

In particular WMS defines:

- *How to request and provide a map as a picture or set of features (GetMap)*
- *How to get and provide information about the content of a map such as the value of a feature at a location (GetFeatureInfo)*
- *How to get and provide information about what types of maps a server can deliver (GetCapabilities)*

WFS¹⁴: Web Feature Service, communicates real geographic data (roads, rivers etc.) to and from the user in the form of GML. WFS (Web Feature Service) publish geospatial data to the web. This means that instead of returning an image, the client now obtains fine-grained information about specific geospatial features of the underlying data. As with other OGC specifications, this interface uses XML over HTTP as its delivery mechanism, and, more precisely, GML (Geography Markup Language), which is a subset of XML.

WFS-T: Web Map Service-Transactional. Allows the users to edit geographic data in transaction blocks.

The following WFS operations are available to manage and query geographic features and elements:

- *Create a new feature instance*
- *Delete a feature instance*
- *Update a feature instance*
- *Lock a feature instance*
- *Get or query features based on spatial and non-spatial constraints*

WPS:¹⁵ can be configured to offer any sort of GIS functionality to clients across a network, including access to pre-programmed calculations and/or computation models that operate on spatially referenced data. A WPS may offer calculations as simple as subtracting one set of spatially referenced numbers from another (e.g., determining the difference in influenza cases between two different seasons), or as complicated as a global climate change model.

¹² http://www.ec-gis.org/inspire/reports/ImplementingRules/INSPIRE_Metadata_ImplementingRule_v3_20071026.pdf

¹³ <http://www.opengeospatial.org/standards/wms>

¹⁴ <http://www.opengeospatial.org/standards/wfs>

¹⁵ OPEN GIS WEB PROCESSING SERVICE, OGC DOCUMENT: OGC 05-007r4 available at: <http://www.opengeospatial.org/>

The data required by the WPS can be delivered across a network, or be made available at the server

WPS defines a standardized interface that facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients.

The WPS specification is designed to allow a service provider to expose a web accessible process, such as polygon intersection, in a way that allows clients to input data and execute the process with no specialized knowledge of the underlying physical process interface or API. The WPS interface standardizes the way processes and their inputs/outputs are described, how a client can request the execution of a process, and how the output from a process is handled.

Working according to these specifications allows for working in a distributed environment. In such an environment data and processes would be installed on *many* machines as opposed to on one *local* machine. And published data and maps will be re-usable in other applications as well.

Proposed approach and software components

After having defined the basic functionalities of the demonstrator and the after having laid out the main principles of design, the choice now has to be made for software that support both the basic functionalities and the design principles. This section described what software packages were used for the VATDIS demonstrator.

With the increasing importance of the OGC web services for managing, viewing and processing geospatial data through the web, a number of software packages have been developed that will support these web services.

In theory they should all work together. However in practice, specific user requirements dictate specific choices. This section described only the choices made for the VATDIS demonstrator. It is not a design written in stone, as in fact many software packages will do the same but in slightly different manner.

In the case of the VATDIS demonstrator, the development of the web service as a completely OGC compliant web service restricts the wide choice of available packages by picking out the WPS package offered by 52 North.

Then the choice needs to be made for a package to serve all the web services. These web services are http based, this implies that a supporting tool is need to be able to deal with requests and to send back images, features and results of geospatial operations.

Our previous selection for 52 North implies that the server has to be a tomcat server, as 52 North was mainly designed for tomcat. Tomcat is a servlet container in which JAVA based servlets (services) are being hosted and can be accessed through the http protocol over the World Wide Web.

From this comes automatically the choice for the other packages: Mapbuilder will be used to translate the web services into images that can be seen on screen in a map viewer and Geoserver will be used to translate geospatial data into OGC compliant web services. Mapbuilder and Geoserver are chosen due to their compatibility with tomcat.

Then a choice needs to be made for a package that stores geospatial data and that is also compatible with Geoserver. In this study PostGIS is used for that purposes. On tope of being able to store geospatial data, PostGIS also offers the functionality of being able to manipulate through parameterized functions that the programmer can design.

This last functionality was a main drive to choose for PostGIS as it allows designing the chemical dispersion model and the routing model¹⁶.

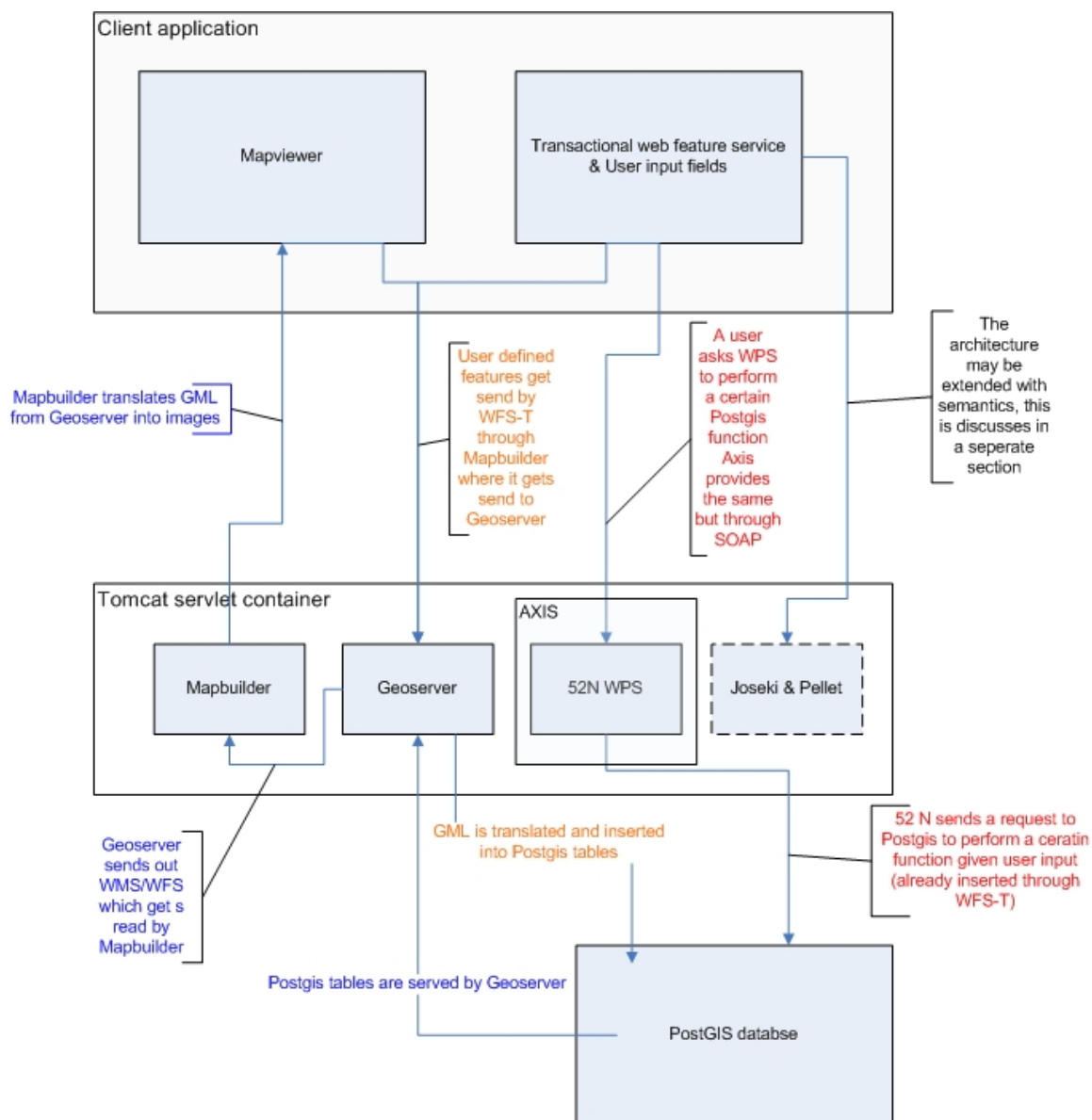
Whereas the integration of WFS and WMS into applications other then the VATDIS application would be facilitated by using Geoserver, the sending of requests to and retrieving results from a Web Processing Service requires some extra attention. A special http protocol has been designed in order to deal with this kind of requests, this is to SOAP protocol The web processing service as such is still hosted under the tomcat however a translation of the incoming SOAP request needs to be made, this is done by a tomcat compliant package that is called AXIS.

¹⁶ <http://www.postlbs.org/>

The packages that were used for the installation of the service are the following ones:

- Postgres with PostGIS extension
- Geoserver
- Mapbuilder
- 52n web processing
- Tomcat web server
- Axis

Another advantage that these packages offer is that they are all open source packages. That means that under a certain license, they are free for download and use.



3-1 Overview of software packages and how they work together. NB: AXIS is placed here as a wrapper around WPS. To demonstrate that it really is an alternative entrance for accessing the same PostGIS functions.

Figure 3-1 describes how these packages work together and what specific role they play in the service. In essence the following functionalities are provided:

- Data storage by Postgres
- Data access through wms, wfs and wfs-t through Geoserver
- Data visualization through Mapbuilder and web browsers.
- Data processing through WPS and 52 north web processing service
- Semantic data access and query through sparql endpoint using Joseki

Together these components make sure data is served on the net and that it can be accessed through a protocol that supports data interoperability.

These packages can be obtained for free and come with tutorials and manuals on how to install and use them. The URL's to the websites where these packages were obtained can be found under the references section of this document.

The following section gives an overview and some characteristic of these main components:

PostGIS

PostGIS is a spatial database add-on for the Postgres relational database server. It includes support for all of the functions and objects defined in the Open IS "Simple Features for SQL" specification. Using the many spatial functions in PostGIS, it is possible to do advanced spatial processing and querying entirely at the SQL command-line.

Geoserver

With Geoserver you can publish and edit data using open standards. Your information is made available in a large variety of formats as maps/images or actual geospatial data. Geoserver's transactional capabilities offer robust support for shared editing. Geoserver's focus is ease of use and support for standards, in order to serve as 'glue' for the geospatial web, connecting from legacy databases to many diverse clients.

Geoserver supports WFS-T and WMS open protocols from the OGC to produce JPEG, PNG, SVG, KML/KMZ, GML, PDF, Shape files and more.

Mapbuilder

Mapbuilder is an open source project allowing you to easily add dynamic maps and information from many other sources to your web site. Using Mapbuilder, Web designers can concentrate on the presentation and usability using HTML with simple links to the Mapbuilder JavaScript code using HTML element IDs, with additional presentation information provided using CSS and XSL. The content and style of the maps themselves will often be defined using Web Map Context documents although the modular design framework allows Mapbuilder to work with many other document types.

Mapbuilder implements a framework for dynamic web page content from XML documents using AJAX (which is shorthand for Asynchronous JavaScript + XML). Mapbuilder consists of a JavaScript library that implements the Model-View-Controller (MVC) design pattern.

52 North web processing service

A WPS can be configured to offer any sort of GIS functionality to clients across a network, including access to pre-programmed calculations and/or computation models that operate on spatially referenced data. The data required by the WPS can be delivered across a network, or available at the server. (Snippet from the OGC website). 52 North is an implementation of such a WPS that can be installed under tomcat. The editing of the web service is done by using Eclipse.

Axis

Axis is used to expose processes as web processes. Whereas in a normal application the Web Processing Service is still tied to the application, axis allows exposing the service and making it re-usable from outside the application for which a web processing services originally was designed. In order to do so, AXIS is designed to support the SOAP protocol.

Tomcat

Tomcat is the container in which all of the above described applications, except PostGIS can be hosted. The above described packages can be downloaded in a .war file format. This file can then be unzipped and loaded into the tomcat container by making use of the tomcat manager that comes with the installation. Tomcat works both on UNIX and Windows.

With these packages a demonstrator has been built that makes it possible to implement OGC based standards in a practical way. Exactly how this can be done is explained in more detail in the next chapter.

4. Implementation

For completing the demonstrator a number of challenges have been identified:

1. Creating a routing web service
2. Creating a web service for a chemical dispersion model
3. Offer a publication service for spatial data / import external features
4. Export functionalities to Google earth
5. Comply with OGC standards for interoperability, i.e. wrap services in the SOAP protocol
6. Demonstrate the possibility of extension with semantics

This section aims at describing in more detail the solutions proposed to the functionalities the tool is supposed to offer (besides the normal functionalities that any web-mapping application offers) technical details will be discussed in the annexes. As a basis we recommend reading "An introduction to GIS systems" from Vincenzo di Lorenzo¹⁷ which provides a basic manual on how to setup Mapbuilder in combination with Geoserver and PostGIS.

Routing Web service

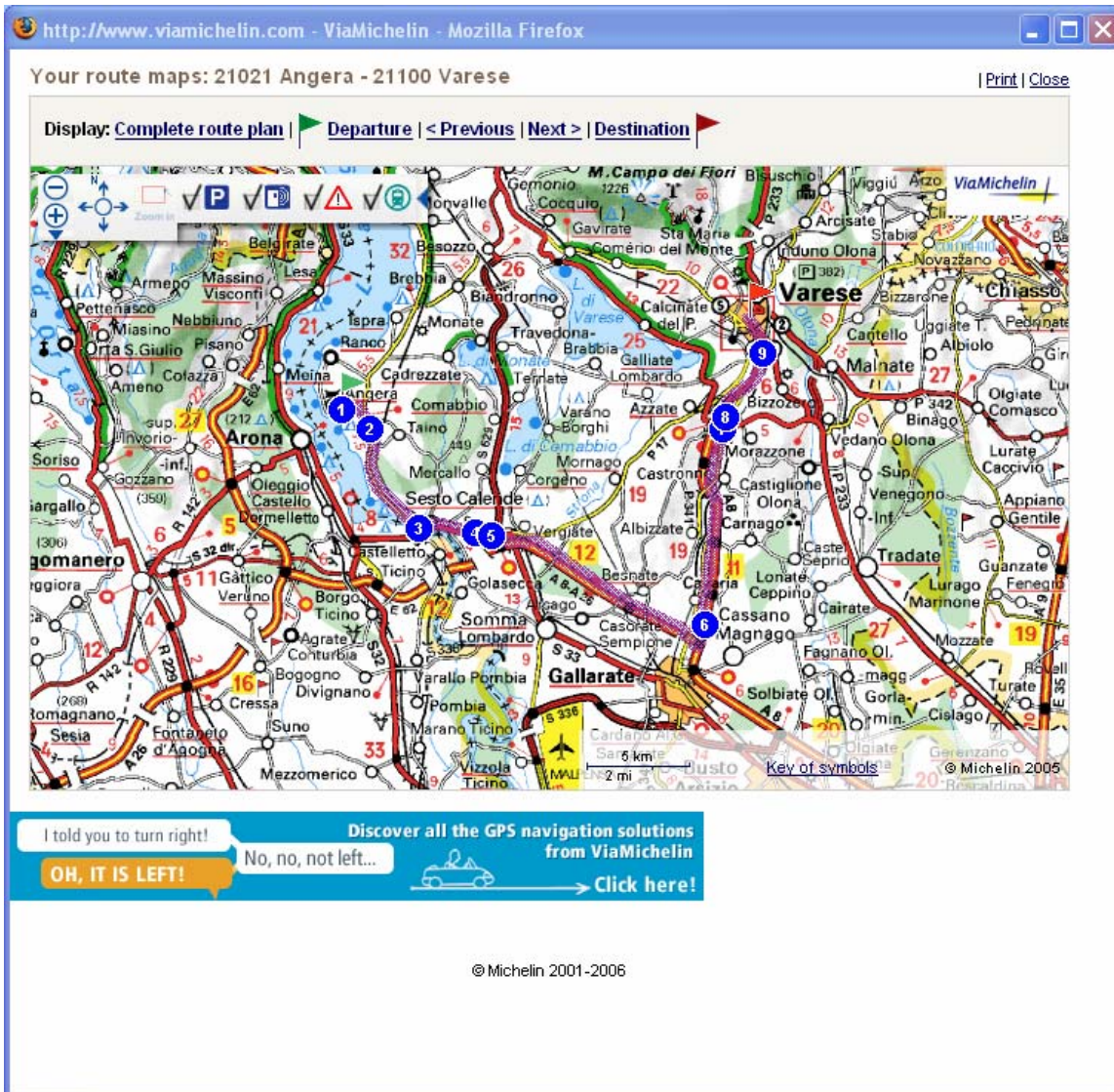
Routing is one of the basic location based service. In fact many web based application are available that handle routing request from users. The most common ones are:

- Viamichelin (www.viamichelin.com)
- Google maps (www.maps.google.com)

Except for the latest version¹⁸ neither of these services handles routing by avoiding certain pre-determined road blocks. I.e. roads cut by incidents. In order to do so a routing implementation was found that allows road status (i.e. road cost) to be set according to the spatial relationship with the footprint of a disaster. Such a solution was found using add-in for the PostGIS database that provides a routing solution.

¹⁷ Downloadable at: <http://www.opensourcesupport.it>

¹⁸ Writing on 09-11-2007



4-1 Example of an on-line routing application, in this case the routing from viamichelin

For routing the PGdijkstra algorithm was chosen. This is an 'add-on' of the PostGIS functions and can be downloaded at:

<http://cartoweb.org/downloads..html#pgdijkstra>

In essence a new set of functions is created within the PostGIS database. These functions are geared towards routing: they serve the creation of the routing graph and implement the routing algorithm.

These functions can be called with SQL statements. These can later be used in database functions which in essence are pre compiled SQL statements that can be called (both internal and external). The installation of the algorithm follows a few steps¹⁹ which are explained into more detail in the annex. The routing algorithm works by inputting a starting point and a departure point in a coordinate format.

¹⁹ the precise stapes can be found at: <http://chris.narx.net/2005/12/14/build-your-own-routing-solution/>

One of the demands of working with routing in a fast way is to have the user interact with the routing algorithm by using visual input rather than typing in coordinates. That means that the user is expected to click on a start and end point in the map viewer and have the client application send the coordinates of this point to the routing algorithm. This is where the WFS-T functionality of Mapbuilder is useful as it provides exactly this functionality.

The visualization of the routing results is done by Mapbuilder. What happens in essence is that the routing algorithm populates a table in the PostGIS database with the routing results per routing request. This table is preconfigured to be served by Geoserver and visualized by Mapbuilder. Therefore whatever is populated in that table will be visualized on screen.

As such this completes the functionality of routing. The routing algorithm determines the shortest path from one point to another point based on a minimizing the total cost of traversing a graph given a cost for each section of the graph. Cost can be anything ranging from distance, travel time, road class, etc. In the VATDIS demonstrator cost is based on length of the traversed sections.

In order to be work with re-routing it is exactly this definition for cost that plays a role into setting road status. The geometry of each section of the road network is stored in the system, when this geometry intersects with geometry of a footprint, the cost of that road section can be manipulated in such a way that it would not be considered in the final routing result.

In order to offer this routing solution through the web it needs to be placed inside a web processing service. This provides a way to send coordinates (in GML) through the WPS to the PostGIS routing service and have the resulting route send back through the Geoserver in the map viewer.

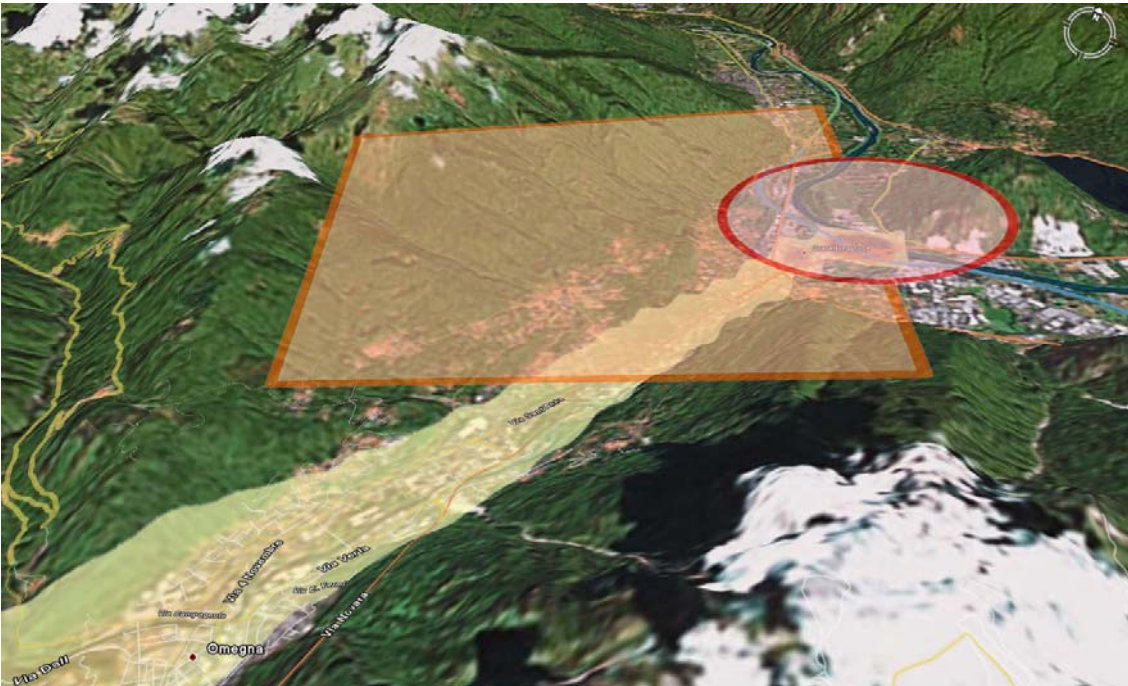
A more detailed description of how the combination 52North / PostGIS for web processing works is given in annex (6) the next section explains how this combination is used to create a chemical dispersions model.

Creating a web service for a chemical dispersion model

Given the huge difficulties and time consuming calculations to provide real-time atmospheric dispersions of chemicals a more simple approach has been developed by the American fire brigade /civil protection. This simpler model offers the advantage of a quick estimate of impact zone and casualties. Within the PREVIEW IP this model has been chosen to be implemented as a web service.

Figure (4-2) shows how the simplification works. The yellow cloud is the outcome of a very precise model taking into account terrain elevation, wind direction, etc. However the calculation time of this model far exceeds the necessary response time in emergency situation. This makes it not practical to use.

The chemicals are ordered by UN code, a unique identifier for the type of chemical. After identification has taken place on the scene a user should be able to insert this chemical in the tool, along with other parameters, and be able to retrieve the relevant impact zone. Consequently the user will identify affected roads and will be able to use these as an input into routing requests.



4-2 Demonstration of the difference between real model (yellow cloud) and the simplified model (red circle and orange square)

Within PostGIS a set of functions have been designed that create the geometry of this footprint. In order to do this, first a table was created in PostGIS containing all the chemicals sorted by UN number. Figure (4-3) shows how such this table looks like in excel

	A	B	C	D	E	F	G	H	I
1	UN	name	isolate,sm	protect da	protect nig	isolate,larg	protect da	protect night	
2	3304	Compressed gas, poisonous, corrosive, n.o.s.	610	5955	12714	914	27198	29290	
3	3304	Compressed gas, poisonous, corrosive, n.o.s. (Inhalation Haz	610	5955	12714	914	27198	29290	
4	3305	Compressed gas, poisonous, flammable, corrosive, n.o.s.	610	5955	12714	914	27198	29290	
5	3305	Compressed gas, poisonous, flammable, corrosive, n.o.s. (Inh	610	5955	12714	914	27198	29290	
6	1955	Compressed gas, poisonous, n.o.s.	610	5955	12714	914	27198	29290	
7	1955	Compressed gas, poisonous, n.o.s. (Inhalation Hazard Zone A	610	5955	12714	914	27198	29290	
8	3306	Compressed gas, poisonous, oxidizing, corrosive, n.o.s.	610	5955	12714	914	27198	29290	
9	3306	Compressed gas, poisonous, oxidizing, corrosive, n.o.s. (Inha	610	5955	12714	914	27198	29290	
10	3303	Compressed gas, poisonous, oxidizing, n.o.s.	610	5955	12714	914	27198	29290	
11	3303	Compressed gas, poisonous, oxidizing, n.o.s. (Inhalation Haz	610	5955	12714	914	27198	29290	
12	3304	Compressed gas, toxic, corrosive, n.o.s.	610	5955	12714	914	27198	29290	
13	3304	Compressed gas, toxic, corrosive, n.o.s. (Inhalation Hazard Z	610	5955	12714	914	27198	29290	
14	3305	Compressed gas, toxic, flammable, corrosive, n.o.s.	610	5955	12714	914	27198	29290	
15	3305	Compressed gas, toxic, flammable, corrosive, n.o.s. (Inhalatio	610	5955	12714	914	27198	29290	
16	1955	Compressed gas, toxic, n.o.s.	610	5955	12714	914	27198	29290	
17	1955	Compressed gas, toxic, n.o.s. (Inhalation Hazard Zone A)	610	5955	12714	914	27198	29290	

4-3 Screenshot from the UN chemicals list (screenshot taken from Excel)

The table consists of the chemicals, identified by its UN number and some attributes concerning relevant zoning distances for respectively small and large spills.

These distances serve as an input for calculating the geometry of the impact zones. To start the process of calculating this geometry the user needs to point to a location on a map and input the type of chemical. Here again we use the transactional wfs that comes with Mapbuilder. The user points to a location on the map, and the coordinates get send to the PostGIS function, this function then builds the rest of the geometry around that point.

Annex (4) explains in more detail the design of the functions in PostGIS. After that a 52n wrapper is placed around it making the design compliant with OGC WPS standards, annex (6) explains in more details the design of the combination 52 North with PostGIS.

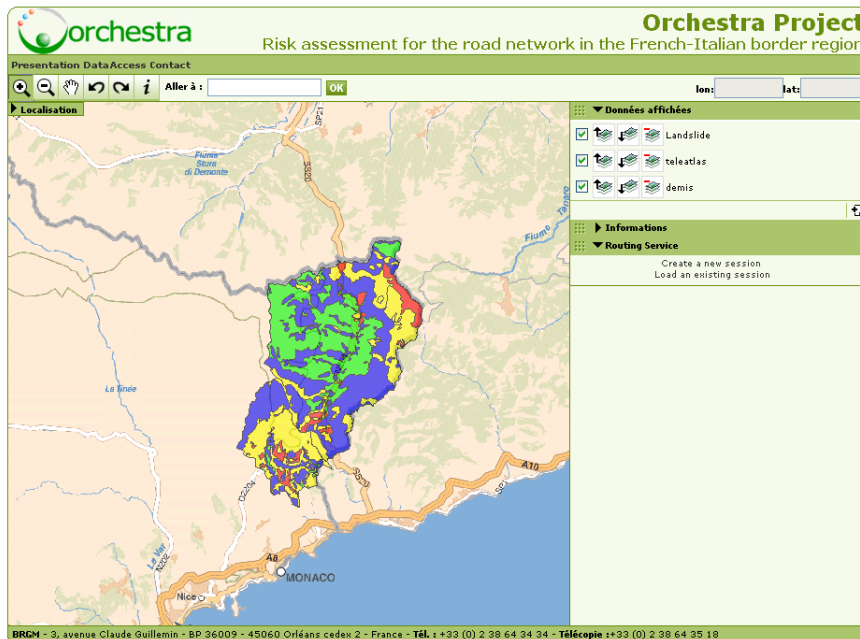
Grabbing of spatial data

In many cases organizations will have published their data in the web in a way that is compliant with the OGC standards. This allows for easy connection to that data and to have that data published also in other applications.

As said before by publishing data through WMS, clients that adhere to this service can import that data into their own map viewer in their application. In the configuration file of map builder one can point to the own, local, publication service or those of others. The publication of geospatial data in our case is done using a package like Geoserver.

Besides merely the images of the map, also the underlying features can be published on the web, this is done using WFS. With this the features are published in gml format. Potentially these features could be grabbed and inserted into a PostGIS database, or stored in some other way.

A practical example is one where a user has a server available and is able to make the data available through WMS/WFS. With this he can make his own mapping application. However we can grab this data with a WFS request (wrapped or not in some sort of java tool) and upload it into our PostGIS database from where we make it available on Geoserver.



4-4 Example of a WFS layer in a map application, these features can be grabbed using the following WFS request:

<http://orchestra.brgm.fr/cgibin/orchestra?service=wfs&version=1.0.0&request=GetFeature&typename=landslide>

The screenshot shows a workflow for data integration. On the left, a web browser displays XML data in GML format, including coordinates and feature names like 'landslide'. In the center, a terminal window shows the execution of the `ogr2ogr` command to import this data into a PostgreSQL database. On the right, a file explorer shows a list of files, with 'landslide' highlighted in red. A text box at the bottom explains the steps: a WFS request is made, the data is saved in GML, it is then imported into the database using `ogr2ogr`, and finally, it becomes available for analysis.

The wfs request looks like this, see also figure 4-4:
 A description of geo-data in gml for the landslide layer on the brgm server
 The file then gets uploaded into our data base with ogr2ogr
 And after that will be available for road network analysis

4-5 Overview of steps taken to insert GML data obtained through WFS into the PostGIS database.

Figures (4-4 and 4-5) demonstrates how this is done: first a wfs feature request is made to WFS compliant server.

1. A WFS request is made to a WFS server.
2. The features of the request are stored in GML format
3. Using ogr2ogr (A publicly available library) the features are stored in a PostGIS database
4. After this the features become available for footprint based services.

Possibility of calling all of the above within Google Earth

To work with geographic data three basic paths are open for the user, there is a way to use only proprietary software, to use open source software or to implement and use data using a combination of one of the previous with Google Earth.

Google Earth however has only limited capabilities for spatial analysis and geo processing. However progress is made to start providing geo processing within the Google Earth platform see for example: <http://www.gulfimpact.com/> a site dedicated to the analysis of the impact of the Hurricane Katrina on the oil industry in the Mexican Gulf.

With this site a demonstration is made of using Google Earth for visualization and using open source geo spatial analysis tools (Postgres, Geoserver) to allow for user input and processing.

Using Google Earth and producing contents on it has some major benefits for risk management:

1. information can be placed and visualized in its context,
2. world wide platform that is easy to use

Ad 1. Geographic information can be put in a context that is easy to recognize for users. They will be able to relate the content to images of the objects they are familiar with instead of having to translate first abstract objects on a map.

Ad 2. The Google earth platform is freely available and runs on most of the modern operating systems and hardware. The use of the platform is rather intuitive and resembles the use of e.g. windows explorer or Mac osx finder. Therefore providing contents on Google Earth ensures risk communication to a larger audience.

Functionalities can be imported into Google Earth (GE) using a network link, this is a bit of code written in GE's own language (KML) with which through HTTP GET and POST the location at which the user is looking at can be send to the server and parsed through to the WPS.

By basing Google Earth functionalities on top of the existing structure of Geoserver and Postgres it becomes a logical extension of the architecture that is already in place.

Comply with INSPIRE/OGC standards for interoperability, i.e. wrap services in SOAP.

Soap²⁰ (Simple Object Access Protocol) is a protocol strongly linked with web services. Web services as such can not talk to each other; they need a common language in order to exchange data.

SOAP is the language that allows web services to talk to each other; it is an XML based language that provides a basic messaging framework. Usually a message pattern is used in which one network node sends a request to another node after which it receives a message back.

²⁰ <http://www.w3.org/TR/soap/>

For geospatial interoperability this means for example that within a client configuration a request for a geo process is send, wrapped in a soap message, to another service which reads the soap message understands what kind of process needs to be carried out and returns the result of the process in a message to the client.

Demonstrate the possibility of extension with semantics

In order to extend the pilot with semantic one of the first steps that needs to be carried out is to describe the use case for semantics. Again emphasize is placed here on integration, i.e. integration of knowledge and data sources. Data can be given a meaning by creating an explicit specification of a conceptualization. I.e. an ontology.

This specification usually takes place within a domain; the domain valid for this study is the domain of risk management and vulnerability analysis.

The use case for the extension of the semantics deals with a description of data that is relevant in scenario's of attacks and the on-the-fly development of these scenarios by making use of reasoning.

In order to achieve these goals a combination of applications has been installed that, can read the rdf-data, reason on it and present the output to the user. These are the following applications with their respective roles:

- Joseki: rdf storages and sparql endpoint
- Pellet: reasoning machine

Joseki is an HTTP engine that supports the SPARQL Protocol and the SPARQL RDF Query language. Joseki can be configured to run under tomcat from where it will be possible to combine RDF queries with the other web mapping functionalities. Joseki can be extended to work with web based applications for reasoning like e.g. pellet. By installing Joseki under e.g. tomcat a sparql end point is made, one of the first steps in creating semantic web applications.

Through such and endpoint ontologies (description of knowledge) can be queried. Eventually such an endpoint can be extended with a reasoning application, like Pellet.

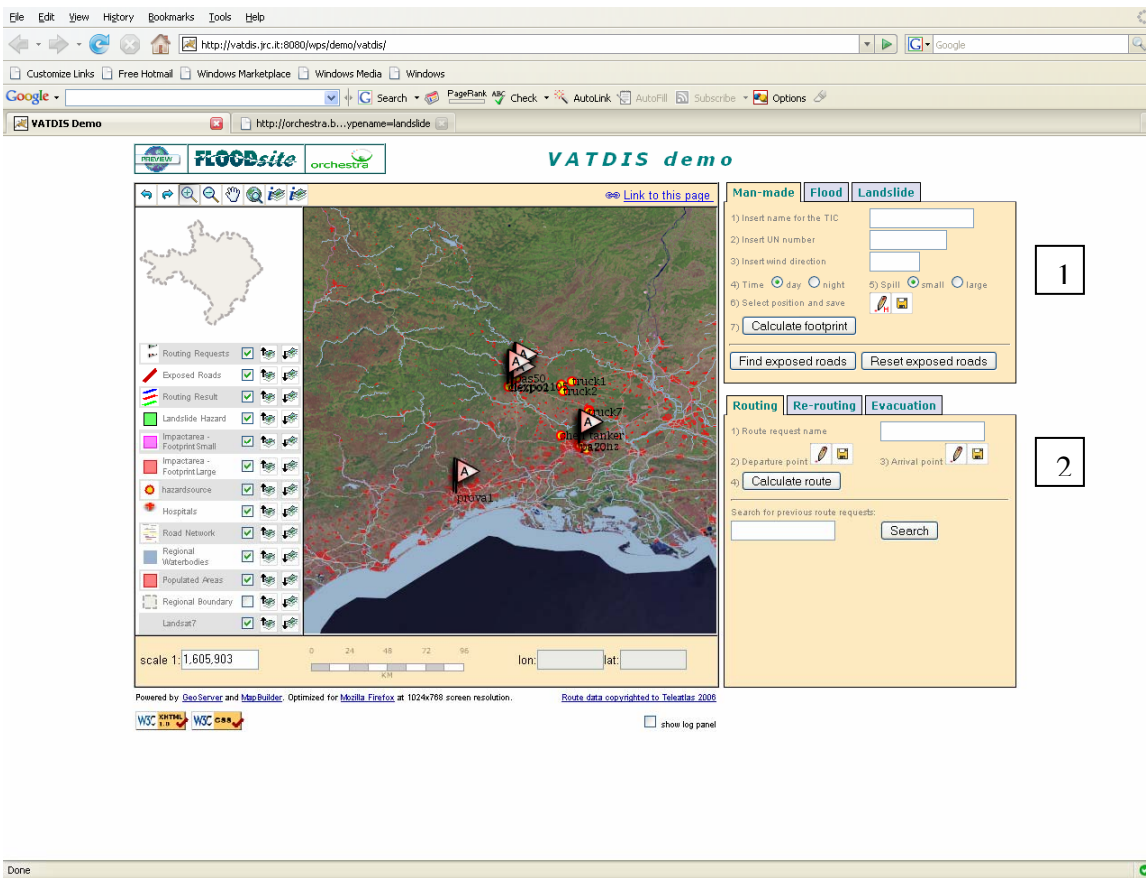
5. Result

Working demo

The resulting demonstrator will be presented in this section. The demonstrator is installed under the following URL: <http://vatdis.jrc.it:8080/vatdis>

Outline of the tool

The tool has designed to serve as a common platform for Preview, Orchestra and Floodsite. The common theme in the contribution to all these three IP's is routing. This is reflected in the tool by separating the footprint part form the (main) routing part.



5-1 Screenshot of the application as it hosted on <http://vatdis.jrc.it:8080/vatdis>

The first part (1) is the part that concerns the creation or grabbing from footprints. The footprints for the main made part are created locally whereas the footprints from the flood and landslide part are either grabbed on-line or uploaded into the database from an external actor.

Creating footprints of hazards

The main feature to consider is the upper right part of the window (section indicated with 1, in figure 5-1). This is where the source term gets defined by the user.

5-2 Screenshot of the part of demonstrator in which the user inserts the source term

The user goes through a number of steps in order to complete his request.

1. insert name for TIV
2. Insert UN number, right now the system works with only ten UN Numbers
3. Insert wind direction, in degrees with the north being the zero degree line

After that the user needs to input the location of the truck by using the pencil.

The last step is to send of the request by clicking on calc buffer. It is here that the previously described functions are called and the requested impact zone is calculated.

In figure number (5-3), this example shows the input from a truck at a certain location named “clexpo2” with a chemical inside with number UN 1082 and wind-direction 45 degrees.

5-3 Screenshot of source term filled in, in the user interface.



5-4 Sequence of screenshots that demonstrate the development of a footprint on the screen





Showing from left to right

1. the location of the hazard source,
2. the first buffer around it (Isolate small)
3. And the square around it angled at wind direction 45 degrees (isolate large)

The next steps are the combine this footprint with the routing algorithm. This is done in section 2 of figure 5-1. Figure 5-5 explains how such a sequence works.

Routing Re-routing Evacuation

1) Route request name

2) Departure point   3) Arrival point  





4)

Search for previous route requests:

A routing request is defined by: a request name, a starting point (little pencil, after that click safe) and an arrival point (little pencil after that click safe)

Routing Re-routing Evacuation


1) Route request name

2) Departure point   3) Arrival point  

4)

Search for previous route requests:

Search results: 1 records. (Search took 0.078 sec)

name: <u>clexpo30106</u> 	<u>Calc. route</u>
--	--------------------

The request is completed by clicking calculate route. After that the request is sent to the database where the relevant functions are called. The request name appears also in the list (after clicking search) and here the user has three options:

1. click on the name and zoom to the bounding box of the request
2. click on the red cross and delete the request
3. click on calc route and (re)recalculate

Routing Re-routing Evacuation

Routing results intersecting Exposed roads:

name: <u>clexpo30106</u>	<u>reroute</u>
--------------------------	----------------

The resulting routes that do intersect with a footprint are displayed under "re-routing". From there the user has a choice to find an alternative route for a certain request.

5-5 sequence of screenshots that demonstrate the routing interface



Routing departure point and arrival point are inserted and displayed with flags



The shortest route is displayed with dangerous sections displayed in red

Dangerous means: route section intersected by footprint.



After the user has chosen to take an alternative route it is calculated and displayed on the map

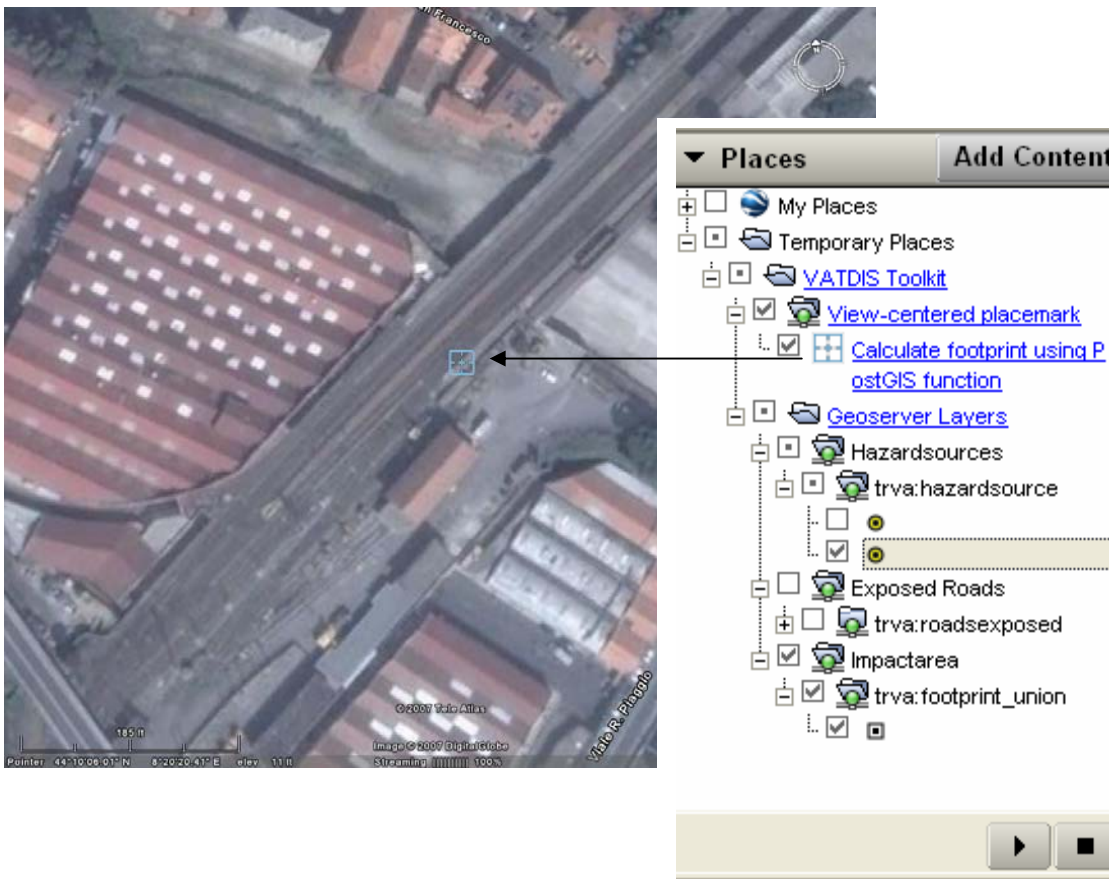
5-6 sequence of screenshots that demonstrate the routing results on the map

More location based services, aimed at supporting disaster management and response, and are implemented within the same platform, using similar functionalities. The following list of functionalities has been implemented as well:

- Find exits from the emergency zone
- Find closest civil protection service (hospital, police station, fire brigade) with contact information.

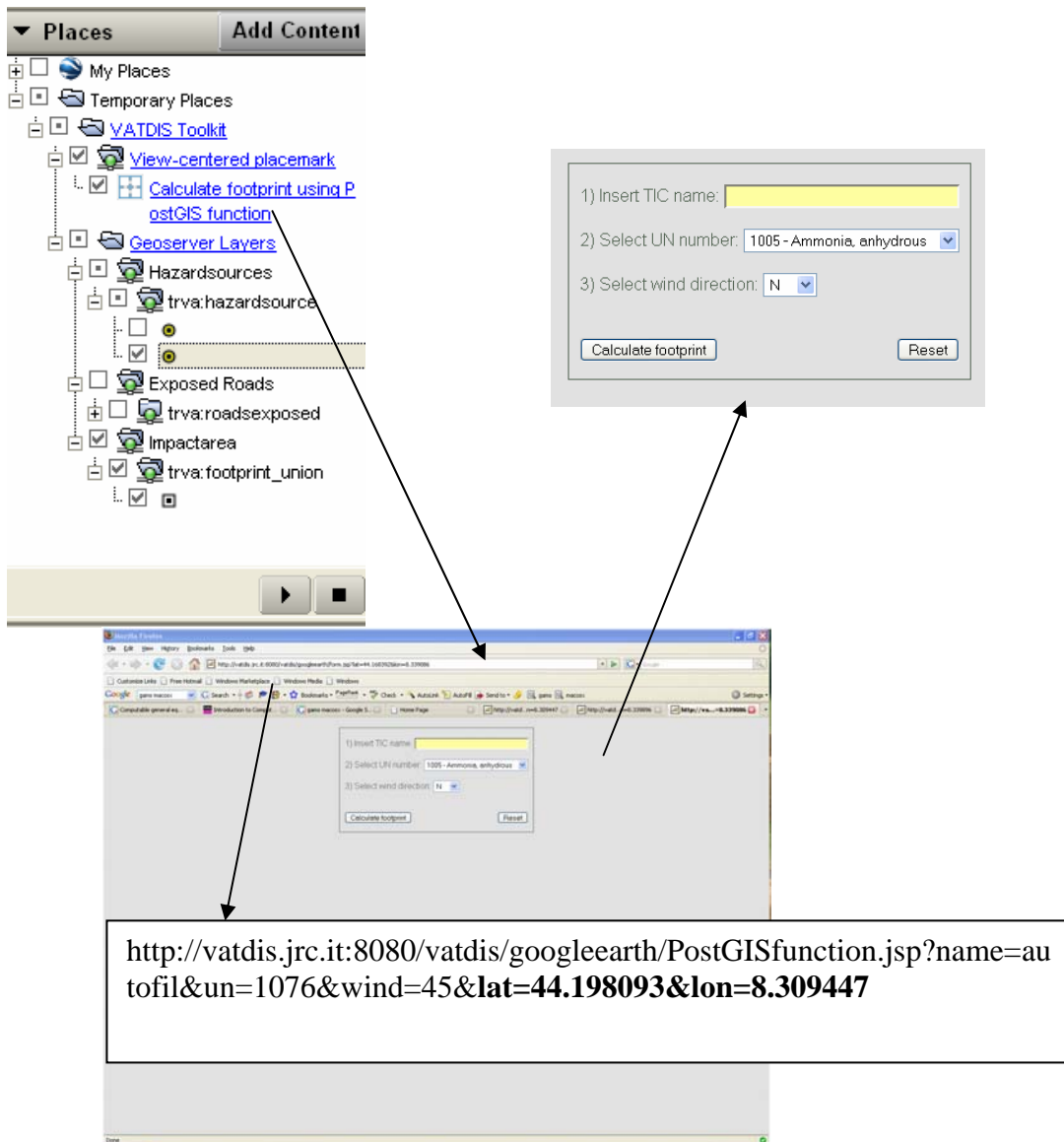
Google Earth

The functionalities within Google Earth are all implemented using a network link. Such a network link provides a port to the implemented web processing service.



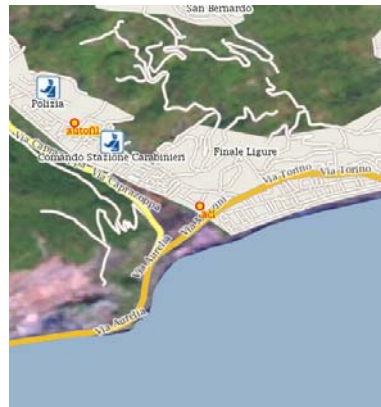
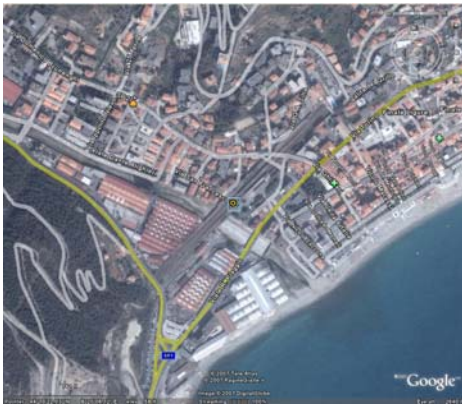
5-8 overview of Google Earth toolbox

This network link is written in a KML format that is placed on the server. By using a link on the client application this KML file is downloaded to the user's installation of Google Earth. The KML file becomes one of the active layers in the user's installation of Google Earth (see figure 5-8) and is dynamically linked to the map section the user is looking at



5-9 Overview of screenshots of a use sequence of the Google Earth link in the VATDIS demonstrator

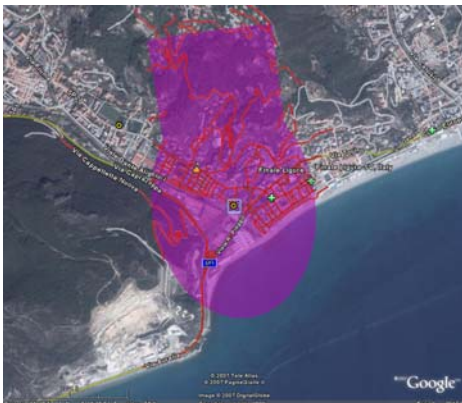
Once this layer is activated, the user can drag and zoom to his location of interest. The coordinates of this location are sent through an HTTP request (see figure 5-9) to a java server page that parses these coordinates to the same PostGIS function that is used in the WPS. The sequence of screenshots in figure (5-9) demonstrates this synergy in Google Earth and the VATDIS demonstrator.



Step 1



Step 2



Step 3

5-10 Sequence of creation of footprint in Google Earth and VATDIS demonstrator

The figure displayed on p.27 21 shows the convergence between working on the vatdis client and working in Google Earth. The same functionalities, exactly the same function within PostGIS, except for the routing, are made available in different applications,

²¹ Differences in shape arise due to difference in projection used by Google Earth and the vatdis client



5-11 Advantage of Google Earth: visualization in wider context

The advantage of working with Google earth becomes more clear in figure (5-10) this figure shows how the footprint can be seen in 3d, showing how the chemical leak would seal up the village. In such a way, it allows for a better visualization of vulnerability. Another advantage in context in which a lot of information is available to a very large audience, this deals with certain problems of distribution, access, and digital rights management. Most of the technical details of the application are given in annex 5

SOAP

In order to be able to extend the previously described architecture with the possibility to use soap, a soap engine needs to be installed that will read and write soap messages. The main open source option is AXIS22, the java based soap engine of apache. AXIS can be installed under tomcat.

For the implementation of the dispersion model as a web service, the idea is that users will be able to send the source description of the chemical accident and retrieve a polygon of the impacted area. As described before in the report, the source description consists of point of accident (longitude and latitude) wind direction at the source, and type of chemical (UN number). The SOAP engine is therefore configured in such a way that users only need to send these four fields to a web service in order to have the footprint calculated.

Figure 5-12 displays the starting screen of the soap based processing service. It explains which services are installed. The most important screen however is the one displayed in figure 5-13 as it explains the grammar to be followed in the soap message that needs to be send to the processing service and it explains the grammar of the message that comes back from the processing service.

Annex 6 provides the details of these messages.

The screenshot shows a Mozilla Firefox browser window with the following content:

- Browser title: List Services - Mozilla Firefox
- Address bar: `http://vatdis.jrc.it:8080/axis2/services/listServices`
- Page content:
 - The Apache Software Foundation logo and URL: `http://www.apache.org/`
 - AXIS2 logo with "Back Home" and "Refresh" links.
 - Section: **Available services**
 - Link: [ProcessingService](#)
 - Service EPR: `http://vatdis.jrc.it:8080/axis2/services/ProcessingService`
 - Service Description: **ProcessingService**
 - Service Status: *Active*
 - Available Operations:
 - `getProcessDescription`
 - `execute`
 - `getCapabilities`
 - Link: [Version](#)
 - Service EPR: `http://vatdis.jrc.it:8080/axis2/services/Version`
 - Service Description: **Version**
 - Service Status: *Active*
 - Available Operations:
 - `getVersion`
- Bottom status bar: `http://vatdis.jrc.it:8080/axis2/services/ProcessingService?wsdl`

Annotations in the image:

- A box containing the URL: `http://vatdis.jrc.it:8080/axis2/services/listServices`. An arrow points from the browser's address bar to this box.
- A box containing the text: "list of web services:
 - ProcessingService
 - Etc.". An arrow points from the [ProcessingService](#) link in the browser to this box.

The screenshot shows the SOAP Message Builder interface in Mozilla Firefox. The browser address bar displays `http://vatdis.jrc.it:8080/test_soap/#`. The interface includes a 'Server Address' field with `http://vatdis.jrc.it:8080/axis2/services/ProcessingService`, a 'SOAP Action' field with 'Execute', and a 'SOAP Message' field containing XML code. Below the message field are 'Request examples' and a 'Submit' button. The 'SOAP Response' field shows the received XML response.

`http://vatdis.jrc.it:8080/test_soap/#`

Soap Message: latitude and longitude will be send to processing service

Soap Response: for the given latitude and longitude, wind direction and a UN number a response is created. This response is a string of coordinates in GML

5-13 sending and receiving messages to the SOAP interface.

Semantics

So far the application has been designed for a user group that has to perform very specific functions with respect to emergency management. This is done by pre-cooked access to geospatial data. This data however has not been given a functional *meaning*, the meaning and significance of the data has to come from ad-hoc user interpretation.

SPARQLer - An RDF Query Server - Mozilla Firefox

http://vatdis.jrc.it:8080/joseki/

SPARQLer

Query forms

- [Form for SPARQL queries on a small books database](#)
- [General purpose SPARQL processor](#)

SPARQL query validator

- [SPARQL query validator](#)

SPARQL Services

The forms above use the following services, which may be accessed directly using the SPARQL protocol:

- [spazql](#) : a general purpose SPARQL processor that processes graphs from the web.
- [pizza](#) : a SPARQL processor that answers queries on small books database only.

SPARQL is defined by 3 documents:

- [SPARQL query language](#)
- [SPARQL protocol](#)
- [SPARQL XML results format](#)

Complete download from the [Joseki download page](#) which includes this set of services.

The Joseki download consists of the following RDF systems:

- [Joseki](#) : A SPARQL Service Processor
- [ARQ](#) : A SPARQL query engine for Jena
- [Jena](#) : An RDF Framework for Java

Done

<http://vatdis.jrc.it:8080/joseki>

Link to sparql query processor.

5-14 overview of semantic querying interface

The geospatial data can be given a meaning by combining this with knowledge from a certain domain. This knowledge typically is represented in a knowledge description, an ontology. The ontology used in the demonstrator can be found at <http://vatdis.jrc.it:8080/joseki/vulnerability.owl>

SPARQLer - General purpose processor

General SPARQL query : input query, set any options and press "Get Results"

```
PREFIX : <http://example.org/book/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX assert: <http://www.owl-ontologies.com/assert.owl#>
PREFIX vulnerability: <http://localhost:8080/joseki/vulnerability.owl#>

SELECT ?explosive_substances ?IsolateSmallv
FROM <http://localhost:8080/joseki/vulnerability.owl>
WHERE
{ ?explosive_substances vulnerability:hasIsolateSmall ?IsolateSmall
.}
```

Target graph URI (or use FROM in the query)

XSLT style sheet (leave blank for none): or JSON output:

<http://vatdis.jrc.it:8080/joseki/sparql.html>

Reference to stylesheet that translates the xml message from the query engine into a more readable html page

```
PREFIX : <http://example.org/book/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX assert: <http://www.owl-ontologies.com/assert.owl#>
PREFIX vulnerability: <http://localhost:8080/joseki/vulnerability.owl#>

SELECT ?explosive_substances ?IsolateSmallv
FROM <http://localhost:8080/joseki/vulnerability.owl>
WHERE
{ ?explosive_substances vulnerability:hasIsolateSmall ?IsolateSmall .}
```

5-15 result of semantic query

The sparql process or provides an interface to that ontology, users can interact with the query processor and ask it to retrieve data. To do so, users need to write in the query in a specific format, insert it in the query field on the webpage and request the results.

The results of this are then displayed on an html page that is created by the query processor. In this example an html page was used to present the data, but also xml could have been used. By using XML, compatibility would have been reached with the other packages as the XML message is in such a format that it can be read, parsed and send to the PostGIS database in order to select geospatial features and present this through, Geoserver and Mapbuilder in the map viewer of the application.

SPARQLer Query Results

explosive_substances	IsolateSmallv
<http://localhost:8080/joseki/vulnerability.owl#TNT>	
<http://localhost:8080/joseki/vulnerability.owl#C4>	
<http://localhost:8080/joseki/vulnerability.owl#gasoline>	
<http://localhost:8080/joseki/vulnerability.owl#chlorine>	
<http://localhost:8080/joseki/vulnerability.owl#sulphur_dioxide>	
<http://localhost:8080/joseki/vulnerability.owl#ammonia>	

5-16 results of sparql query in html format

At this stage the “semantic” part of the demonstration is limited to querying an ontology (in owl format) and retrieving information from it. Rule based reasoning, i.e. finding items in an ontology based on derived information, is certainly possible by extending Joseki further with a reasoning engine like pellet or Racerpro. Hints on how to proceed are given on some pages on the internet²³. Real documentation on how to use it in an application however, is lacking.

The extension of the demonstrator is therefore possible, by installing an application under the same server as where the other parts of the demonstrator were installed. In theory this could be made to work with the other applications as well by having the XML messages from and to the query/ reasoning engine.

²³ <http://clarkparsia.com/weblog/category/semweb/rules/>

6. Summary and Conclusion

The main objective of this study was to demonstrate geospatial interoperability. This objective has been reached by gluing together several elements on work for a number of integrated projects in which emergency response was a central theme. This has resulted in demonstrator that can be viewed and used online on the following address: <http://vatdis.jrc.it:8080/vatdis>.

One of the main requirements has been to achieve this objective of interoperability was by using an open architecture and open standards based software packages. In the collection of available packages a choice has been made for a number of packages that, based on our user-requirements were deemed to be the most appropriate. Starting with the choice for a platform for web processing services (52 North) and how to interact with it a number of other packages were chosen that were compatible with this first choice.

Implementation:

On the implementation the following can be said:

- creating a routing web service

A routing service has been implemented based on open source software within an open architecture.

- creating a web service for a chemical dispersion model

Based on the UN list of chemicals, a simplified chemical dispersion model has been implemented. This model creates the geometry of an impact area based on PostGIS functions. The model has been published on the web both as WPS compliant service and as a SOAP compliant service, making it possible to call the model from the VATDIS demonstrator but also to call it from other applications installed on other servers and to re-use it in later work.

- offer a publication service for spatial data / import external features

By using Mapbuilder and Geoserver in combination with PostGIS a complete package is offered that allows user to work with their data both locally and on the web.

- Export functionalities to Google earth

By making use of the network link possibilities in Google Earth, the same functionalities that were designed for the VATDIS demonstrator can be re-used in different contexts.

- Comply with OGC standards for interoperability, i.e. wrap services in the SOAP protocol.

The web processing service from 52 North has been wrapped in a soap based web service. In such a way interoperability is guaranteed for other soap based clients. The adherence to WMS and WFS ensures that data can easily be integrated and re-used into other applications.

- Demonstrate the possibility of extension with semantics

With the extension of the demonstrator to support semantic reasoning some problems were encountered due to the infancy of the technology. Whereas the other elements of the demonstrator (WMS/WFS, WPS, Google Earth) have reached a certain state of maturity, the technology to support semantic reasoning has not reached that yet. Programs like Joseki and Pellet do work, however documentation lacks, software is difficult to install and the number of examples on the web is scarce.

This study has demonstrated that the existing architecture can be extended to support semantic reasoning; however a full implementation has not been reached due to lack of time.

Added value created

Actors involved in an emergency response need to exchange this information in a time constrained situation. Often multiple organizations must work together and sometimes in an ad-hoc manner. Therefore data exchange and integration must be streamlined. This makes emergency response a good demonstration case; if interoperability can be demonstrated in an emergency situation, it will probably work as well in more routine risk management processes.

In this study it is demonstrated that by adhering to open standards, information that is managed at different places can be combined at a new place and easily be transformed into new information. This creates an added value for e.g. risk management as the integration of information becomes less time consuming.

Difficulties encountered

The process of integrating the work for the different IP's in theory should have worked smoothly. In practice however it was found that a number of the open source packages that provide the functionalities that were needed were not that easy to integrate: documentation was lacking and a lot of customization was needed to make the several packages work together. The final user interface differs severely from the very first one this study has started with or from the standard one offered by Mapbuilder.

The same functionalities could have been offered by ArcServer from ESRI, however this would have put the development into a direction where the infamous software lock in would have been encountered. The choices made in this study stand as good choices: the final demonstrator is extendable, was developed without paying for the software and profits from developments made in the open source community.

Conclusion

The main achievement of this study is that by adapting to an open architecture and opens standards based software packages *synergy* can be reached between a diverse number of packages. The added value is achieved by an “easy” exchange of information and data between all packages. Easy is put in between brackets here as the adaptation of open standards can proof to be less straight forward as at some points is assumed. Documentation exists but is often incomplete and to customize the open source software asks for expert knowledge.

7. Resources

PGdijkstra:

<http://cartoweb.org/downloads.html#pgdijkstra>

Mapbuilder

<http://docs.codehaus.org/display/MAP/Home>

Geoserver

<http://docs.codehaus.org/display/GEOS/Home>

PostGIS

<http://PostGIS.refrations.net/>

52North

<http://52north.org/index.php>

Axis

<http://ws.apache.org/axis/>

Joseki

<http://www.joseki.org/>

Pellet

<http://pellet.owlidl.com/>

Some additional resources have been used among which:

ArcGIS for preparing and editing data

(www.esri.com)

Qgis, preparing and editing data

(www.qgis.org)

Putty for connecting from one desktop to the server

PgAdmin for editing the data in PostGIS

WINSCP3 for secure transfer of data from a desktop to the server

Ogr2ogr

(<http://www.gdal.org/ogr/>)

8. References

Development of the Table of Initial Isolation and Protective Action Distances for the 2004 Emergency Response Guidebook, D.F. Brown, W.A. Freeman, R.A. Carhart, and M. Krumpol Argonne National Laboratory, University of Chicago, (May 2005)

Web Mapping Illustrated Using Open Source GIS Toolkits, Tyler Mitchell, O'Reilly Media, Inc. First Edition June (2005).

Hacking GoogleMaps and GoogleEarth by Martin C. Brown (Author), Wiley (2006).

Beginning Google Maps Applications with PHP and Ajax: From Novice to Professional by Michael Purvis (Author), Jeffrey Sambells (Author), Cameron Turner (Author) Apress, (2006)

Beginning MapServer: Open Source GIS Development (Expert's Voice in Open Source) by Bill Kropla, Apress (2005)

GIS for Web Developers: Adding 'Where' to Your Web Application by Scott Davis, Pragmatic Bookshelf (2007)

Google Maps Hacks by Rich Gibson, O'Reilly Media, Inc. (2006)

Annex 1: Sequence diagrams

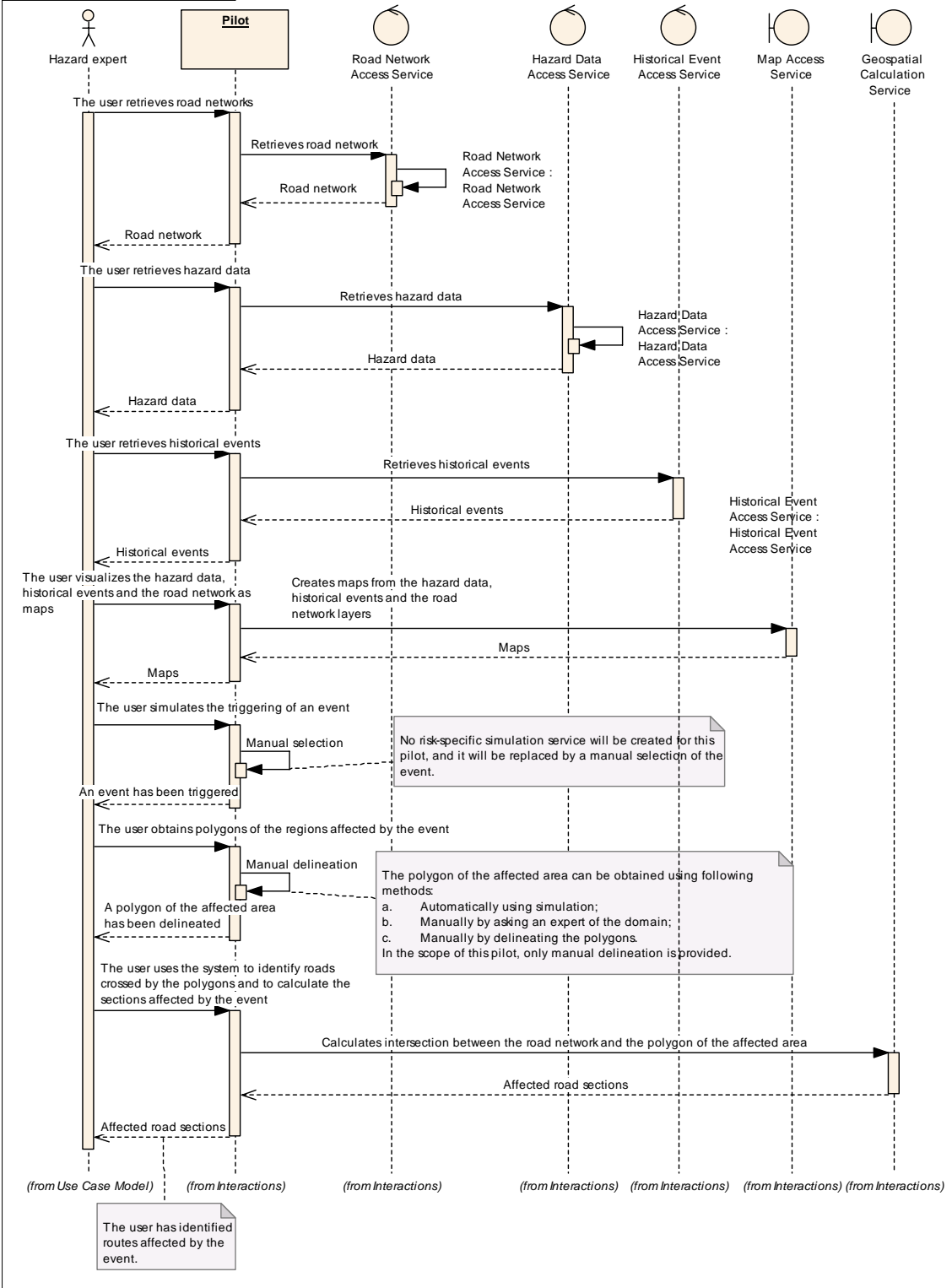
The sequence diagrams show how, in the use-case, the several services relate to each other and how each service is called.

The following sequence diagrams are presented²⁴:

1. Finding roads cut by a possible hazard
2. Finding alternative routes after a road is cut
3. Chemical cloud dispersion
4. Transport of dangerous goods

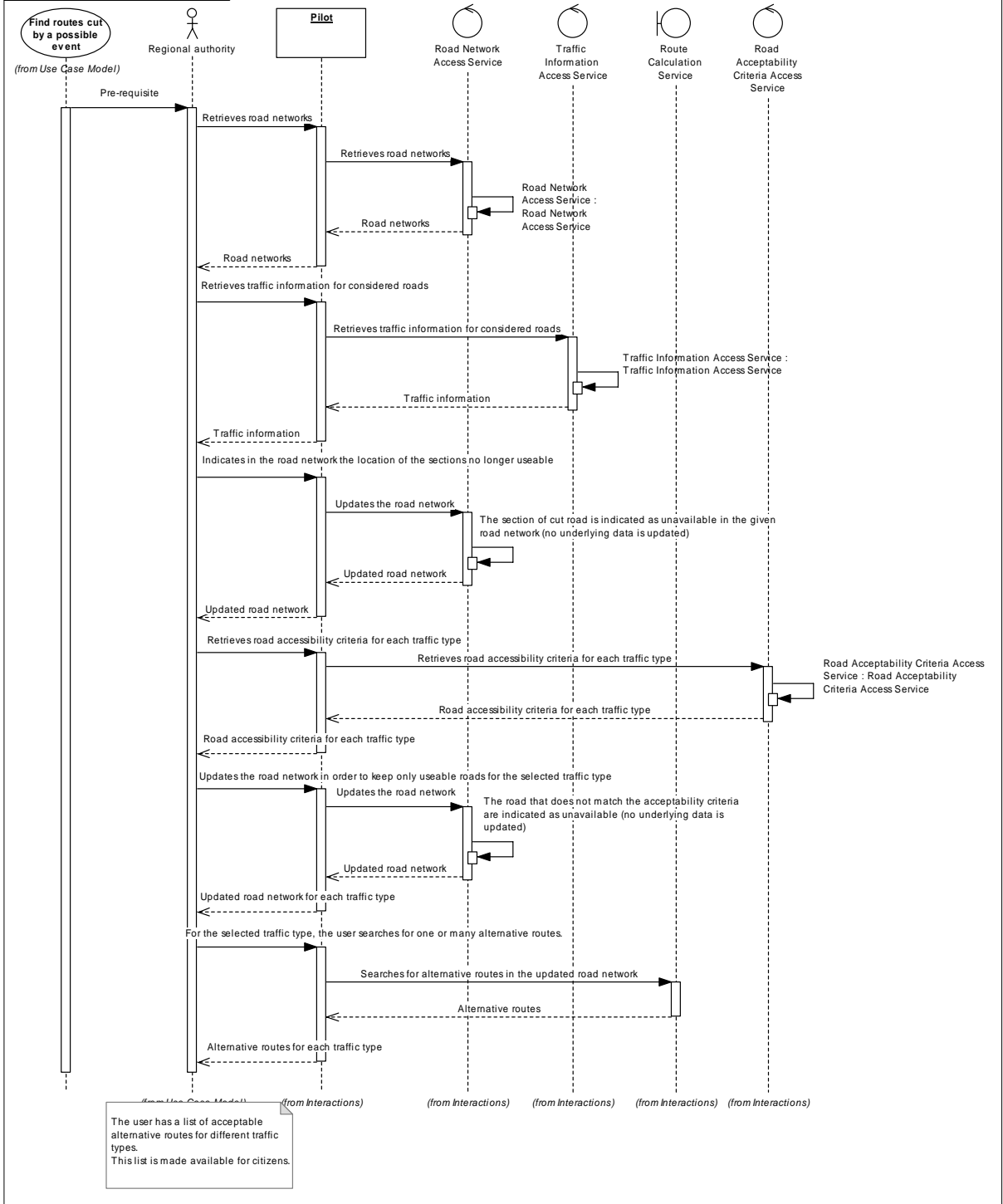
²⁴ With diagrams 1,2 & 4 taken from Orchestra Pilot description

sd Find routes cut by a possible event

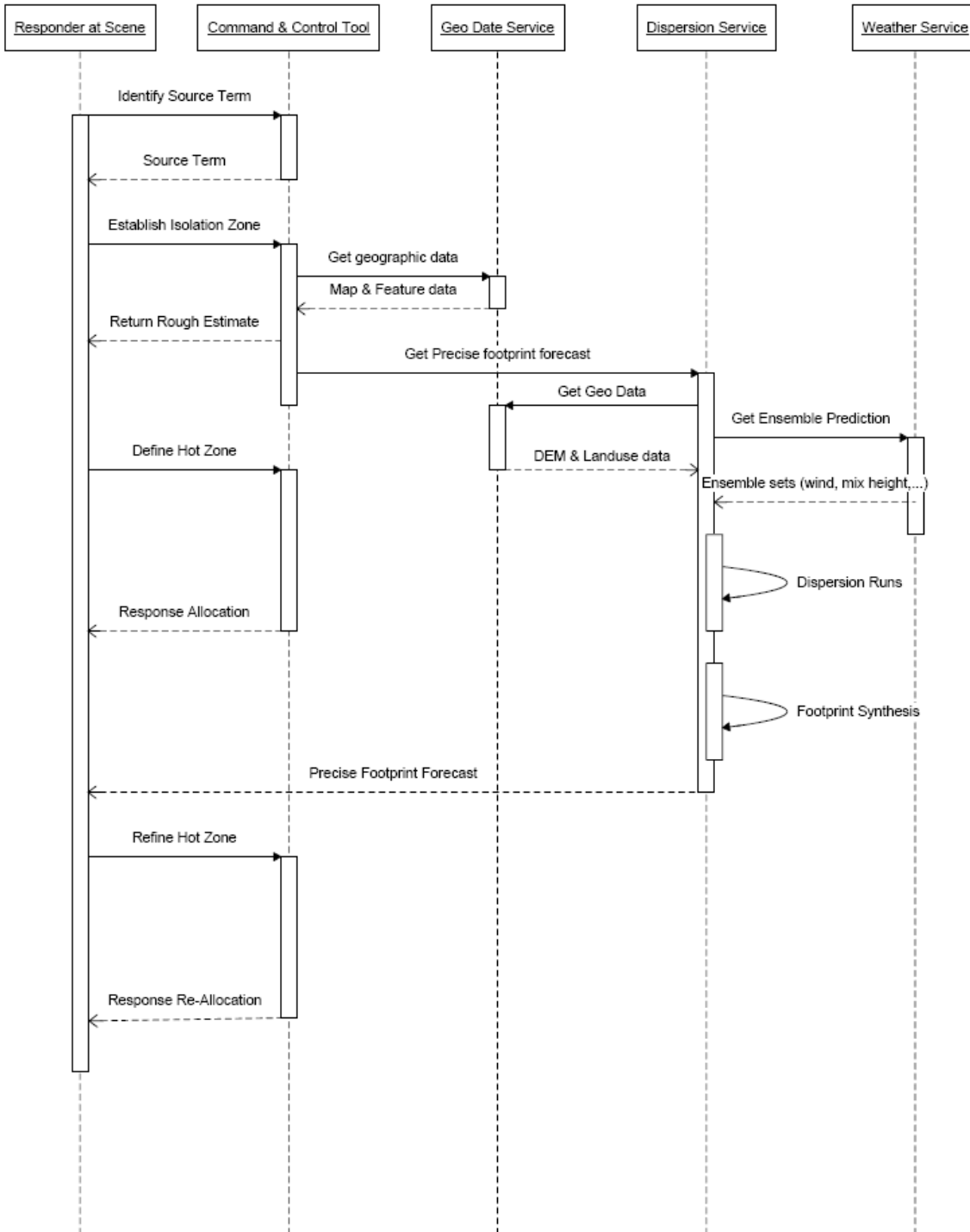


Find roads cut by a possible hazard

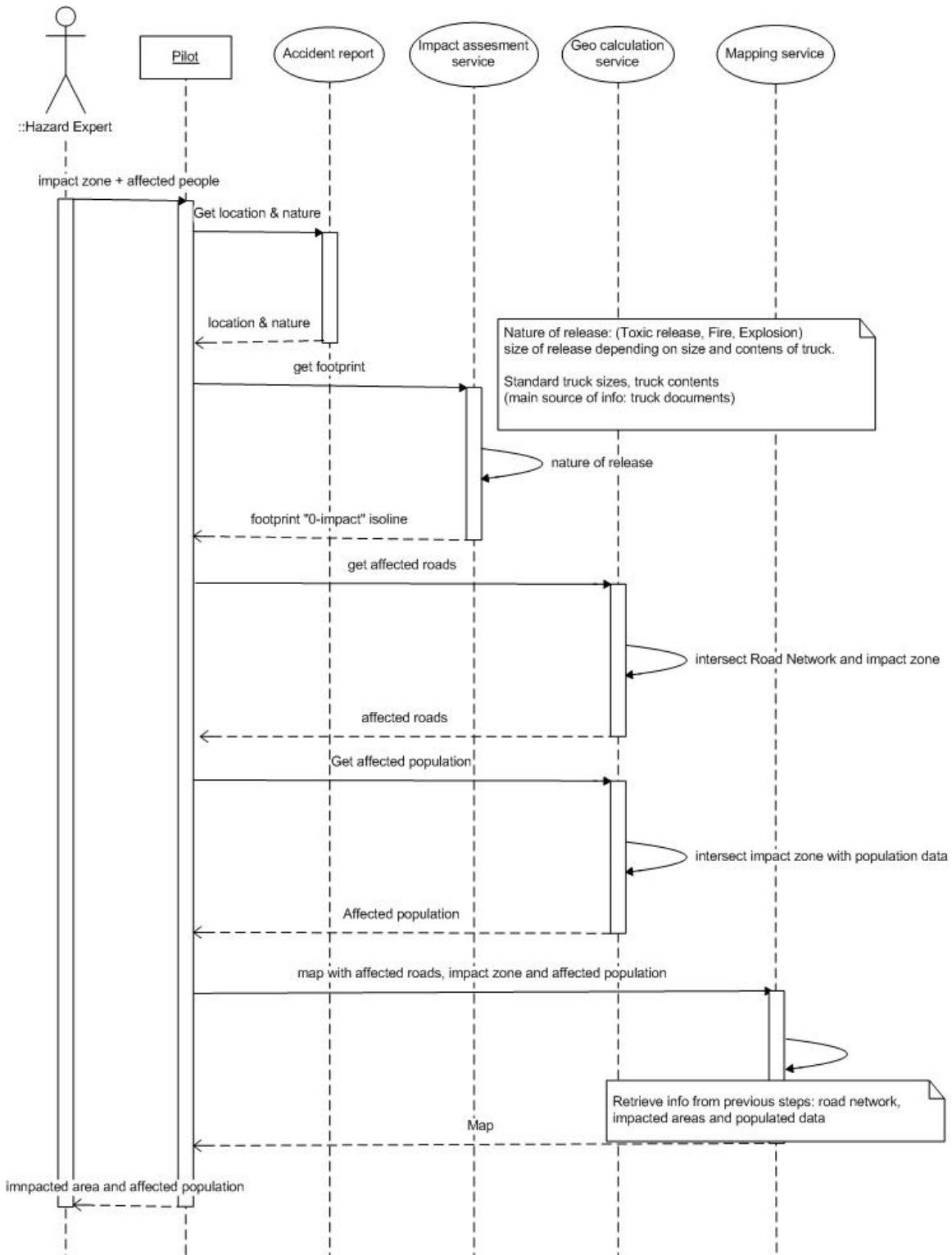
sd Find alternative routes after a route is cut



Find alternative routes after a route is cut



Chemical cloud dispersion



Transport of dangerous goods

Annex 2: Used data

The major part of the data that was used in the construction of this demonstrator comes from TeleAtlas.

The following data was used:

Event data / Historical event data

- Map showing contours of a hazard's footprint for each considered hazard type.
- Location polygon of area affected by previous event.

With respect to the two aforementioned types of data, it should be mentioned that modelling results are considered to be part of the effort as well. It is however noteworthy that the main interest in this case is in "footprints" of disasters. That is, the spatial description (by extent and parameter) of the impact a hazard might have.

Road network

- Name of road
- Type of road. Motorway, national road, departmental road etc.
- Location of roads/connectivity

Political divisions

- Location polygons of communes
- Location polygons of departments
- Location polygons of regions
- Location polygons of countries

Topographical information

- Topographic map to be used for overlaying other information and in combination with geonames to provide a gazetteer function.

Points of interest

- Data on the location of rescue services: addresses of fire brigade stations, ambulance post and police stations and other data that is deemed to be relevant in the field of emergency response.

Data on Chemicals.

- The main source for this is:
<http://www.unece.org/trans/danger/publi/adr/adr2007/07ContentsE.html>

Annex 3: Routing service

```
-- Function: vatdis_calcroute(aname text)

-- DROP FUNCTION vatdis_calcroute(aname text);

CREATE OR REPLACE FUNCTION vatdis_calcroute(aname text)
  RETURNS void AS
  $BODY$

DECLARE
  geom geometry;
  rec record;
  rec2 record;
  source_id int4 := -1;
  target_id int4 := -1;
  edgesel varchar := 'SELECT id, source, target, minutes AS cost FROM roads_edges';

BEGIN

  --Flag the route request
  UPDATE routereq SET calc=1 WHERE name=aname;

  --Calculate start and end road segments from points
  FOR rec IN (SELECT * FROM routereq WHERE name=aname) LOOP
    IF rec.type=0 THEN
      geom := buffer(rec.the_geom, 0.001);
      source_id := vatdis_roadforpoint(geom);
    ELSIF rec.type=1 THEN
      geom := buffer(rec.the_geom, 0.001);
      target_id := vatdis_roadforpoint(geom);
    END IF;
  END LOOP;

  --For every segment of the shortest path...
  FOR rec2 IN (SELECT * FROM shortest_path(edgesel, source_id, target_id, false, false)) LOOP
    --...store it in the result table
    INSERT INTO result (gid, cost, name, nname, the_geom, exproads, calc_rer, length)
      (SELECT r.gid, rec2.cost, r.name, aname, r.the_geom, 0, 0, r.meters --
length2d_spheroid(r.the_geom, 'SPHEROID["WGS_1984",6378137,298.257223563]')
      FROM roads AS r
      WHERE r.edge_id=rec2.edge_id);
  END LOOP;

  --Calculate route total length
  UPDATE routereq SET length_route = (SELECT sum(re.length) FROM result re WHERE
re.rname=aname) WHERE name=aname;

  -- update result table
  PERFORM vatdis_updateresult();

END;

$BODY$
LANGUAGE 'plpgsql' VOLATILE STRICT;
ALTER FUNCTION vatdis_calcroute(aname text) OWNER TO postgres;
```


Annex 4: Dispersion model

Creation of hazard impact zone

This is split up into two parts: calculation of the buffer and the calculation of the “square”. All the calculations done in this part are done by functions in PostGIS that either create a geometry or translate and rotate them.

For the creation of the buffer for example we use the “create buffer()” function in PostGIS.

What happens in this function? See below:

-- Function: calcbuffer(aname text)

-- DROP FUNCTION calcbuffer(aname text);

1. CREATE OR REPLACE FUNCTION calcbuffer(aname text)
2. RETURNS void AS
3. \$BODY\$
4. DECLARE
5. geom geometry;
6. rec record;
7. BEGIN

8. execute 'insert into footprint (name, typ, the_geom)
i. SELECT name, typ, (multi(buffer(the_geom, isolate_small)))
ii. FROM hazardsource
iii. WHERE name=' || quote_literal(aname) limit 1;

9. END;
10. \$BODY\$
11. LANGUAGE 'plpgsql' VOLATILE STRICT;
12. ALTER FUNCTION calcbuffer(aname text) OWNER TO postgres;

The main functionality is described in line 8. This is where the functionality is defined:

Execute	tells the system that an sql statement will follow
Insert into footprint	tells the system that a new record will be create din the table footprint and that he following field (between brackets will be inserted)
Select	tells the system from the values of the fields to eb inserted come from, play particlaur attention to the field “the_geom”:

(multi(buffer(the_geom, isolate_small))

This field tells the system that the field the_geom from hazard source²⁵ table should be altered and well in such a way that around this user defined input a buffer should be drawn with a diameter that corresponds to the value of (isolate_small)

Where name is This is the major improvement we have made with respect to the work of martin. We have changed the function in such a way that it calculates impact zone per hazard.

Then the second part:

The creation of the square according of the distance isolate_large and having it rotate according to the wind-direction.

1. Function: preview(aname text)
2. DROP FUNCTION preview(aname text);

²⁵ which is the location from the truck with dangerous goods or the fixed installation or the location where e.g. Al-Qaeda could have released a chemical

```

3. CREATE OR REPLACE FUNCTION preview(aname text)
4. RETURNS void AS
5. $BODY$
6. DECLARE
    1. geom geometry;
    2. rec record;
7. BEGIN

8. execute 'insert into footprint2 (gid, name, typ, angle, isolate_large, the_geom) SELECT gid, name, typ, angle, isolate_large,
9. translate(the_geom, isolate_large/2, 0) from hazardsource where name = '||quote_literal(aname) limit 1;

10. execute 'insert into footprint3 (gid, name, typ, angle, the_geom) select gid, name, typ, angle, (multi(Expand(the_geom,
11. isolate_large/2)))FROM footprint2 where name= '|| quote_literal(aname) limit 1;

12. execute 'update footprint3 set xhulp= (select xmin(extent(the_geom)) from footprint3 where name=||quote_literal(aname)||)' limit
1;

13. execute 'update footprint3 set yhulp= (select y(centroid(the_geom)) from footprint3 where name=||quote_literal(aname)||)' limit 1;

14. execute 'insert into footprint4 (gid, name, typ, angle, the_geom) SELECT gid, name, typ, angle, translate(
rotate(translate(the_geom,-
15. xhulp, -yhulp ), radians(angle)), xhulp, yhulp)from footprint3 where name = '|| quote_literal(aname) limit 1;

16. END;
17. $BODY$
18. LANGUAGE 'plpgsql' VOLATILE STRICT;
19. ALTER FUNCTION preview(aname text) OWNER TO postgres;

```

Again the first seven lines just tell the system that a certain function will be made and that some parameters will be parsed in to the function. A bit shorter explanation because some of it is pure repetition:

Lines 8 and 9 tell the system that location of the hazard source will be translated to a new midpoint of a geometry and this point is translated over exactly ($1/2 * \text{length of square}$, 0).

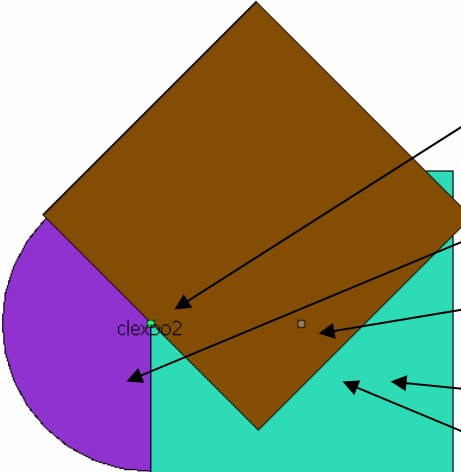
Lines 10 and 11 then tell the system that it should create a square around this point with the the length of isolate_large.

Line 12 extracts the x coordinate of point of rotation

Line 13 extracts the y coordinate of the point of rotation

Line 14 and 15 tell the system to translate the rotation point of the geometry to the origin (0,0) and rotate it there according to the winddirection (which the user has defined and is stored in the column "angle") and after that translate it back to the original location.

In quantum GIS it looks like this:



Input from user: hazardsource with source description (green dot)

Creation of buffer according to `calcbuffer(text)` (purple circle)

Creation of midpoint of new square (brown dot)

Creation square (green square)

Rotation of square (brown square)

Annex 5: Google Earth

The possibility to work with the VATDIS functionalities in Google Earth are based on being able to parse the location of the object a user is looking at in Google Earth to the Postgis database. The location of that object is described in longitude and latitude and these are sent to the PostGIS database by a java server page.

The network link

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Folder>
    <name>VATDIS Toolkit</name>
    <visibility>1</visibility>
    <open>1</open>
    <Snippet maxLines="2"></Snippet>
    <description><![CDATA[VATDIS Toolkit]]></description>
    <NetworkLink>
      <name>View-centered placemark</name>
      <visibility>1</visibility>
      <open>1</open>
      <Snippet maxLines="2"></Snippet>
      <description><![CDATA[The view-based refresh allows the remote server to
calculate the center of your screen and return a placemark.]]></description>
      <refreshVisibility>0</refreshVisibility>
      <flyToView>0</flyToView>
      <Link>
        <href>http://vatdis.jrc.it:8080/vatdis/googleearth/centeredplacemark.jsp</href>
        <!--
<href>http://vatdis.jrc.it:8080/wps/demo/googleearth/centeredplacemark.php</href> -->
        <RefreshMode>onChange</RefreshMode>
        <refreshInterval>1</refreshInterval>
        <viewRefreshMode>onStop </viewRefreshMode>
        <viewRefreshTime>1</viewRefreshTime>
        <viewBoundScale>1</viewBoundScale>
        <viewFormat>BBOX=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth]</viewFormat>
      </Link>
    </NetworkLink>
  </Folder>
  <Folder>
    <name>Geoserver Layers</name>
    <visibility>1</visibility>
    <open>0</open>
    <Snippet maxLines="2"></Snippet>
    <description><![CDATA[Geoserver
Layers]]></description>
    <!-- Hazardsources -->
    <NetworkLink>
      <name>Hazardsources</name>
      <visibility>1</visibility>
      <open>0</open>
      <Snippet maxLines="2"></Snippet>
      <description></description>
      <refreshVisibility>0</refreshVisibility>
      <flyToView>0</flyToView>
      <Link>
        <href>http://vatdis.jrc.it:8080/Geoserver/wms/kml_reflect?layers=trva:hazardsource&srs=EPSG
:4326</href>
        <RefreshMode>onChange</RefreshMode>
        <refreshInterval>1</refreshInterval>
        <viewRefreshMode>onStop </viewRefreshMode>
        <viewRefreshTime>3</viewRefreshTime>
        <viewBoundScale>1</viewBoundScale>
      </Link>
    </NetworkLink>
    <!-- Exposed Roads -->
    <NetworkLink>
      <name>Exposed Roads</name>
      <visibility>1</visibility>
      <open>0</open>
```

```

        <Snippet maxLines="2"></Snippet>
        <description></description>
        <refreshVisibility>0</refreshVisibility>
        <flyToView>0</flyToView>
        <Link>
            <href>http://vatdis.jrc.it:8080/Geoserver/wms/kml_reflect?layers=trva:roadsexposed&srs=EPSG:4326</href>
            <RefreshMode>onChange</RefreshMode>
            <refreshInterval>1</refreshInterval>roadsexposed
            <viewRefreshMode>onStop </viewRefreshMode>
            <viewRefreshTime>3</viewRefreshTime>
            <viewBoundScale>1</viewBoundScale>
        </Link>
    </NetworkLink>
    <!-- Impactarea -->
    <NetworkLink>
        <name>Impactarea</name>
        <visibility>1</visibility>
        <open>0</open>
        <Snippet maxLines="2"></Snippet>
        <description></description>
        <refreshVisibility>0</refreshVisibility>
        <flyToView>0</flyToView>
        <Link>
            <href>http://vatdis.jrc.it:8080/Geoserver/wms/kml_reflect?layers=trva:footprint_union&srs=EPSG:4326</href>
            <RefreshMode>onChange</RefreshMode>
            <refreshInterval>1</refreshInterval>
            <viewRefreshMode>onStop </viewRefreshMode>
            <viewRefreshTime>3</viewRefreshTime>
            <viewBoundScale>1</viewBoundScale>
        </Link>
    </NetworkLink>
    <!-- Impactarea - FootprintSmall -->
    <NetworkLink>
        <name>Impactarea - FootprintSmall</name>
        <visibility>1</visibility>
        <open>0</open>
        <Snippet maxLines="2"></Snippet>
        <description></description>
        <refreshVisibility>0</refreshVisibility>
        <flyToView>0</flyToView>
        <Link>
            <href>http://vatdis.jrc.it:8080/Geoserver/wms/kml_reflect?layers=trva:footprint&srs=EPSG:4326</href>
            <RefreshMode>onChange</RefreshMode>
            <refreshInterval>1</refreshInterval>
            <viewRefreshMode>onStop </viewRefreshMode>
            <viewRefreshTime>3</viewRefreshTime>
            <viewBoundScale>1</viewBoundScale>
        </Link>
    </NetworkLink>
    <!-- Impactarea - FootprintLarge -->
    <NetworkLink>
        <name>Impactarea - FootprintLarge</name>
        <visibility>1</visibility>
        <open>0</open>
        <Snippet maxLines="2"></Snippet>
        <description></description>
        <refreshVisibility>0</refreshVisibility>
        <flyToView>0</flyToView>
        <Link>
            <href>http://vatdis.jrc.it:8080/Geoserver/wms/kml_reflect?layers=trva:footprint4&srs=EPSG:4326</href>
            <RefreshMode>onChange</RefreshMode>
            <refreshInterval>1</refreshInterval>
            <viewRefreshMode>onStop </viewRefreshMode>
            <viewRefreshTime>3</viewRefreshTime>
            <viewBoundScale>1</viewBoundScale>
        </Link>
    </NetworkLink>
</Folder>
</Folder>
</kml>

```

This gets downloaded in it's complete in to the Google Earth toolbox. The red lines in the text form the section in the code where the middle point of the screen is calculated. The blue lines in the code are the points where a link is made to the layers that are created using PostGIS functions. These layers are served by Geoserver.

From this link the bounding box is send to a java server page where it is read:

```
<%// split the client's BBOX return by commas and spaces to obtain an array of coordinates
    bbox = request.getParameter("BBOX");
    [] coords = bbox.split(",");

    //parameters from request Uri
    west = Float.parseFloat(coords[0]);
    south = Float.parseFloat(coords[1]);
    east = Float.parseFloat(coords[2]);
    north = Float.parseFloat(coords[3]);

    // calculate the approx center of the view -- note that this is innaccurate if the user
    is not looking straight down
    center_lon = ((east - west)/2) + west;
    center_lat = ((north - south)/2) + south;

    kmlString = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"
+ "<kml xmlns=\"http://earth.google.com/kml/2.1\">\n"
+ "<Placemark>\n"
+ "    "
+ "        <name><![CDATA[<a
href=\"http://vatdis.jrc.it:8080/vatdis/googleearth/form.jsp?lat="+center_lat+"&lon="+cente
r_lon+"\">Calculate footprint using PostGIS function</a>]]></name>\n"
+ "        <description><![CDATA[The view-based refresh allows the remote server to calculate the
center of your screen and return a placemark.]]></description>\n"
+ "        <Style>\n"
+ "            <IconStyle>\n"
+ "                <color>ffefebde</color>\n"
+ "                <Icon>\n"
+ "                    <href>http://maps.google.com/mapfiles/kml/pal3/icon52.png</href>\n"
+ "                </Icon>\n"
+ "            </IconStyle>\n"
+ "            <LabelStyle>\n"
+ "                <scale>0</scale>\n"
+ "            </LabelStyle>\n"
+ "        </Style>\n"
+ "        <Point>\n"
+ "            <coordinates>"+center_lon+", "+center_lat+"</coordinates>\n"
+ "        </Point>\n"
+ "</Placemark>\n"
+ "</kml>\n";
    .println(kmlString);%>
```

This piece of reads the bounding box the user is looking at, calculates the middle point of the bonding box (blue sections) and creates a kml file which places a place mark on that point (red section) this last bit is dynamic, which ensures the middle point is more or less dragged by the user. This points in itself I also again a network ink, it points to another java page (green section) that opens up a form (see figure on page nr)

```
<html>
<head>
    <script type="text/javascript" language="javascript">
    checkform() {
        (document.fform.name.value == "") {
            ('Name field can not be empty');
            false;
        }
        true;
    }
    </script>
</head>
<body style="background-color:#e1e1e1; color:#556655; font-family:arial,Helvetica,sans-serif;">

<%
GElat = request.getParameter("lat");
GElon = request.getParameter("lon");
%>
```

```

<div style="border:1px solid #556655; padding:0px 10px 0px 10px; margin-top:20px; margin-right:auto; margin-left:auto; width:352px;">
  <form name="fform" method="get" action="postgisfunction.jsp" onSubmit="return checkform()">
    <br/>
    <label for="name">1) Insert TIC name:</label>
    <input type="text" name="name" size="32">
    <br/><br/>
    <label for="un">2) Select UN number:</label>
    <select name="un" size="1" style="width:200px;">
      <option value="1005" selected="selected">1005 - Ammonia, anhydrous</option>
      <option value="1016">1016 - Carbon monoxide</option>
      <option value="1017">1017 - Chlorine</option>
      <option value="1051">1051 - Hydrogen cyanide, anhydrous, stabilized</option>
      <option value="1052">1052 - Hydrogen fluoride, anhydrous</option>
      <option value="1053">1053 - Hydrogen sulfide</option>
      <option value="1076">1076 - Phosgene</option>
      <option value="1079">1079 - Sulphur dioxide</option>
      <option value="1082">1082 - Trifluorochloroethylene</option>
      <option value="1092">1092 - Acrolein, inhibited</option>
      <option value="1251">1251 - Methyl vinyl ketone</option>
      <option value="1581">1581 - Chloropicrin and Methyl bromide mixture</option>
      <option value="1955">1955 - Compressed gas, poisonous, n.o.s.</option>
      <option value="2013">2013 - Strontium phosphide (when spilled in water)</option>
      <option value="2190">2190 - Oxygen difluoride</option>
      <option value="2480">2480 - Methyl isocyanate</option>
      <option value="2810">2810 - Mustard (when used as a weapon)</option>
      <option value="2810">2810 - Sarin (when used as a weapon)</option>
    </select>
    <br/><br/>
    <label for="wind">3) Select wind direction:</label>
    <select name="wind" size="1" style="width:50px;">
      <option value="90" selected="selected">N</option>
      <option value="45">NE</option>
      <option value="0">E</option>
      <option value="315">SE</option>
      <option value="270">S</option>
      <option value="225">SW</option>
      <option value="180">W</option>
      <option value="135">NW</option>
    </select>
    <br/><br/>
    <label for="spill_time">3) Select spill time:</label>
    <input type="radio" name="spill_time" id="spill_time_day" value="0" checked="checked"/>
    <label for="spill_time_day">day</label>
    <input type="radio" name="spill_time" id="spill_time_night" value="1"/>
    <label for="spill_time_night">night</label>
    <br/><br/>
    <label for="spill_size">3) Select spill size:</label>
    <input type="radio" name="spill_size" id="spill_size_small" value="0" checked="checked"/>
    <label for="spill_size_small">small</label>
    <input type="radio" name="spill_size" id="spill_size_large" value="1"/>
    <label for="spill_size_large">large</label>

    <input type="hidden" name="lat" size="30" value="<%= GElat %>">
    <input type="hidden" name="lon" size="30" value="<%= GElon %>">
    <br/><br/><br/>
    <table border="0" cellspacing="0" cellpadding="0" width="100%">
      <tr>
        <td align="left"><input type="submit" value="Calculate footprint"></td>
        <td align="right"><input type="reset" value="Reset"></td>
      </tr>
    </table>
  </form>
</div>
</body>
</html>

```

The yellow sections are the key elements in this piece of code. Here the longitude and latitude are put in http request together with the source description the use enters in the form.jsp. This page gets send again to another page where the are linked to postgis functions (blue section below)

```

<%@ page
= " java.sql.* "
%>
<%

```

```

{
.forName("org.postgresql.Driver");
{
//parameters from url
name = request.getParameter("name");
un = request.getParameter("un");
wind = request.getParameter("wind");
time = request.getParameter("spill_time");
size = request.getParameter("spill_size");
GElat = request.getParameter("lat");
GElon = request.getParameter("lon");

// Connecting, selecting database
dbconn = DriverManager.getConnection("jdbc:postgresql://vatdis.jrc.it:5432/RoadNetwork",
"postgres", "postgres2006database");

// Performing SQL query1
query1 = "INSERT INTO hazardsource (name,un,the_geom,angle,spill_type,spill_time) VALUES
('"+name+"', '"+un+"',         GeomFromText('POINT('"+GElon+"          '"+GElat+"')',
4326),"+wind+", "+size+", "+time+)";
pstmt1 = dbconn.prepareStatement(query1);
numUpd = pstmt1.executeUpdate();
.close();

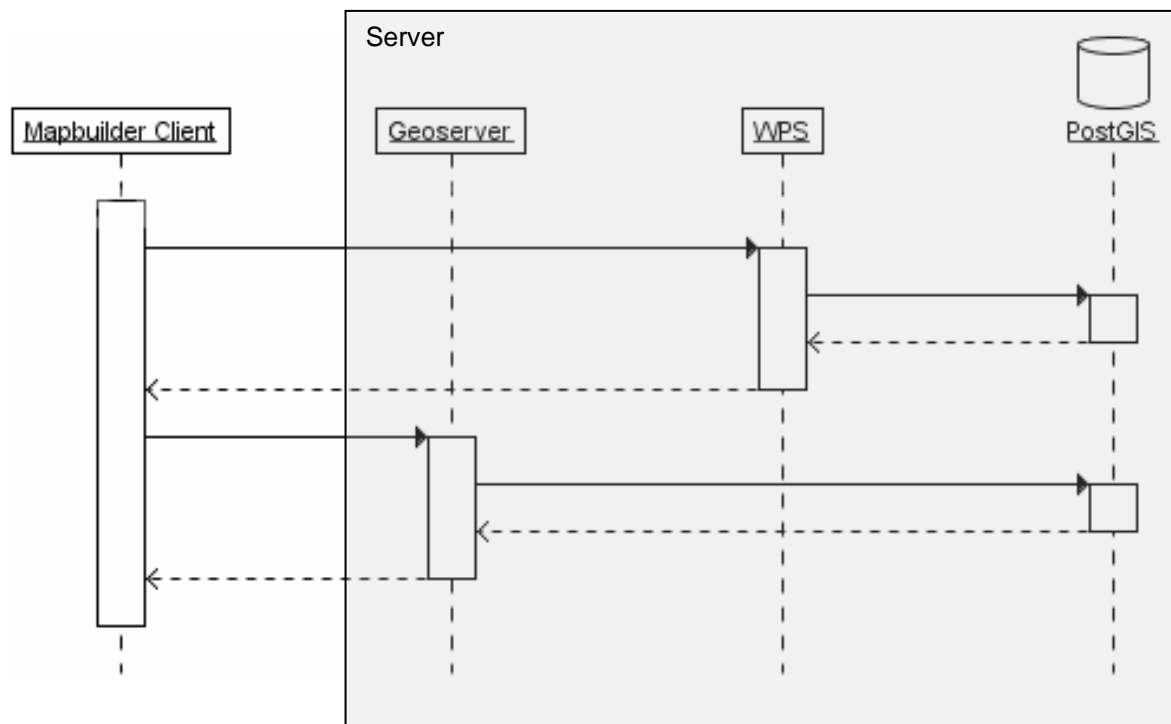
// Performing SQL query2
query2 = "SELECT vatdis_calcfootprint(CAST('"+name+"' AS text)); stmt2 =
dbconn.prepareStatement(query2);
rs2 = stmt2.executeQuery();
.close();
.close();
//close the connection
.close();
}
(SQLException s)
{
.println(s);
}
}
(ClassNotFoundException e)
{
.println(e);
}
%>
<html>
<body style="background-color:#e1e1e1; color:#556655; font-family:arial,Helvetica,sans-serif;">

</body>
</html>

```


Annex 6: WPS code snippets

This annex aims to explain in more detail how the web processing service of 52 north has been implemented and how it works together with the functions that are implemented in the PostGIS database.



The figure above gives an overview on how the integration of these two packages works. A Mapbuilder based client (or any other client that supports transactional web feature services) sends gml to Geoserver. Geoserver translates this then in to coordinates which are parsed to the relevant PostGIS functions by the web processing service. The PostGIS function then stores the results into a table which is served by Geoserver and the features in the table can then be viewed online by the Mapbuilder client.

The whole process therefore takes place in a few

1. Client side
2. Server side
3. PostGIS

Client side (Javascript)

```
/**
 * calculate footprint for selected hazard
 */

function calcFootprint(hazard) {
    URL_WPS = "http://vatdis.jrc.it:8080/wps/WebProcessingService";
    XML = makeDbFunctionCall('vatdis_calcfootprint', hazard);
    getXML(URL_WPS, XML);
}
```

This is the start of the sequence of actions that lead to the calculations of a footprint. Within the client side javascript a function is introduced with the name “calcfootprint”, the parameter of this function is simply the name of hazard that appears in the list of hazards stored in the PostGIS database.

Take note that the actual sending of the hazard source information and the calculation of a footprint are two separate functions! The first one is web processing service where the latter is a transactional web feature service.

This step described which functions should be called and what the URL of the web processing service is. In the next step the actual xml message for the web processing service is composed.

```
/**
 * Call a generic db function (fun) with a parameter (par) through WPS
 * service RouterDBInterface
 */

function makeDbFunctionCall(fun, par) {
    XML = '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>\n';
    XML += '<Execute service="WPS" version="0.4.0" store="true" status="true"\n';
    XML += 'xmlns="http://www.openeospatial.net/wps"\n';
    XML += 'xmlns:ows="http://www.openeospatial.net/ows"\n';
    XML += 'xmlns:xlink="http://www.w3.org/1999/xlink" \n';
    XML += 'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"\n';
    XML += 'xsi:schemaLocation="http://www.openeospatial.net/wps ..\wpsExecute.xsd">\n';
    XML += '    <ows:Identifier>org.jrc.wps.routing.RouterDBInterface</ows:Identifier>\n';
    XML += '    <DataInputs>\n';
    XML += '        <Input>\n';
    XML += '            <ows:Identifier>FUNCTION</ows:Identifier>\n';
    XML += '            <ows:Title>The database function to call</ows:Title>\n';
    XML += '            <LiteralValue dataType="xs:string">\n';
    XML += '                '+ fun +' \n';
    XML += '            </LiteralValue>\n';
    XML += '        </Input>\n';
    XML += '        <Input>\n';
    XML += '            <ows:Identifier>PARAMETER</ows:Identifier>\n';
    XML += '            <ows:Title>The value of the input parameter</ows:Title>\n';
    XML += '            <LiteralValue dataType="xs:string">\n';
    XML += '                '+ par +' \n';
    XML += '            </LiteralValue>\n';
    XML += '        </Input>\n';
    XML += '    </DataInputs>\n';
    XML += '</Execute>\n';

    return XML;
}
```

In the second step of the client side javascript the function is actually described. The name of the function to be called is parsed as well as its parameter. In this step an XML message is composed (See front of lines “XML”)

```
/**
 * Send the XML request to the server
 */
```

```

function getXML(url,docToSend) {
    if (window.ActiveXObject) {
        geo_xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");    //IE
    } else if (window.XMLHttpRequest) {
        geo_xmlhttp = new XMLHttpRequest();    //Mozilla
    }
    geo_xmlhttp.open("POST", url, true);
    geo_xmlhttp.setRequestHeader('Content-Type', 'text/xml');
    geo_xmlhttp.send(docToSend);
}

```

This is the final part of the client side scripting. This is the place where the XML message that was composed in the previous step is send to web processing service by making use of geo_xmlhttp request.

Server side

WPS (Java)

```

/**
 * RouterDBInterface.java
 */

package org.jrc.wps.routing;
import org.n52.wps.server.*;
import java.io.*;
import java.net.*;
import java.sql.*;
import java.util.*;
import javax.naming.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.*;
import javax.sql.*;
import javax.servlet.http.*;
import com.vividsolutions.jump.feature.Feature;
import com.vividsolutions.jump.feature.FeatureCollection;

public class RouterDBInterface extends AbstractAlgorithm {

    DataSource ds = null;
    StringBuffer sb;
    Context initCtx = null;

    public RouterDBInterface() {
        super();
    }

    public HashMap run(HashMap layers, HashMap parameters) {

        sb = null;
        HashMap result = new HashMap();

        try {
            if (ds == null) {
                initCtx = new InitialContext();
                Context envCtx = (Context) initCtx.lookup("java:/comp/env");
                ds = (DataSource) envCtx.lookup("jdbc/RiskDB");
            }

            Connection conn = ds.getConnection();
            String function = (String)parameters.get("FUNCTION");
            String param = (String)parameters.get("PARAMETER");
            PreparedStatement ps;

            if (param != null && param.length() > 0) {
                ps = conn.prepareStatement("select "+function+" (cast('"+param+"' as text))");
            } else {
                ps = conn.prepareStatement("select "+function+"()");
            }

            ResultSet rs = ps.executeQuery();

```

```

        ResultSetMetaData rsmd = rs.getMetaData();

        String message = "Query returned "+rsmd.getColumnCount()+" columns";
        result.put("RESULT", message);

        conn.close();

    } catch (SQLException e) {

        sb = new StringBuffer("");
        sb.append("ERROR:"+e.toString()).append("");
        result.put("RESULT", sb.toString());

    } catch (NamingException e) {

        sb = new StringBuffer("");
        sb.append("ERROR:"+e.toString()).append("");
        result.put("RESULT", sb.toString());

    }

    return result;
}
}

```

This is the implementation of the 52 web processing service. It was constructed using eclipse and it is designed in such a way that it makes a bridge between the input from the user, on the client side, and the PostGIS functions on the server side.

What happens in this step is that the XML message that was send by the client in the final step of the client side scripting is no read by the Web processing services. This is just a java class that is installed under tomcat. The yellow and the blue colored texts formt he continuation form the step before. Here, the function and the parameter actually get filtered out form the XML message and this is send to the PostGIS database.

POSTGIS (SQL)

```

CREATE OR REPLACE FUNCTION vatdis_calcfootprint(aname text)
  RETURNS void AS
  $BODY$

BEGIN

  -- calculate circle footprint
  PERFORM vatdis_calcfootprintcircle(aname);

  -- calculate square footprint
  PERFORM vatdis_calcfootprintsquare(aname);

  -- union of circle and square footprints
  INSERT INTO footprint_union (name, un, the_geom)
    SELECT f4.name, f4.un, multi(geomunion(f4.the_geom, f.the_geom))
    FROM footprint AS f, footprint4 AS f4
    WHERE f4.name = aname
    AND f.name = aname;

  -- perimeter and area (France projection)
  UPDATE footprint_union SET
    area_m2 = (area2d(transform(the_geom, 27582))),
    perimeter_m = (perimeter2d(the_geom)*100000)
    WHERE name = aname;

  -- flag hazardsource as calculated
  UPDATE hazardsource SET calc=1, calc_exproads=calc_exproads WHERE name=aname;

END;
$BODY$

```

```
LANGUAGE 'plpgsql' VOLATILE STRICT;
ALTER FUNCTION vatdis_calcfingerprint(text) OWNER TO postgres;
```

This is the main element of the chemical dispersions model that was implemented in PostGIS. It actually consists of a function that is built based on two other function: one that calculates the square and one that calculates the circle. However the main thing is that the name of the source and the type of function that were colored in both yellow and blu all throughout the code snippets actually come back here.

```
CREATE OR REPLACE FUNCTION vatdis_calcfingerprintcircle(aname text)
  RETURNS void AS
  $BODY$

BEGIN

  INSERT INTO footprint (name, un, the_geom)
  SELECT
    h.name,
    h.un,
    (multi(buffer(h.the_geom, r.isolate_small)))
  FROM
    hazardssource AS h,
    release_distance2 AS r
  WHERE
    r.un = h.un
  AND
    h.name = aname;

END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE STRICT;
ALTER FUNCTION vatdis_calcfingerprintcircle(text) OWNER TO postgres;
```

```
CREATE OR REPLACE FUNCTION vatdis_calcfingerprintsquare(aname text)
  RETURNS void AS
  $BODY$

BEGIN
  -- calculate square footprint
  INSERT INTO footprint2 (name, un, angle, isolate_large, the_geom)
  SELECT h.name, h.un, h.angle, r.isolate_large, translate(h.the_geom, r.isolate_large/2, 0)
  FROM hazardssource AS h, release_distance2 AS r
  WHERE r.un = h.un
  AND h.name = aname;

  INSERT INTO footprint3 (name, un, angle, isolate_large, the_geom)
  SELECT name, un, angle, isolate_large, (multi(Expand(the_geom, isolate_large/2)))
  FROM footprint2
  WHERE name = aname;

  UPDATE footprint3 SET xhulp = (SELECT xmin(extent(the_geom))
    FROM footprint3
    WHERE name = aname);

  UPDATE footprint3 SET yhulp = (SELECT y(centroid(the_geom))
    FROM footprint3
    WHERE name = aname);

  INSERT INTO footprint4 (name, un, angle, isolate_large, the_geom)
  SELECT f3.name, f3.un, f3.angle, f3.isolate_large, translate( rotate( translate( f3.the_geom,
-f3.xhulp, -f3.yhulp ), radians(f3.angle)), f3.xhulp, f3.yhulp)
  FROM footprint3 AS f3
  WHERE f3.name = aname;

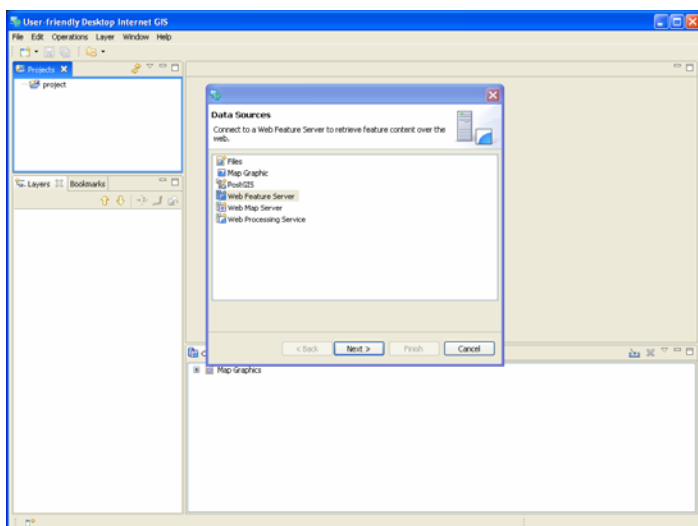
END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE STRICT;
ALTER FUNCTION vatdis_calcfingerprintsquare(text) OWNER TO postgres;
```

What is important to see in this combination of client and server side scripting is that we start by creating a function in PostGIS and have this work together with input coming from a table. This table is preferably populated with data coming from a transactional web feature service.

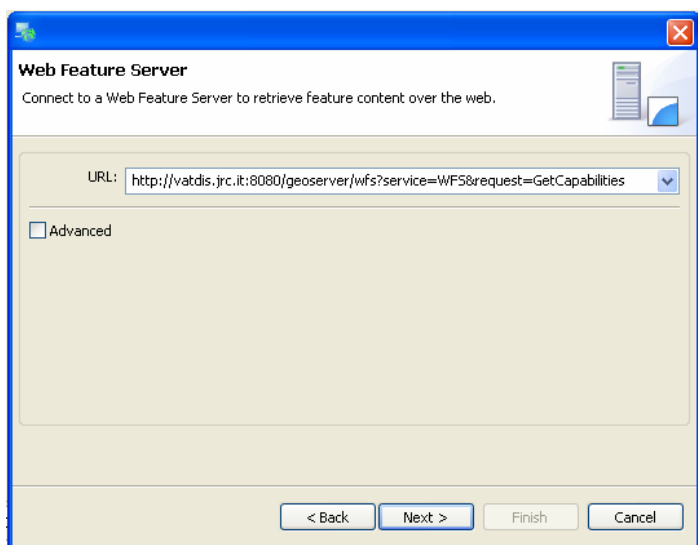
Then we can build a wrapper around all this, by using the WPS. Which is nothing more than a mechanism to make the PostGIS function compatible with XML input. The XML message is itself is composed using a standard format.

The following sequence of screen shots shows how users can interact with the WPS using uDig²⁶. What happens in this sequence is that a user start the uDig interface, connect with a WFS server where he selects features and ask the WPS to build a footprint around those features. For this footprint, the user again has to fill in the source terms.

Step one: a users loads WFS data from a point layers in geoserver.:

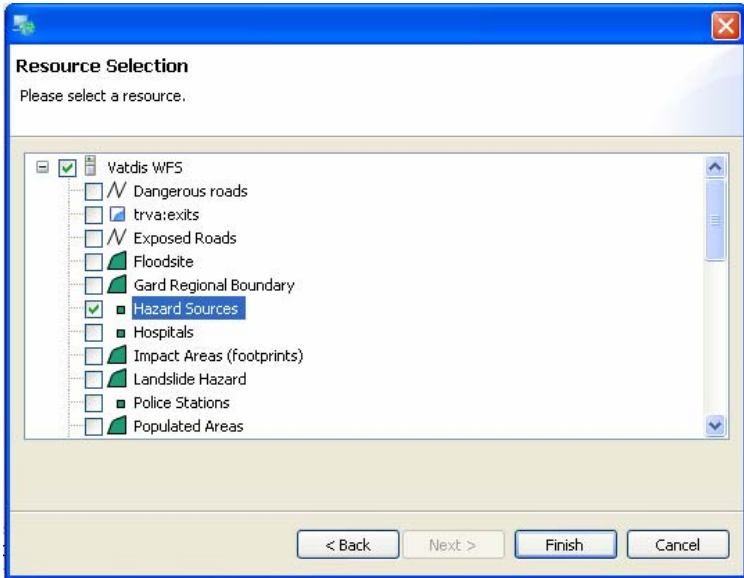


Step 1a:
initiate Udig
toolbox and
add layers to
the map



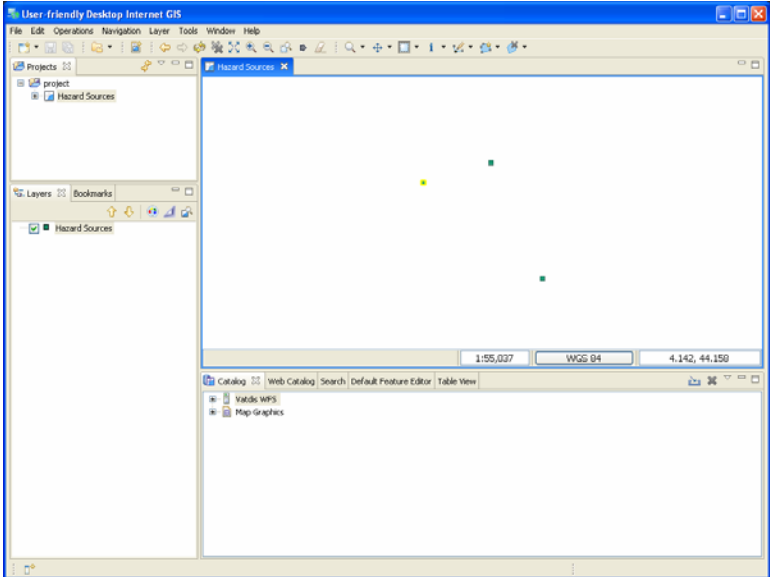
Step 1b: point
to a WFS
compliant
server

²⁶ <http://udig.refrations.net/confluence/display/UDIG/Home>



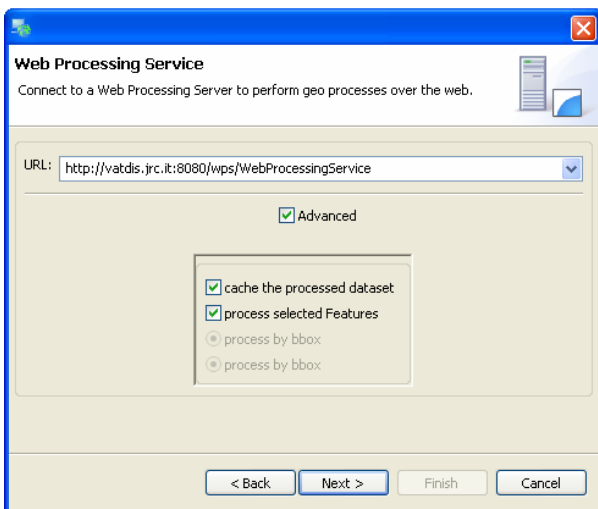
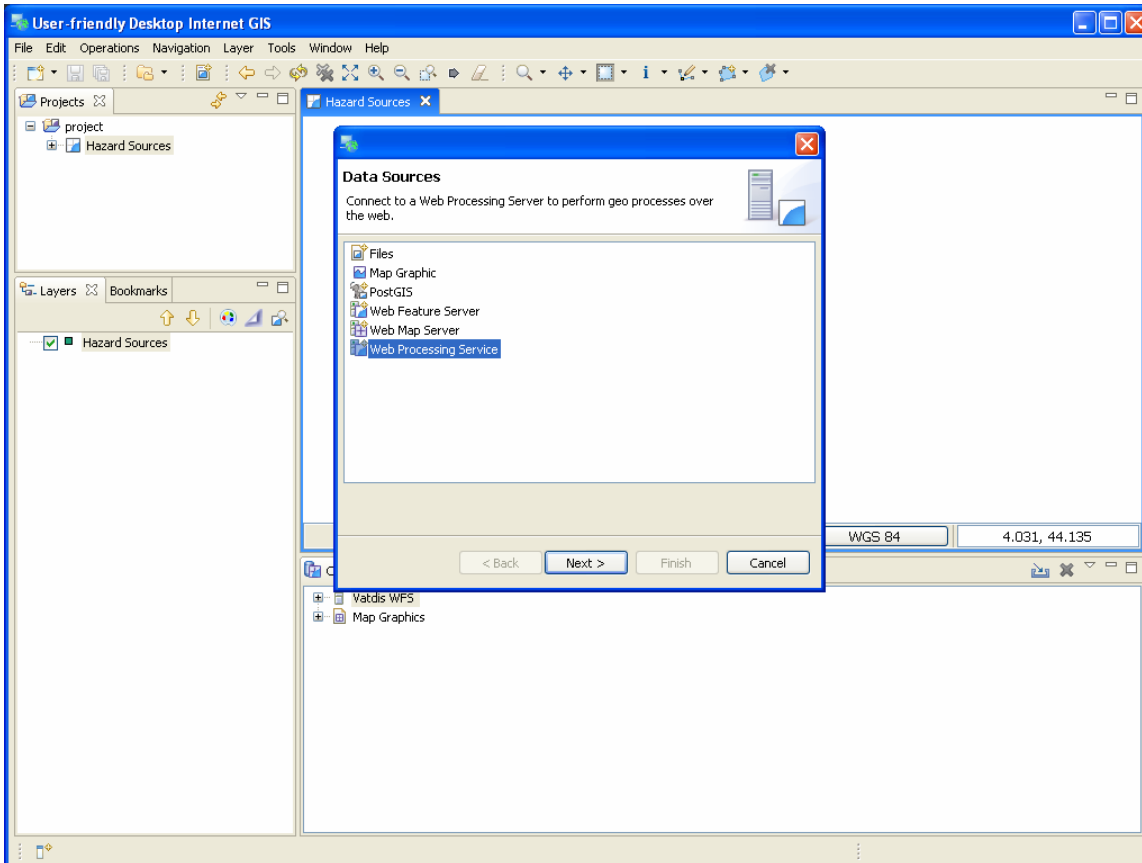
Step 1c: select a point layer from the list in the menu.

After that the layer is displayed on the screen:

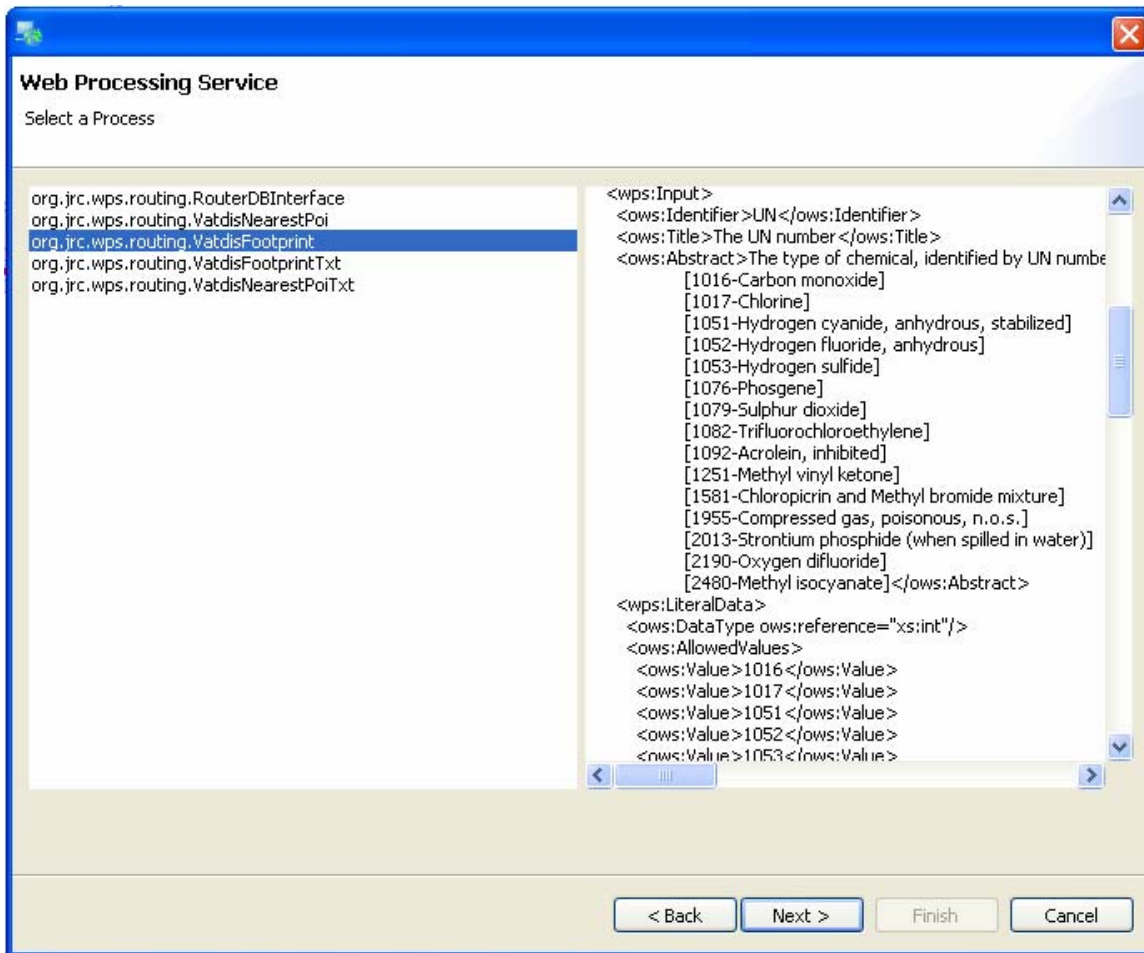


Step 1d: display of layer on screen

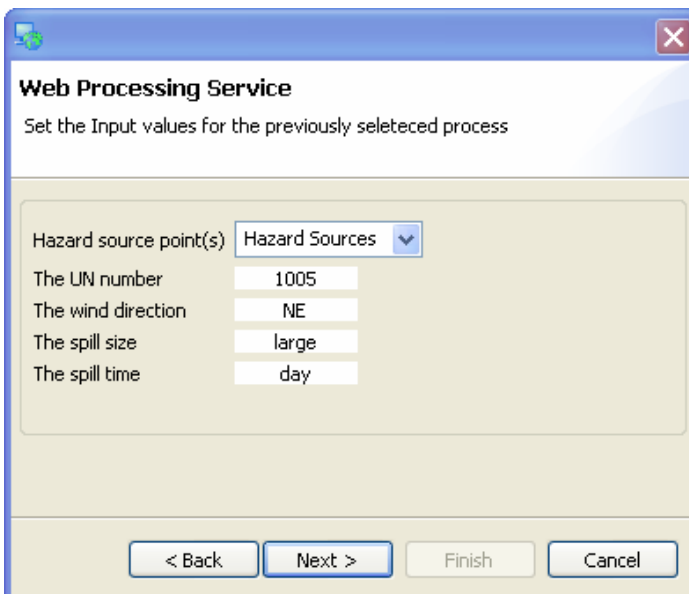
Step two: the user then selects WPS Footprint process from Vatdis server



Step 2a: select web processing service to add to the map, and type in the address of the service.

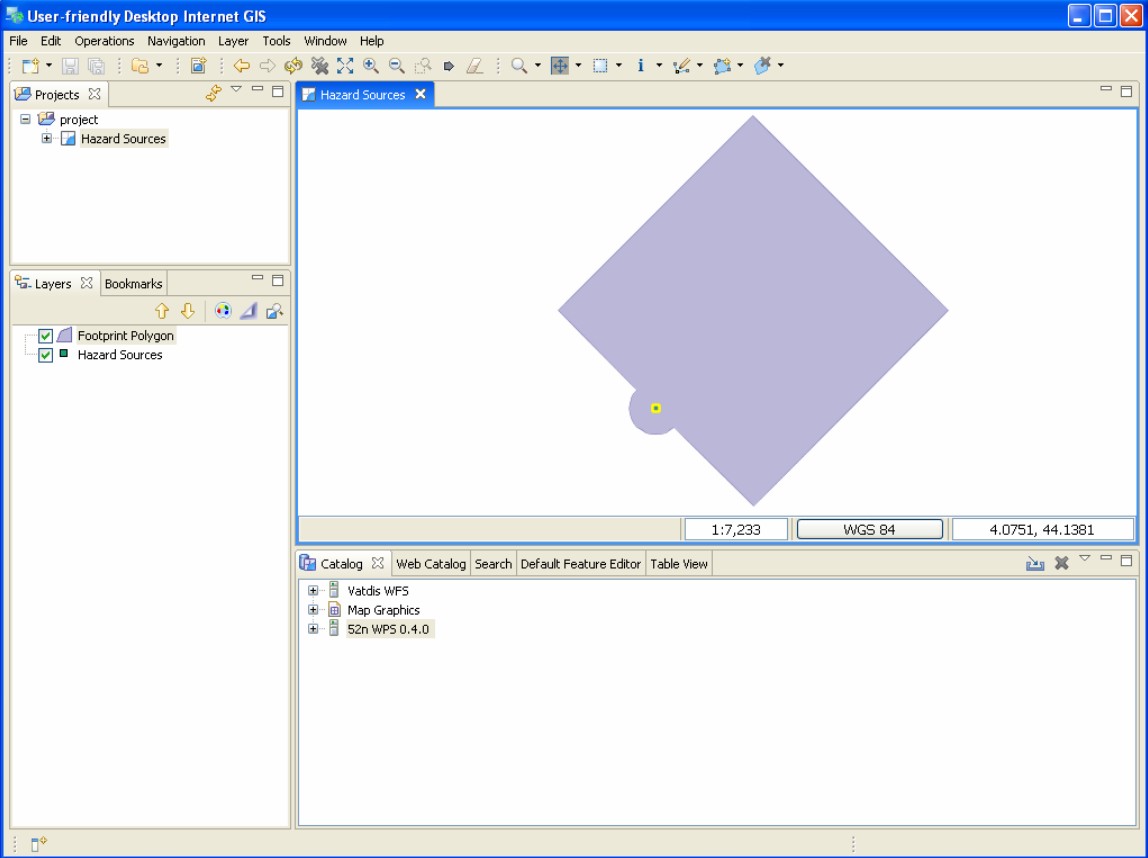


The user then types in the input values (source term)



Step 3a: type
in the
description of
the source term

Final step the result is displayed on the screen.



Annex 7: VATDIS and SOAP

The following point is a so called service endpoint. This is where the user can direct his or her client to in order to start interacting with the Processing service.

<http://vatdis.jrc.it:8080/axis2/services/ProcessingService>

Message send:

In more technical terms, the following is the message that is send to the service. The important elements of this message are displayed in blue., This is the section where the wps gets asked to calculate for a given LATITUDE and LONGITUDE, the footprint of a chemical leak given a certain UN number and a certain wind direction (WIND)

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ps="http://vatdis.jrc.it:8080/axis2/services/ps"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:wps="http://www.opengeospatial.net/wps"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink" >
  <SOAP-ENV:Body>
    <wps:Execute service="WPS" version="0.4.0" store="false" status="false"
xmlns:wps="http://www.opengeospatial.net/wps"
xmlns:ows="http://www.opengeospatial.net/ows" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengeospatial.net/wps..wpsExecute.xsd">
      <ows:Identifier>org.jrc.wps.routing.VatdisFootprint</ows:Identifier>
      <wps:DataInputs>
        <wps:Input>
          <ows:Identifier>LATITUDE</ows:Identifier>
          <ows:Title>Point latitude</ows:Title>
          <wps:LiteralValue dataType="xs:string">44</wps:LiteralValue>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>LONGITUDE</ows:Identifier>
          <ows:Title>Point longitude</ows:Title>
          <wps:LiteralValue dataType="xs:string">4</wps:LiteralValue>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>UN</ows:Identifier>
          <ows:Title>The UN number</ows:Title>
          <wps:LiteralValue dataType="xs:string">1005</wps:LiteralValue>
      </wps:Execute>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

```

    </wps:Input>
    <wps:Input>
      <ows:Identifier>WIND</ows:Identifier>
      <ows:Title>The wind direction</ows:Title>
      <wps:LiteralValue dataType="xs:string">NE</wps:LiteralValue>
    </wps:Input>
  </wps>DataInputs>
</wps:Execute>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Return message

The return message contains a description of the message that was send and contains a combination of coordinates that together make the polygon of the footprint (displayed in red)

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body><wps:Execute
eResponse xmlns:wps="http://www.opengeospatial.net/wps"><ows:Identifier
xmlns:ows="http://www.opengeospatial.net/ows" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">org.jrc.wps.routing.VatdisFootprint</ows:Identifier><wps:Status><wps:ProcessAcc
epted>This process was executed at: Nov 28,
2007</wps:ProcessAccepted></wps:Status><wps>DataInputs
xmlns:ows="http://www.opengeospatial.net/ows" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <wps:Input>
    <ows:Identifier>LATITUDE</ows:Identifier>
    <ows:Title>Point latitude</ows:Title>
    <wps:LiteralValue dataType="xs:string">44</wps:LiteralValue>
  </wps:Input>
  <wps:Input>
    <ows:Identifier>LONGITUDE</ows:Identifier>
    <ows:Title>Point longitude</ows:Title>
    <wps:LiteralValue dataType="xs:string">4</wps:LiteralValue>
  </wps:Input>
  <wps:Input>
    <ows:Identifier>UN</ows:Identifier>
    <ows:Title>The UN number</ows:Title>
    <wps:LiteralValue dataType="xs:string">1005</wps:LiteralValue>
  </wps:Input>
  <wps:Input>
    <ows:Identifier>WIND</ows:Identifier>
    <ows:Title>The wind direction</ows:Title>
    <wps:LiteralValue dataType="xs:string">NE</wps:LiteralValue>
  </wps:Input>
</wps>DataInputs><wps:ProcessOutputs><wps:Output><ows:Identifier
xmlns:ows="http://www.opengeospatial.net/ows">RESULT</ows:Identifier><wps:LiteralValue
>&lt;gml:featureMember&gt;

```

<gml:MultiPolygon
srsName="EPSG:4326"><gml:Polygon><gml:OuterBoundaryIs>4.093611240386
96,44.1416298344988 4.09361124038696,44.1462473869324
4.10182428359985,44.1462473869324 4.10182428359985,44.1416296365941
4.10227392389846,44.1410817483052 4.10278030688164,44.1401343723781
4.1030921359831,44.1391064095945 4.10319742778535,44.1380373639878
4.10319742778535,44.1380373639868 4.10309213598329,44.13696831838
4.10278030688202,44.1359403555964 4.10227392389901,44.1349929796692
4.10159244704833,44.134162597724 4.10076206510316,44.1334811208733
4.09981468917597,44.1329747378902 4.09878672639236,44.1326629087889
4.0977176807856,44.1325576169868 4.09664863517883,44.1326629087888
4.09562067239518,44.13297473789 4.09467329646795,44.133481120873
4.09384291452273,44.1341625977237 4.09316143767197,44.1349929796688
4.09265505468888,44.135940355596 4.09234322558751,44.1369683183796
4.09223793378535,44.1380373639863 4.09234322558732,44.1391064095931
4.0926550546885,44.1401343723767 4.09316143767142,44.141081748304
4.09361124038696,44.1416298344988</gml:OuterBoundaryIs></gml:Polygon></gml:
MultiPolygon></gml:featureMember></wps:LiteralValue></wps:Output></wps:ProcessOutputs></wps:ExecuteResponse></soape
nv:Body></soapenv:Envelope>

Annex 8: Setting up Joseki

One demand is put on extending the application with semantics and that is that the supporting tools must be installed under the same tomcat servlet container. This because of one severe restriction caused by using AJAX techniques: requests from the client can only be sent to the same server as where the client is hosted. In essence: an AJAX request send from a client hosted at <http://vatdis.jrc.it:8080> can only be send to applications running at the same address. Having this functionality available is essential because it is part of the techniques used for Mapbuilder/Geoserver.

There are only a limited number of tools that meet these requirements. Of these, this study continues with Joseki. It is relatively easy to install, works under tomcat, is open source and is well accepted in the community. On top of that, it is extendable with a free open source reasoning machine (pellet).

Installing under tomcat is however not straightforward. Several sources aim at the possibility of doing this²⁷. However each of them is incomplete and among each other they are inconsistent. The technical details in this annex should provide the user a guide to install it²⁸.

First the version available at www.joseki.org is not a complete version of the application. It is advised to get the latest version of sourceforge and build it in eclipse.

In the build version, the folder joseki under webapps should be used for putting in the tomcat servlet container. In addition to that the web.xml fiel under web-inf in webapps points to the classes and lib directory. These should be copied from the joseki built version to the web-inf directory under the joseki webapps version (web.xml already points to it).

Getting the service to run with actual data is also not completely documented. Whitin the existing documents the essential steps are left out.

The following steps have proven to work .

1. The data folder under joseki-built should also be brought under the joseki web application folder.
2. A rdf/owl file (constructed using an editor like protégé) should be converted to a n3 file²⁹ e.g. vulnerability.n3
3. This file should be configured under the joseki configuration file and under the web application web.xml

The joseki configuration continues with two important files. First there is the configuration of the web application as such under tomcat, this is done using the web.xml then there is the configuration of data under joseki which is done with the joseki-config.ttl.

Registering data under joseki involves tow steps: registering the data and registering the service involving the data. Suppose data is stored in n3 format under a file name "vulnerability.n3" which is tored in a directory called Data under the web application.

²⁷ <http://www.joseki.org/webapp.html> and <http://imageweb.zoo.ox.ac.uk/wiki/index.php/DefiningImageAccess/Tool/Joseki>

²⁸ Using Joseki 3.1 from svn, and tomcat 5.x or higher.

²⁹ Using for example <http://semanticmoby.org/developer/format-converter.jsp>

<<Register service under congif.ttl and web.xml>>

This dataset then need to be referred to in the joseki-config.ttl as following:

```
_:vulnerability rdf:type ja:RDFDataset ;
  rdfs:label "Vulnerability" ;
  # Either - or
  ja:defaultGraph
    [ rdfs:label "vulnerability.n3" ;
      a ja:MemoryModel ;
      ja:content [ja:externalContent <http://localhost:8080/joseki/Data/vulnerability.n3> ] ;
    ] ;
.
```

Of which the last sentence need to be updated according to the exact name of your host an application.

The service itself needs to be registered in the same file as well, using the following lines of code:

```
# Service 4 - SPARQL processor only handling a given dataset
[]
  rdf:type      joseki:Service ;
  rdfs:label    "SPARQL on the books model" ;
  joseki:serviceRef "vulnerability" ;
  # dataset part
  joseki:dataset  _:vulnerability ;
  # Service part.
  # This processor will not allow either the protocol,
  # nor the query, to specify the dataset.
  joseki:processor  joseki:ProcessorSPARQL_FixedDS ;
.
```

After that he web.xml under web-inf need to be brought in-line with the joseki-config.ttl. this involves updating the servlet mapping part of the web.xml with the following additional lines of code:

```
<servlet-mapping>
<servlet-name>SPARQL service processor</servlet-name>
<url-pattern>/vulnerability</url-pattern>
</servlet-mapping>
```

These lines can be copied and dragged to that section of the file.

<<Execute query>>

After that the service can be tested by using the endpoint in a sparql query.

```
PREFIX : <http://example.org/book/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX assert: <http://www.owl-ontologies.com/assert.owl#>
PREFIX vulnerability: <http://localhost:8080/joseki/vulnerability.owl#>

SELECT ?explosive_substances ?IsolateSmallv
FROM <http://localhost:8080/joseki/vulnerability.owl>
WHERE
{ ?explosive_substances vulnerability:hasIsolateSmall ?IsolateSmall .}
```


European Commission

EUR 23071 EN – Joint Research Centre – Institute for the Protection and Security of the Citizen

Title: VATDIS Web Mapping - A Report on the Application of Open Standards and Open Architecture in Geospatial Interoperability for Emergency Management

Author(s): C. Logtmeijer, P. Isoardi, B. Schupp, J P Nordvik.

Luxembourg: Office for Official Publications of the European Communities

2007 – 84 pp. – 21 x 29.7 cm

EUR – Scientific and Technical Research series – ISSN 1018-5593

ISBN 978-92-79-08110-1

DOI 10.2788/6125

This report provides an overview of the web mapping activities carried out in the VATDIS action in 2007. These web mapping activities aimed at integrating the work done for the Orchestra, Floodsite and Preview IP into one single demonstrator.

In the main section of this report an outline is given of the reasons for integrating the projects and the contents of this integration. The report should be seen as a summary of the technical choices made in order to accomplish this integration. The annexes on this report help the more technical reader to understand the design of the demonstrator and to duplicate it for its own purposes.

How to obtain EU publications

Our priced publications are available from EU Bookshop (<http://bookshop.europa.eu>), where you can place an order with the sales agent of your choice.

The Publications Office has a worldwide network of sales agents. You can obtain their contact details by sending a fax to (352) 29 29-42758.

The mission of the JRC is to provide customer-driven scientific and technical support for the conception, development, implementation and monitoring of EU policies. As a service of the European Commission, the JRC functions as a reference centre of science and technology for the Union. Close to the policy-making process, it serves the common interest of the Member States, while being independent of special interests, whether private or national.

LB-NA-23071-EN-C



ISBN 978-92-79-08110-1

