RIMapper

A test bed for online Risk Indicator Maps using data-driven SVG visualisation

Barend Köbben <kobben@itc.nl>
Department of Geo-Information Processing
International Institute for Geo-information Sciences and Earth Observation (ITC)
PO box 6, 7500 AA Enschede, The Netherlands
http://kartoweb.itc.nl/RIMapper/

Abstract

This paper presents results of a test bed application called *RIMapper*, that was used to look into the possibilities of generating Risk Indicator Maps (RIMs) from online databases, using Java server technology (JSP) to generate light-weight, versatile maps in the Scalable Vector Graphics (SVG) format. These maps are to be part of an urban risk management system, and therefore need to fit a multitude of use cases, ranging from providing the general public with information about risks to providing local authorities an interface to the underlying risk assessment databases and models. Furthermore, the maps need to be usable on a wide range of platforms, from the office systems of the local authorities to hand-held devices providing location based services to field personnel. This paper will present results of the second phase of this test bed project, in which the central issue was the online generation of the RIMs by a web application from a spatial database back end, implemented using Open Source standards and software: OpenGIS Simple Features stored in mySQL Spatial Extension, extracted by a JSP Tomcat server application and delivered in SVG format to web clients.

Keywords

Java, JSP, OpenGIS Simple Features, Open Source, mySQL Spatial Extension, Risk Indicator Maps, SVG, web clients

Introduction - the SLARIM project

The test bed this paper discusses is part of an internal research project at the International Institute for Geo-information Science and Earth Observation (ITC), called “Strengthening Local Authorities in Risk Management” (SLARIM). The objective of this project is the development of a methodology for the implementation of risk assessment and spatial decision support systems for risk management by local authorities in flood and earthquake threatened urban areas in developing countries [1]. Risk information, presented spatially, is needed by local authorities to take decisions on how to reduce the risk for particular areas, either by reducing the hazard probability (eg. structural measures like dikes) or by reducing the vulnerability (eg. restrictive zoning, building codes). Risk information also forms the basis for a proper emergency response planning. The project focuses on the two types of natural hazards that cause most damage in urban areas: flooding and earthquakes. The methodology will be developed in a number of case study cities in different countries. The case study cities are at the moment: Kathmandu (Nepal), with a focus on earthquake and flood risks, Naga City (Philippines) for floods and Dehra Dun (India) for earthquakes.

The project is structured as a series of work packages, dealing with analysis of the institutional setting; user needs assessment, evaluation of the spatial data infrastructure, hazard assessment, and generation of databases of elements at risk, vulnerability assessment and risk assessment. One of these work packages deals with the spatial data infrastructure and base data aspects of a flood/seismic risk assessment system, directed towards the needs of medium sized local authorities in developing countries. The activities within this work package include the development of a methodology to handle interoperability aspects. Distributed processing, interoperability research and the impact of the OpenGIS specification will play an important part in this activity. The author of this paper is responsible for "recommendations concerning the methods for the online mapping and presentation of risk indicators".

For this work, visualisation will be considered as part of a system of interoperable web services. The reason for this is that such a system could support online, distributed data from various local authorities, and it can be scalable (ranging from in-office planning to rapid response in the field) and time-aware (supporting comparisons, real-time views and extrapolation into future scenarios). All these requirements, plus the intention to comply to open standards and use open source, have led to the decision to use XML-based solutions in this work package, and thus GML for spatial data exchange and SVG for visualisation.

The RIMapper test bed - a lightweight GDI?

It was decided to try out the concepts developed in the work package in a test bed application. The implementation of this test bed has been started in March 2003. This first phase concentrated on the cartographic design of maps that visualise real and perceived risks and some preliminary tests to realise these in SVG format suited for multiple platforms. The results of this first phase were presented in July 2003 at the 2nd Annual Conference on Scalable Vector Graphics [2], and the output can be seen at the RIMapper website [3].

This paper presents results of the second phase, in which the central issue was the online generation of the RIMs by a web application from a spatial database back end. This focus can be seen outlined in the conceptual set-up for the overall test bed in figure 1. The main building blocks of this system are:

- A spatial database back end that stores the geometry and the attribute data; For this part both PostGIS (a spatial data extension to PostgreSQL) and mySQL were under consideration, finally the choice has been made to use mySQL. From the onset it was envisaged that spatial data should be stored using the OpenGIS (OGIS) Simple Features specifications.
- A (set of) web application(s) that extract data from the database and delivers it in SVG for visualisation purposes and other XML formats (such as GML) for data exchange. In first instance this will be done real-time, although it is foreseen that performance issues might require various types of pre-processing in a later stage. These are developed using the open source Java server technology.
- A web-based user interface enabling access to the maps and data for both desktop browsers as well as mobile platforms (PDA's). The original data providers initially will have data access through this same web-based interface. In a later stage however, they should also have more sophisticated clients, for example by providing the data through an OpenGIS Web Feature Server to GIS clients.

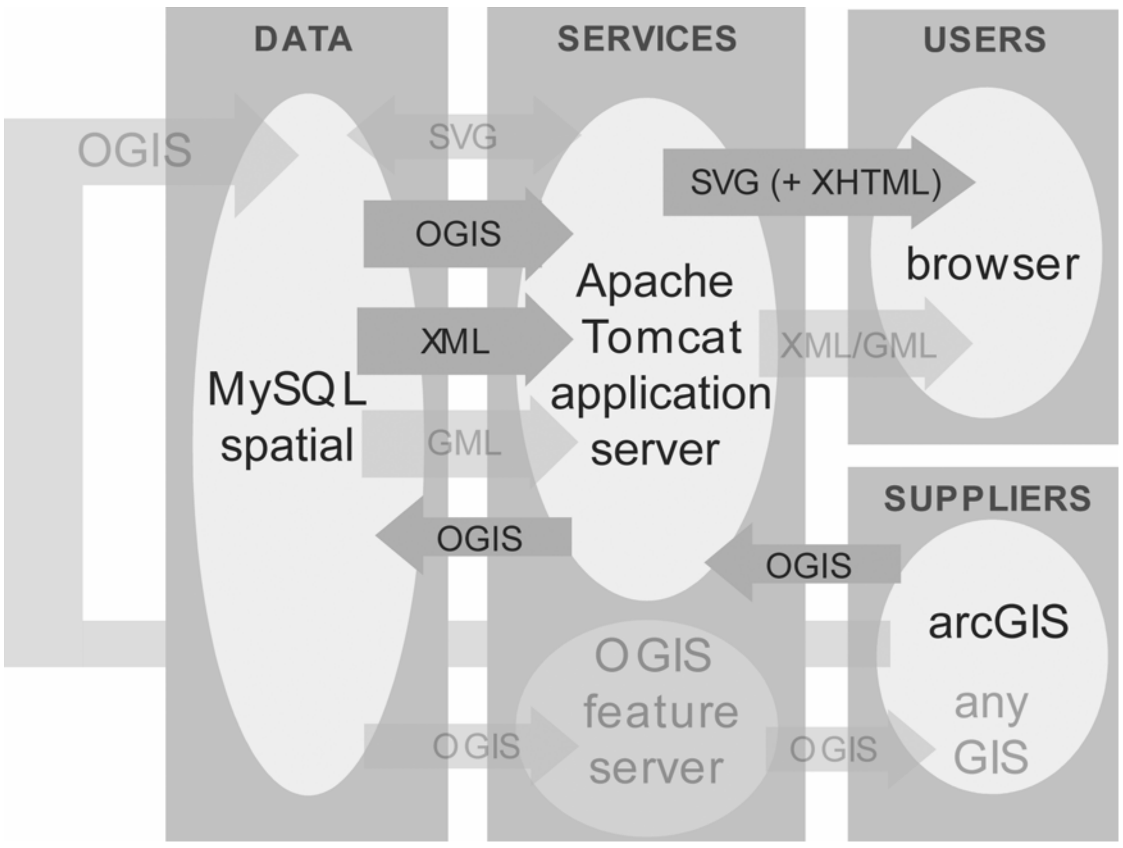


Fig. 1: Conceptual set-up of RIMapper test bed.
Faded elements are not part of the current implementation;
OGIS stands for OpenGIS Simple Features.

Using the setup described, the RIMapper test bed would provide the local authorities that the SLARIM project addresses with a relatively simple, low-cost, yet powerful way of sharing data amongst various distributed offices and institutions as well as the general public. As such, the system could be considered the technological heart of a lightweight Geospatial Data Infrastructure (or GDI). The term GDI might be more usually connected with (very) large regional or national spatial data warehouses, but it is defined more generally (in [4]) as "the networked geospatial databases and data handling facilities, the complex of institutional, organizational, technological, human and economic resources (…) facilitating the sharing, access to, and responsible use of geospatial data at an affordable cost for a specific application domain or enterprise". For the test bed to rightfully earn the title "light-weight GDI", the most obvious addition needed would be a clearing house component [4, ch.9.5]. Although this is not being pursued in the current test bed, such a component could be part of the larger SLARIM project, and would be relatively simple to add, using existing standards and software solutions (see for instance [4], [5] and [6]).

In the following paragraphs, the components of the test bed application will be described. The emphasis will be on the on the technological solutions chosen and the reasons for these choices. As this is very much a work in progress, there is as of yet not much insight into the effectiveness and performance of the system and there is no "real user" experience at all. These questions will have to be addressed at a later stage in the project.

Database backend - mySQL spatial extension

If one needs a database to serve as the backend of a geospatial service such as RIMapper, the most obvious choice is a so-called *spatially enabled* database, that has special data types, column types, functions and operators to deal with geometries. Such *spatial extensions* or *spatial cartridges* as they are sometimes called, provide access to the geometry through a standardised interface (SQL). Other solutions are possible, such as storing the data in arbitrary binary objects (BLOBs) and providing the geometry "intelligence" in the application tier instead of in the database itself. But such solutions are non-standardised and often proprietary. As the RIMapper project set out to comply to open standards, there was the need to use a database that supported the OpenGIS Consortium (OGC) standards. The OGC is a member-driven, non-profit international trade association that is leading the development of geoprocessing interoperability computing standards [7]. Among (many) other things, the OGC has set the Simple Features SQL Specification that provides for publishing, storage, access, and simple operations on Simple Features (point, line, polygon, multi-point, etcetera) through an SQL interface.

There are several databases with OGC-compliant spatial extensions, of which *Oracle* is probably the most prominent, but as another goal was to use open source software, only two databases were under consideration: *Postgre*SQL is a database system with *Post*GIS as an extension, that supports OGC spatial geometry, spatial queries and simple analysis [8]. It is available for many platforms, however currently not in a native Windows version. The *my*SQL database server is available for free under the GNU General Public License (besides being sold under a commercial license) for all major platforms including Windows. MySQL claims to be the most popular open source database server in the world [9]. The most recent version (4.1.0) includes OGC-compliant spatial extensions, but it is currently in the alpha stage, thus not a fully reliable production release.

Both do not adhere to the full set of OGC specifications, the most obvious being the lack of Spatial Reference Identifiers (SRID) support in mySQL. Its geometry tables do, however, already include an SRID field, making it easy to prepare for the projection support in a later version. The reasons for the choice of mySQL for the RIMapper project were the native Windows support and the simple "lightweight" character of the software as compared to the complicated though more fully-featured PostgreSQL. Furthermore, as far as we know, at the time of writing there is no publicly available work using mySQL as a database backend for (SVG) mapping, whereas this is the case for PostGIS (eg. using Perl, as described in [10]). By adhering strictly to the OGC standards it should be straightforward anyway to change or even mix database platforms in the future.

The database setup of the current RIMapper system, as realised in mySQL, can be seen in figure 2. The "features" table is the central place where all geometric objects available to the application are stored. Initially and most importantly, the features are stored as OGC Simple Features geometry and transformed to other formats at run-time, when needed by the application tier. At a later stage, for performance reasons a strategy might be adopted with various types of pre-processing, where for instance database triggers make sure SVG and GML expressions of the geometry are (re)calculated whenever a change in the OGC geometry takes place.

The non-spatial attributes of the features are stored per category (or layer) in specific tables ("homes", "roads", etcetera), related with the "features" table through their ID. A "layers" table is provided as a per layer link to the "styles" table, for layers that should be styled uniformly (eg. all roads sharing the same visualisation). Whenever the visualisation should depend on some data attribute per feature (eg. for a chorochromatic map of homes viewed by vulnerability type), the link is made from the data specific attribute table directly to the "styles" table. The choice for the visualisation type mentioned above is directed by the XML map description file (stored in the table “MAPS”), that is processed by the application tier (see next paragraph). A further two tables, “fragments” and “actions” are also for use by the application tier, storing SVG code fragments and ECMAscript event listeners, respectively.

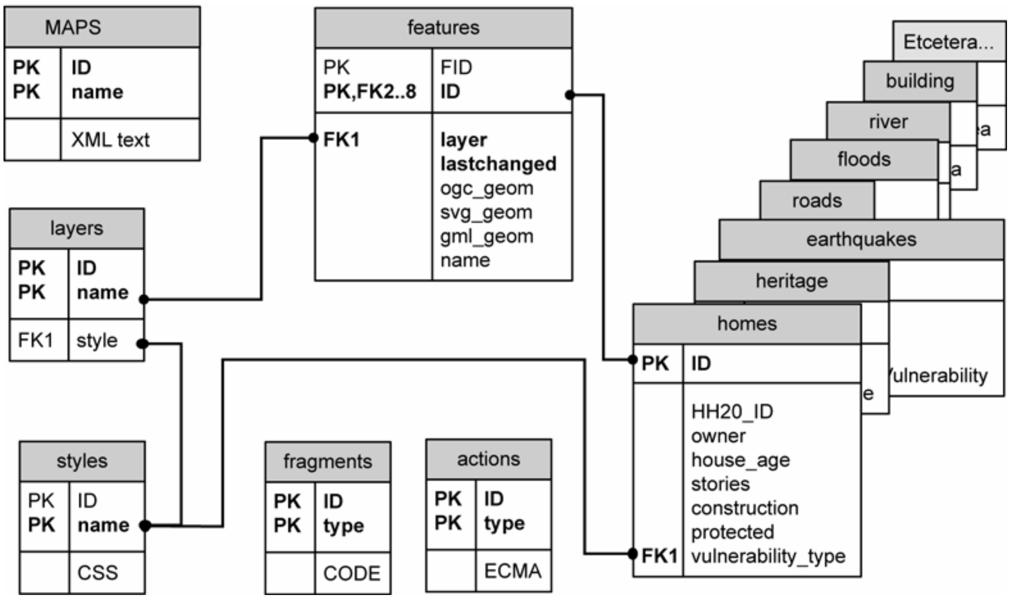


Fig. 2: Setup of a the current RIMapper database
(dot-terminated lines indicate relations specified in the database)

Services - Tomcat Java application server

The heart of the RIMapper system are the services provided by a set of Java Servlets and Java Server Pages. In our case, the application tier runs on Tomcat, a well-known open source servlet container from the Apache Software Foundation [11], but the applications should run on any standards-compliant servlet container.

As described earlier and seen in figure 1, the functionality at the moment is focussed on providing end users with RIMs in the form of SVG-formatted files. Providing SVG files from a Java Server Page is very straightforward: One basically takes the desired SVG output and adds little bits of Java code to it to provide data-driven visualisations. Take for example the SVG fragment:

```
<circle id=”12” cx=”100” cy=”100” r=”
<%if (someData == true) {%>
   10
<%} else {%>
   20
<% } %>
” />
```

The bits between the `<% %>` tags will be processed by the Tomcat engine before being delivered to the client and therefore this will result in line 02 being in the output if some data value (eg. coming from a database) is true, and line 04 if false, resulting in a circle with either radius 10 or 20. This approach is fine if one aims at delivering maps that are largely similar for all cases and clients, and only have small differences in visualisation. For our purposes it is, however, not generic nor flexible enough, more so because it is foreseen that in time the applications should also enable output in other formats.

The RIMapper system therefore uses a set of much more generic Java classes to do recurring tasks like extracting OGC features and attribute data from the database, translating these into fragments of SVG and ECMAscript, collecting and structuring these fragments into valid output and delivering this output to the clients.

The glue provided to make all these parts act together are the XML *map definitions*. These are basically XML-formatted text files, but like all data for the system they are stored in database tables. They are parsed to get a description of the map needed and all its component parts. Take for example the XML map description below:

```
00 <?xml version="1.0" encoding="iso-8859-1"?>
01 <RIM ID=”999” TYPE=”SVG_STANDALONE” DB=”RIMapper_DB” UN=”user” PW=”pass”>
02   <TITLE>A Risk Indicator Map</TITLE>
03   <AUTHOR>Somebody</AUTHOR>
04   <HEADER>
05     <FRAGMENT ID=”001” NAME=”standardRoot” TYPE=”SVG_ROOT” />
06     <FRAGMENT ID=”143” NAME=”opacityGradient” TYPE=”SVG_GRADIENT” />
07     <FRAGMENT ID=”14” NAME=”roadMarker” TYPE=”SVG_MARKER” />
08     <FRAGMENT ID=”56” NAME=”citySymbol” TYPE=”SVG_SYMBOL” />
09     <FRAGMENT ID=”33” NAME=”dropShadow” TYPE=”SVG_FILTER” />
10     <STYLES>
11       <STYLE ID=”100” NAME=”defaultText” TYPE=”CSS” />
12       <STYLE ID=”101” NAME=”defaultPoint” TYPE=”CSS” />
13       <STYLE ID=”102” NAME= defaultLine” TYPE=”CSS” />
14       <STYLE ID=”103” NAME=”defaultArea” TYPE=”CSS” />
15     </STYLES>
16     <FRAGMENT ID=”534” NAME=”” TYPE=”ECMASCRIPT” />
17   </HEADER>
18   <BODY>
19     <LAYER ID=”1098” NAME=”floods” TYPE=”single” PARAMS=”none”>
20       <ACTION ID=”12” NAME=”changeSymbol” TYPE=”feature” EVENT=”onmouseover”
         PARAMS=”flood_type” />
21     </LAYER>
22     <LAYER ID=”354” NAME=”houses” TYPE=”chorochromatic” PARAMS=”vulnerability_type” >
23       <ACTION ID=”12” NAME=”showLayer” TYPE=”layer” EVENT=”onclick” PARAMS=”null” />
24     </LAYER>
25   </BODY>
26   <FOOTER>
27     <FRAGMENT ID=”002” TYPE=”SVG_FOOTER” />
28   </FOOTER>
29 </RIM>
```

Line 01 indicates the connection parameters for the database and the type of RIM output that will be generated. For this phase of the test bed, the only types supported will be “SVG_STANDALONE” and “SVG_EMBEDDED”.

The former will result in an SVG map with all style information, scripting and data incorporated in the one file. The SVG generated will adhere to the SVG-Basic profile and this type of map will be suited for the broadest range of clients, including PDA’s. The user interface is limited to standard SVG viewer capabilities such as zooming, panning and querying data by clicking or moving the pointer to mapped elements.

With the latter type, the result will be an XHTML file in which one or more SVG's, formatted according to the SVG-Full profile, are embedded. Although this limits the possible user platforms (see next paragraph), it offers a fully-featured user interface with additional possibilities, such as downloading geometry and attribute data, queries by data attributes or layer, printing, and brushing through time-series. The SVG_EMBEDDED type is still under development at the time of writing.

In the HEADER section several fragments of SVG code are loaded to define gradients, symbols, markers and filters. It is important to point out that the XML definition is not something that is necessarily processed by the RIMapper applications line-by-line in this order. This is for example important for the first fragment (line 05), that defines the SVG_ROOT, ie. the ECMAscript to add event listeners to the DOM tree, and the size and the viewbox of the map. For this last item, the system needs to know the spatial extent of the data, and thus it will be defined only after processing all geometric information in the layers. The same is true for the styles item, because the list of CSS styles to be loaded from the database as seen here will be expanded with further styles needed, as found in the layer definitions.

The last part, the FOOTER section, simply declares the closing part of the output.

In between is the BODY section that lists the actual layers of information to be mapped. Per layer the NAME is used to retrieve the attributes from the table with that same name and the geometry from the "features" table. The TYPE indicates how the layer should be visualised. With some types, such as "single" in line 19, one style will be used to visualise all features in this layer in the same way, and the link to that style will come from the "layers" tables in the database. With others, such as "chorochromatic" in line 22, the style needed will be determined based on the value found in the database attribute stated in PARAMS. Layers can have one or more ACTIONS that define the interactivity. Here the TYPE determines if the action uses the same parameters for the whole layer (type "layer" in line 23), which will result in an ECMAscript eventlistener being attached to the enclosing SVG group, or if it needs different parameters (PARAMS) per feature depending on some attribute (eg. the "flood_type" attribute in line 20). The latter will result in an eventlistener being attached to all features in the layer separately. Lastly, the EVENT item determines which event will trigger the action.

Several strategies were considered to extract the OpenGIS geometry from the "features" table. One possible solution, chosen for example in [10], is to make use of the Well-Know Text (WKT) expression of the geometry. This WKT is designed as the OpenGIS standardised way to exchange geometry data in ASCII form. After issuing an SQL SELECT followed by the appropriate conversion function to retrieve the geometry of the feature as WKT, one can then parse the resulting (usually very long) string to extract the geometry type, the component points, lines and polygons and their co-ordinates. This method is very straightforward, but the resulting code can be very confusing and the parse process has to be different for every geometry type.

For RIMapper, a more database-centric and transparent method was chosen, making full use of the spatially enabled SQL functions and without using the WKT format. As can be seen in the pseudo-code below, a first query is done to find out the geometry type of the features (= the table rows) for a particular layer, and the number of geometries in the feature. This last item is needed because many OGC geometry types are complex, such as the MULTILINESTRING geometry that is a collection of LINESTRINGs. For every geometry type the feature is further dissected into its component parts by nested loops of SQL queries. In the code shown this is worked out for the MULTILINESTRING case only, where one can see that for every geometry found, a second query finds the number of points in that geometry and for all these points the X and Y co-ordinates are found by a third loop of queries. The part in line 13 that stores the feature data is done by another Java class, dependant on the type of output needed.

```
SQL Query 1 = SELECT FID, ID, GeometryType(ogc_geom), numGeometries(ogc_geom),
FROM FEATURES WHERE layer=[LAYERNAME];
while RecsFound {
   if OGC_geometry Type = POINT {
     […deal with POINT…] }
   else if OGC_geometry Type = MULTIPOINT {
     […deal with MULTIPOINT…] }
   else if OGC_geometry Type = LINESTRING {
     […deal with LINESTRING…] }
   else if OGC_geometry Type = MULTILINESTRING {
     for (j=1; j<= numGeometries; j++) { //loop through NumGeoms
        SQL Query 2 = SELECT NumPoints(GeometryN(ogc_geom,[j]))
        FROM features WHERE FID=[FID];
        for (i=1; i<=NumPoints; i++) { //loop through NumPoints in this Geometry
           SQL Query 3 = X(PointN(GeometryN(ogc_geom,[j]),[i])),
           Y(PointN(GeometryN(ogc_geom,[j]),[i])) FROM features WHERE FID=[FID];
           […store X and Y coordinates and other data for the Feature…]
           } } }
   else if OGC_geometry Type = POLYGON {
     […deal with POLYGON…] }
   else if OGC_geometry Type = MULTIPOLYGON {
     […deal with MULTIPOLYGON …] }
   else {
     [Unknown Geometrytype: Error] }
}
```

This setup provides a very flexible and standardised way to extract the geometry, but compared with the WKT-parsing method it uses many more database queries. Because mySQL is known to be one of the fastest database engines, and the current test data set is not very big (roughly 2000 features, ranging from single points to multipolygons with hundreds of points), the currently the system performs satisfactory, but it's clear that serious performance tests should be conducted at a later stage.

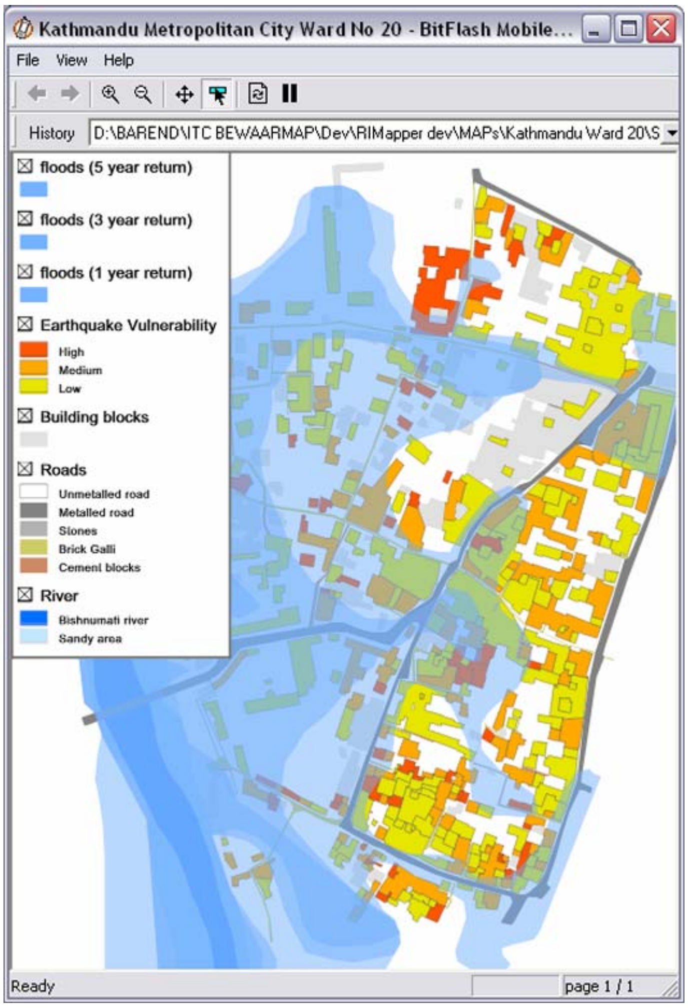
When all data needed has been collected by the system, the output is composed and handed over to the web server (that role is also played by Tomcat) for delivering to the client.

Clients - SVG enabled web browsers and PDA's

As the output maps are to be part of an urban risk management system, they need to fit a multitude of use cases, ranging from providing the general public with information about risks, to providing local authorities an interface to the underlying risk assessment databases and models. Furthermore, the maps need to be usable on a wide range of platforms, from the office systems of the local authorities to hand-held devices providing location based services to field personnel.

As explained earlier, in the XML map definition the type of RIM will determine if the map is to be a standalone SVG file or embedded in an XHTML page. The reasons for this lies in the current viewer situation: only standalone files, with all their styling and scripting incorporated locally, can be constructed in such a way that they'll function in almost all SVG viewers in a similar fashion. For these standalone files, the choice was also made to use not the full SVG feature set, but the SVG-basic profile. They will therefore also run in more lightweight viewers, such as the BitFlash mobile viewer, shown in figure 3 in its WindowsXP incarnation but also available for PDA's running on WindowsCE, Symbian and some other mobile platform operating systems.

The SVG_EMBEDDED type under development will have to allow for the different ways that browsers have implemented plug-in embedding and more importantly the inter-document communication to allow scripts to work across XHTML and SVG. It is therefore foreseen that various versions for specific combinations will be needed, eg. for Internet Explorer + Adobe SVG viewer or for Mozilla Firebird + Corel SVG viewer.



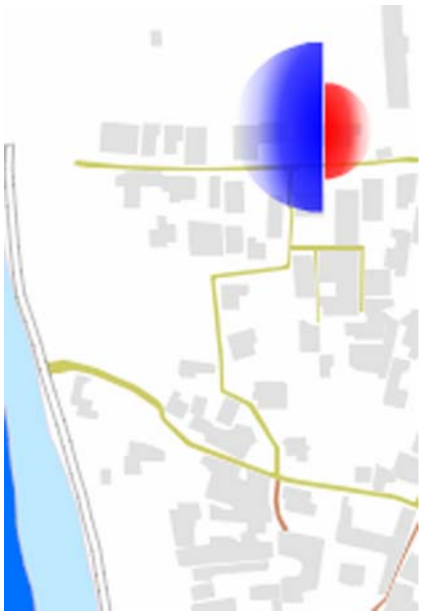


fig. 3 (above): Example of a RIM *dynamic risk symbol showing the modelled risks (blue for floods, red for earthquakes) at the current pointer location.*

fig. 4 (left): Example of a RIM *screen dump from the BitFlash SVG mobile viewer*

The actual cartographic design and user interfaces are not part of the RIMapper test bed described in this paper. It is however an important part of the SLARIM project as the work package "Visualization and Use of Risk Indicator Maps". This work package aims at making recommendations concerning the methods for the online mapping and presentation of risk indicators suitable for the use in assessment procedures by the public, commercial and government agency planning activities. It encompasses among other things the analysis of geospatial risk data coming from other work packages and research into the cartographic grammar for visualisation of real and perceived risks.

Data suppliers - shp2mysql loader

At the time of writing, there has been only limited attention given to the 'suppliers' part (as seen in figure 1) of the test bed. The focus has been on providing a way to easily and quickly get useful test data into the database. Most of the data used in the SLARIM project is either available directly in the ESRI ArcGIS environment or can be imported into it easily, and can therefore be made available in the much-used shapefile format. It was therefore decided to take the existing "shp2postsql" shapefile loader by Jeff Lounsbury, available on the PostGIS site [8] (based in turn on the ShapeLib library by Frank Warmerdam) and rewrite the C code to provide a shapefile loader for mySQL. The resulting "shp2mysql" program was then used successfully to transfer the test data for the case study city of Kathmandu (Nepal) into a text file of SQL CREATE and INSERT queries which can be run in the mySQL command line interface. The next step will be to port the code to Java and include it in the project as a Tomcat web service to provide a more robust, better integrated and user-friendly way of inputting data for the "supplier" users of the RIMapper system.

Conclusion

The RIMapper test bed described in this paper is first and foremost exactly that: a test bed. As such, it is and will remain to be a work in progress and not expected to result in a fully-functional production system. However, from the use of OpenGIS standards in a database backend and the building of Java services to provide client browsers with SVG mapping, useful insights were gained into the possible deployment of these techniques in the future SLARIM GDI to be build. Furthermore, the setup has proved to be so flexible and powerful that it is expected that it can be used to provide access to data for several other applications and projects currently under development at ITC.

References

[1] *Westen, C.J. Van (2002):* STRENGTHENING LOCAL AUTHORITIES IN RISK MANAGEMENT (SLARIM. Enschede, ITC; Internal report. http://www.itc.nl/research/policy/spearhead3/vwesten.asp

[2] *Köbben, Barend (2003)*: VERSATILE RISK INDICATOR MAPS USING SVG. Proceedings of SVG Open - 2nd annual conference on scalable vector graphics. 2003. Vancouver. pp. 14 [CD-ROM and website http://www.svgopen.org/2003/proceedings.html]

[3] *RIMapper web site:* http://kartoweb.itc.nl/RIMapper/ [accessed 30-11-03]

[4] *Groot, R. & J. McLaughlin (eds) (2000):* GEOSPATIAL DATA INFRASTRUCTURE – CONCEPTS, CASES AND GOOD PRACTICE. Oxford University Press, Oxford etc.

[5] *FDGC Clearinghouse Information Resource Page*: http://www.fgdc.gov/clearinghouse/clearinghouse.html [accessed 22-11-03]

[6] *ISO/TC211 (2000):* CD 19115.3, GEOGRAPHIC INFORMATION – METADATA. International Organisation for Standardization.

[7] *OpenGIS Consortium web site:* http://www.opengis.org/ [accessed 22-11-03]

[8] *PostGIS site*: http://postgis.refractions.net/ [accessed 28-11-03]

[9] *mySQL AB site:* http://www.mysql.com/ [accessed 22-11-03]

[10] *Neumann, Andreas (2003)*: DELIVERING INTERACTIVE TOPOGRAPHIC WEB-MAPS USING OPEN SOURCE DATABASE TECHNOLOGY. Proceedings of SVG Open - 2nd annual conference on scalable vector graphics. 2003. Vancouver. pp. 13 [CD-ROM and website http://www.svgopen.org/2003/proceedings.html]

[11] *Apache Foundation Tomcat pages:* http://jakarta.apache.org/ [accessed 24-11-03]