

ADVANCED LBS TO PDA

Francesco Bartoli, University of Tor Vergata, Roma
Ing. Fabrizio Bernardini, A&C 2000 Srl, Roma

In this context a service is a server-based application that delivers geographical and geographical-related data to mobile clients (PDA) on demand. The query, analysis and retrieval operations are performed on the server. The management of the map presentation is performed on the client with a flexible approach. The service has a number of important properties: it is always available (subject to security and connectivity constraints), it can support lightweight clients, it can process multiple clients at the same time, it can be added to existing control infrastructures for critical applications scenarios, it provides GML-formatted data. The client application is developed in Java and uses a new virtual machine SuperWaba optimized for PDA and supported on both PalmOS and Windows CE. The PDA sends requests (as XML) over TCP/IP protocols and interfaces to a local GPS receiver, which permits navigation of the locally composed map without maintaining the connection with the server. Maps are drawn in a purposely-developed Geo-Browser characterized by maximum flexibility and total application independency.

INTRODUCTION

GIS technology is often described as “a patchwork of independent data spaces”, managed with tools provided by a relatively small number of GIS vendors, which “has the potential to alter the way people work and live”. Typically this potential can be exploited only after a complex technological framework is established due to the “lack of a comprehensive and open geographical data space”.

A geographical data space is more often than not the result of the interaction of different collection of geographical data, in different formats and different coordinate systems, more or less linked to rich attribute data. The need to exchange all these data, over different media, with different protocols pushes toward a uniform, universal, definition of protocols and formats able to permit a free definition of the characteristics of each isolated data space while preserving the possibility to merge them in a bigger, overall, picture. Open standards are the solution toward this goal and to these standard we turned when we tackled the problem we wished to solve for a practical implementation.

The now ubiquitous availability of reasonably powerful PDA devices, has well as the possibility to remotely connect to the Internet thanks to mobile services and, of course, the availability of low-cost/size/power consumption GPS receivers, identifies these three technologies has the enabling ones for the dissemination, and presentation, of geographical data in critical applications. Namely we were interested in applications, like management of civil emergencies, law enforcement, quick response surveys, and so on, where the geographical data management infrastructure shall be flexible enough to adapt to continuously changing requirements, even within the same application context. In other words we looked at the possibility to put into practice an “on-the-fly map” concept for mobile users in dynamic environment, thus leaving the “closest restaurant” paradigm to shift to a more generic “situation data + available resources” one.

While developing the idea and during its implementation, we discovered that all the main building blocks were already available.

PROBLEM DEFINITION

Typically a Location Based Service responds to a user query (containing the location of the user) with a practically fixed depiction of the immediate surrounding of the user. The PDA device often becomes just a viewer for images provided by a remote server. In variable application contexts, this is a serious limitation.

In a typical scenario where a situation arises with short-term notice, one, or more users are dispatched in an operation area without any proper map or plan. While the PDA may already hold basic cartographical information, the operation-relevant data and related resources are assembled into a database accessed through a main server. Upon reaching the operations area the user will connect to the server downloading one or more data spaces provided for him. Other data spaces may be accessible, for auxiliary information (i.e. resources), from other servers. By requesting one or more data spaces a PDA-client application, defined as Geo-Browser, collects, stores, depicts and shows the “on-the-fly” map which results from the fusion of all the available data. The merged representation, controlled by user visualization preferences should be usable for navigation, identification and data entry purposes with the capability to send the server updates and annotations. Server interaction should be assumed to be sporadic without a real need for a continuous connection. The representation, based on background cartographical information (switch able in content) and on multiple layers of geographical data, will be completely independent from the application. Dynamic creation of new symbologies and definitions should also be supported.

With this context in mind these are the resulting top-level requirements:

- Availability of data to mobile users operating in ‘real-world’ complex scenarios.
- Need to present detailed geographical information despite limited hardware capabilities.
- Need to update data in ‘real-time’ or ‘almost real-time’ with the assumption of a not permanent Internet connection.

Application Scenario

In a typical Civil Emergency scenario it is possible to identify two key interacting roles: the Control Center and the Mobile User.

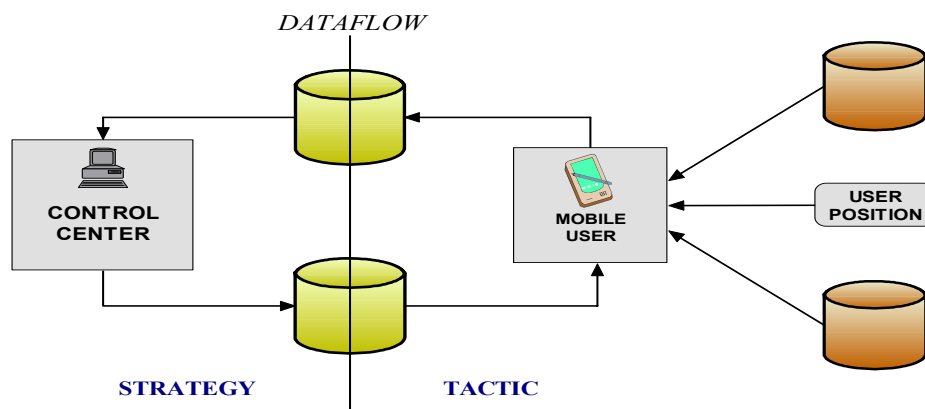


Figure 1: Civil Emergency scenario.

The Control Center manages the overall strategic picture of the emergency situation on a wide scale and update a server which can be accessed by mobile users (operators).

Operators in the field acts on information provided by the Control Center and perform tactical choices and collects local data. Decisions and data can be forwarded back to the control center via the server. The Mobile User has the option to contact other data repositories which may help his/her decision making process.

This typical environment for management of civil emergencies has been chosen for demonstrating the application of spatial data and GML.

Idea

The basic idea for satisfying the requirements is based on the analogy with the World Wide Web. The WWW model employs the HTTP protocol to transfer information described in html markup language for

rendering in a html browser. The browser renders the information depending on the hardware/software platform and on styles defined by the user.

Geographical data can be transferred to a geographically-enabled browser using the same paradigm: a transfer protocol used by a human-readable markup language permits to access unstructured information from multiple servers.

The basic elements of the analogy already exist. The transfer protocol can be the same protocol (HTTP) used on the WWW. The markup language will be based on XML, which permits to define geographical objects, and related attributes, without any previous agreed structure.

The third element, the browser, is a specialized multi-platform software component that renders geographical objects in order to compose maps (instead of WWW documents) and offers the following characteristics:

- geographical data persistence in a local cache;
- multi-layer presentation based on classification of available objects;
- refresh of local cache based on objects time-tag info or expiry dates;
- availability of an alternate input for user location data and/or dynamic data from a local interface (messages, positions, ...);
- availability of a raster background (locally available or remotely downloaded);
- 'empty document' is a coordinates grid on which is presented, if available, the current user position.

The single most important feature of this browser will be the fact that information accessed from multiple sites can be composed in a single geographical representation (a map). This is a major departure from the WWW browser analogy (where, typically, a web page shows content from a single server, or from multiple server, without a possibility for the user to "compose" the content of the page).

SOLUTION

The solution foresees three main areas of interest: the Client, the Server and the exchange Protocol(s), and the data Formats.

Client

A mobile environment imposes strong restrictions on a LBS thus creating a demand for information relevance:

- Mobile terminals have limited memory, limited computational power, limited screen size and resolution, limited battery life, and input devices that usually do not allow a user to operate quick enough.
- Mobile networks have high cost, limited bandwidth, high latency, low connection stability, and low predictable availability.

LBSs are often used in critical circumstances (e.g. during emergency situations) thus a user should be able to select the amount of data to be displayed in order to improve the decision making process.

The Geo-Browser becomes the key software component of the mobile terminal and is used to present to the user a local area drawn for an object-oriented representation of geographical entities. The current location of the user is also represented on the map and continuously updated gathering inputs from the connected GPS. The interface supports basic operations like panning and zooming and the customizing of the map using data accessed through the Internet connection.

Modern PDA's, equipped with a GPS receiver and an Internet-compatible communication link (e.g. a cellphone) are now able to handle the platform requirements for the Geo-Browser. The availability of Java language-based development tools for PDA's make the effort more attractive in term of the wider user which will be reachable.

Server

The choice of the server architecture conditions the entire application. Starting from the WWW analogy we initially considered a simple HTTP-based approach. From there, and after searching the available literature, four different architectures have been identified.

Basic HTTP solution

The Hypertext Transfer Protocol is a simple and elegant method which permit to request a file from a server or to pass a string of parameters to a server-side application (through the Common Gateway Interface). Both the file or the application output are served back to the client as a stream of data. The universal availability of HTTP servers and the simplicity of the approach makes this protocol a valid choice for any kind of experimentation. The major, important, drawback is that the format for expressing the request is not generic; while being flexible for many applications the HTTP request does not permit sending a file to express a complex request.

Web Map Server solution

The WMS solution is for HTTP-based implementation of LBS services which, from a simple user request, produce a static image (typically a map) to the client. This solution is acceptable for many constrained applications (like cellphones) and for not critical applications.

Geographical Data Transfer Protocol solution

This solution is an advanced and elegant mechanism which, in practical terms, redefines HTTP (GDTP has therefore been assigned by IANA a different TCP port) and permits sending an xml formatted query (a file) to a server. Query results are GML formatted files. The GDTP server is a dedicated product which interacts with a spatial data base.

Web Feature Server solution

WFS is a well-defined kind of hybrid solution where a series of geographical operations are designed and managed through HTTP's CGI requests. Despite the operations are defined in xml, the real transaction are performed with standard HTTP value/parameter mechanisms and are routed to a CGI application which interface to a geographical data store.

A difficult choice

In practice, the GDTP architecture offers the maximum possible flexibility. The WFS solution is a close possible choice but is inherently rigid for what pertains complex, and more generic, requests. On the other hand, GDTP require the installation of a dedicated server product, while WFS leverages on readily available technologies, like the HTTP server and the CGI interface.

A GDTP-based architecture is important for our application scenario because it enables the transfer of data also from the client to the server. This capability is particularly useful not only to express rich geographical features requests, but also to perform generic data base requests (answered with GML) and to send data collected from the roving user. In particular it is foreseen the need for a user to send a request aimed at exploring the attributes associated with the geographical entities in the server database and/or to update attributes (or even create new features) as a result of activities performed in the field.

Despite the fact that flexibility of request is an important requirement for our application scenario, we decided to implement a hybrid HTTP-based solution as an interim architecture for the server (this to avoid the development of GDTP software component and concentrate on the client software). In this architecture, not too complex requests are expressed in xml and sent to a server application, the Geo-Server, by means of the CGI interface (this limits the possible length of the requests). The Geo-Server interprets the request, interacts with a data base, and produce a GML-formatted response.

Formats

A key aspect of the protocols selection is the choice to use geographic vector data by means of which data are sent to the client in vector form, instead of the bitmap format, on which most of the existing LBS's rely on. This choice brings the following advantages:

- Linkage of spatial data with attributes (data associated to geographical entities).
- Explicit identification of entities.
- Customization of maps both by definition of colors and symbols and by selection/de-selection of data spaces.
- Reduce storage requirements.

Vector data are easily represented by means of XML, Extensible Markup Language (XML). One specific "application" of XML, called GML (Geography Markup Language) is dedicated to the encoding of geographical data including the associated attributes. The transfer of GML-described geographical data can be easily accomplished through HTTP (the hypertext transfer protocol) on which the World Wide Web is based. GML is an OpenGIS Consortium specification.

One of the basic principles of the conceptual model behind the GML specification is the separation of content and presentation. Because GML is still very new, there are currently not many viewers that can display GML. In order to render GML data returned from server side in a visual map we talk about GML rendering. Two methods have been considered: a method based on Java classes that read and graphically represent the GML document and a method based on transformation of GML into another XML format, SVG.

A viewer that can be developed is based on the transformation of GML into SVG (= Scalable Vector Graphics). SVG, yet another XML application, is a new format especially developed for the display of graphic content (pictures and animation) on Internet. It is a standard of the W3C Consortium and can be displayed in standard Internet browsers with the help of a plug-in.

The transformation from GML into SVG can be done in the browser or as a batch process with a SVG file as a result. XSLT (= eXtensible Stylesheet Language for Transformations) can be selected as tool for the transformation of GML into SVG.

There is also a little SVG supported small device like PDA, cellular phones, etc called eSVG.

While SVG is an established technology it presents a series of limitation for our application (in particular available software components are just "viewers" for SVG formatted data). In order to achieve the maximum flexibility from the Geo-Browser we decided therefore to proceed with a development directly based on GML parsing and a set of Java classes for the rendering of geographical objects.

SCENARIO REPRESENTATION

The conceptual model behind our prototype has been derived from TOP10vector project developed at TU Delft. It describes a Civil Emergency scenario; it has been converted into a technical model, also described in UML class diagrams and based on the OpenGIS Simple Feature specification. This specification contains standards to represent geo-spatial features and geo-spatial feature collections. Technical data modelling is described below, which also includes the technical UML schemas.

In addition to the conceptual UML model of the Civil Emergency, the required structure of a GML model has quite an impact on the final GML prototype of the Civil Emergency. GML is described by two XML Schemas: the geometry schema and the feature schema. The UML schemas belonging to these two XML Schemas of GML are shown in Figure 2 and Figure 3. As we will see GML organises features in collections has a serious impact on the overall technical model of the Civil Emergency prototype. For example, it is quite particular that geometry is not represented in the application model as an attribute of an application object, but is inherited from the GML class AbstractFeature. Also the fact that was been created two

technical models is related to the structure and rules of GML. Basically, two models are available in the GML specification: one model with and one model without member restrictions.

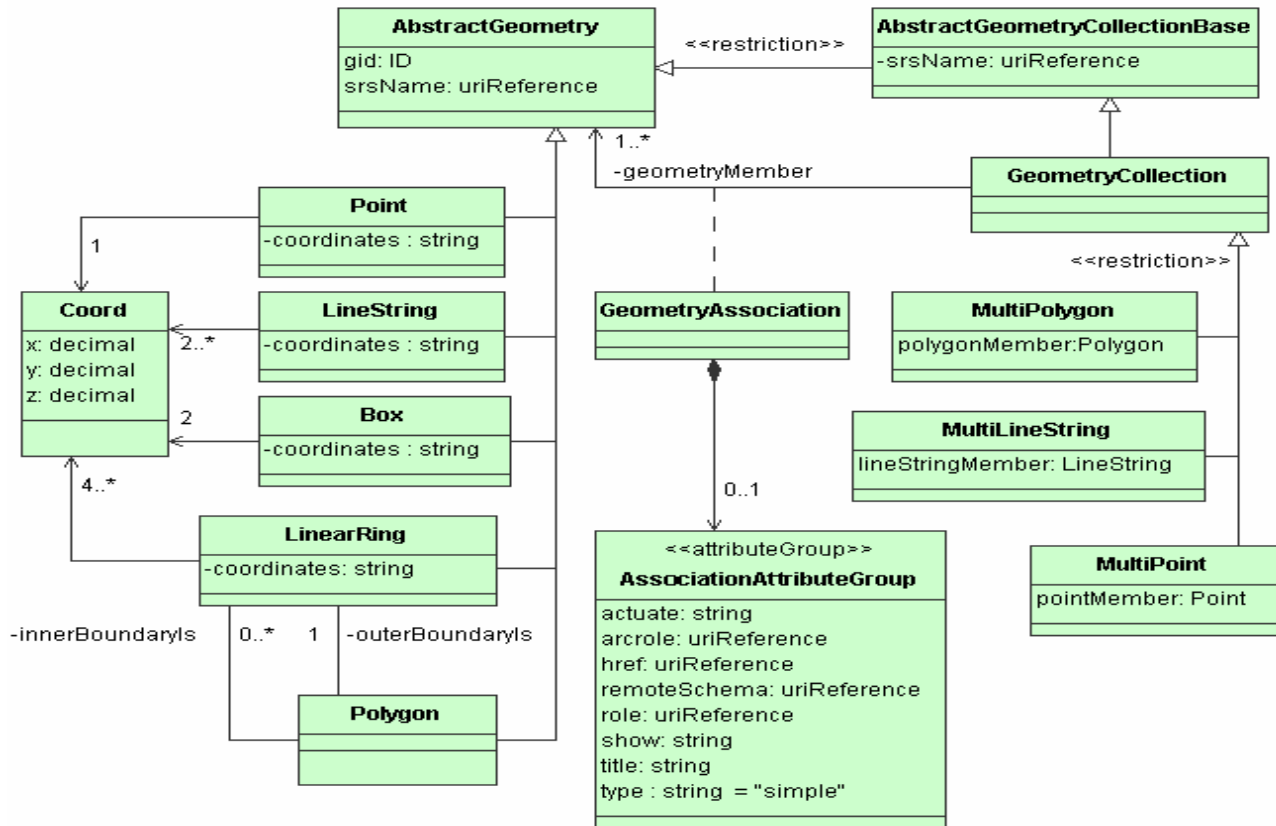


Figure 2: UML representation of the GML Geometry schema, taken from the GML 2.0 standard.

Description about technical and conceptual UML model

The UML schema of the conceptual model is given per main feature category. That is there are actually UML schemas for road part (“Tratto_Strada”), railroad part (“Tratto_Ferrovia”), water part (“ZonaAcquatica”), building (“Edificio”), terrain (“Terreno”), energetic resources (“Risorse_Energetiche”), protection resources (“Risorse_Intervento” as Ambulance structure, Firemen, Police, Army etc) and zones (for “ZoneFunzionali”, “ZoneAmministrative”, and “ZoneGeografiche”). All these classes inherit from a generic class geographic object (“OggettoCatastrofe”), which has an ID and two time attributes and a relationship to the meta data object class. The technical model integrates the different UML schemas in one overall schema. In this process, additional (inheritance) structure is created: road, railroad and water are all derived from the more general class infrastructure (“Infrastruttura”). The main reason for this is that these classes have many attributes in common. In order to avoid repeating these all the time in the model, it was decided to introduce this additional class infrastructure.

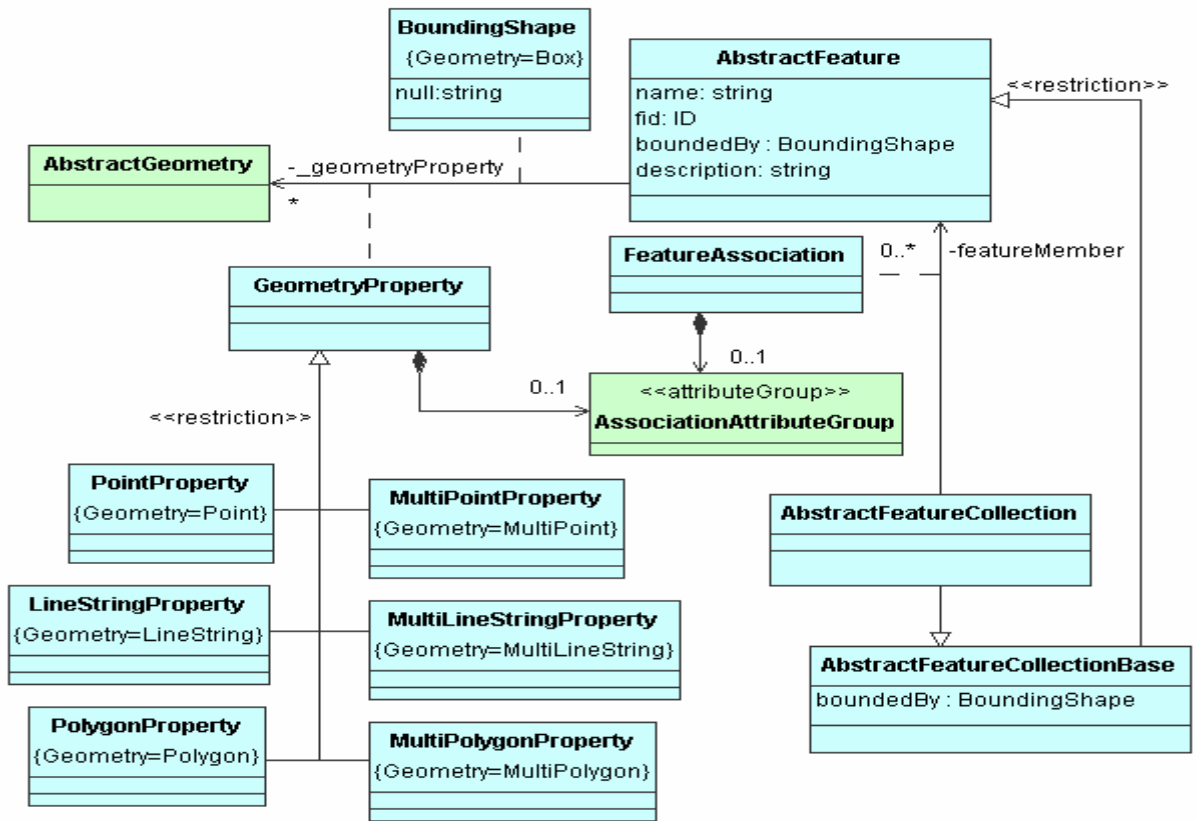


Figure 3: UML representation of the GML Feature schema, taken from the GML 2.0 standard.

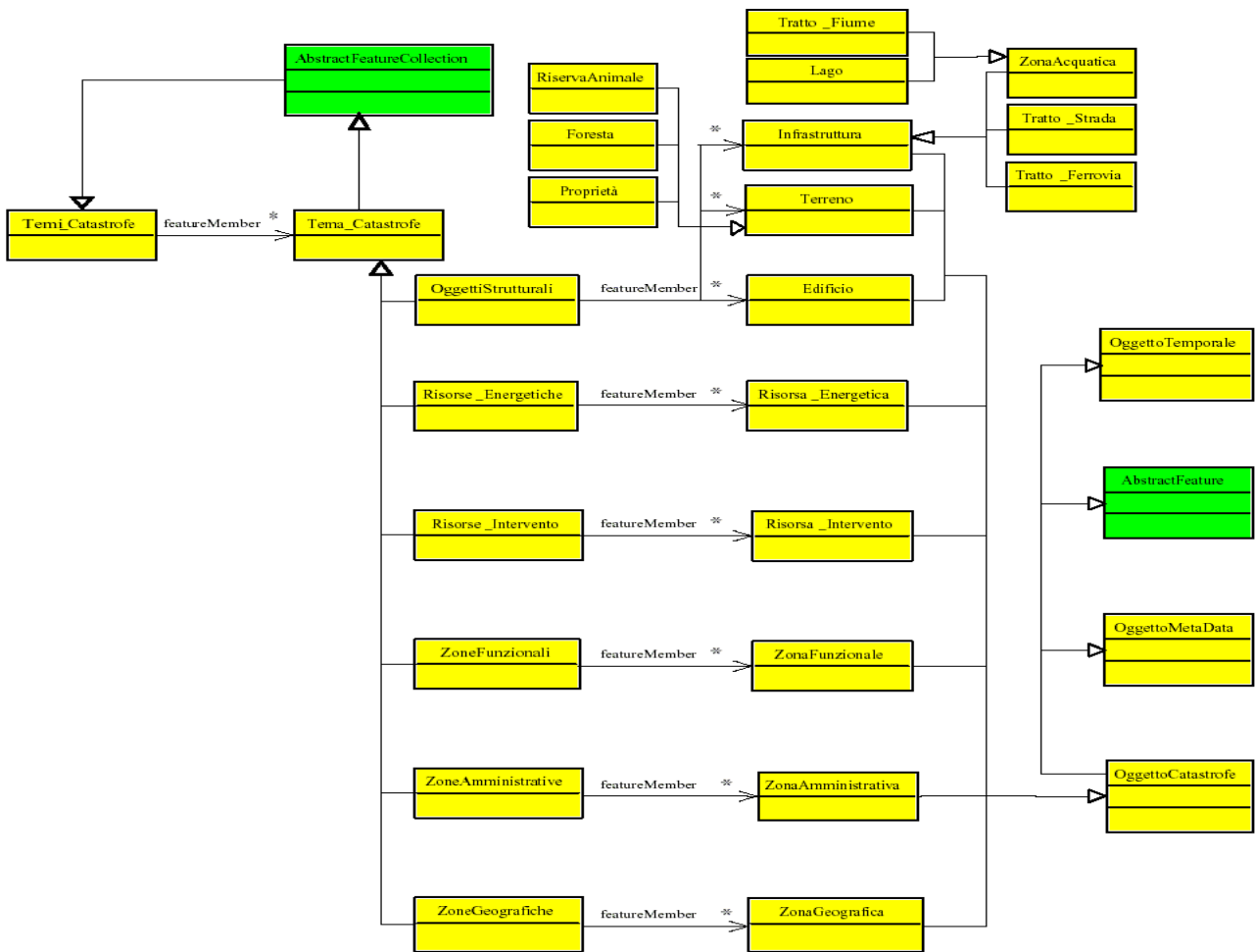


Figure 4: UML application model of Civil Emergency without member restrictions Explicit collections in the technical UML model

The conceptual model is at class level only. The instances are from these classes. In addition the technical model also recognises explicitly sets of instances, which form the explicit collections of the classes. Six “set” classes are modelled in the technical UML schemas:

1. spatial objects (“OggettiStrutturali”), which contain instances from the classes infrastructure, terrain and buildings;
2. energetic resources (“Risorse_Energetiche”), which contain instances from the class energetic resource; note the difference between plural for the set and singular for the individual element;
3. protection resources (“Risorse_Intervento”), which contain instances from the class protection resource;
4. functional zones (“ZoneFunzionali”);
5. administrative zones (“ZoneAmministrative”);
6. geographic regions (“ZoneGeografiche”).

Finally there is one set of sets in the model and this is called Disaster_Themes (“Temi_Catastrofe”), which has as its elements instances. These instances are from the class Disaster_theme (“TemaCatastrofe”), which are in turn again also collection (classes). Specialisations of this abstract class Disaster_Theme are the set classes which have already been described above: spatial objects, energetic resources, protection resources, functional zones, administrative zones and geographic zones. The reason for explicitly modelling Disaster_Themes is that an XML document can have only one “root” element. This corresponds to the “set of sets”, that is an instance of the class Disaster_Themes. An alternative would be creating several GML

documents, one for each collection with instances from a specific specialisation class (that is 6 in total), for one data set. This was considered to be an inferior solution.

The *OggettoCatastrofe* inherits from three classes: the *AbstractFeature* (GML basic class for a feature with geometry), *OggettoMetaData* (contains everything related to meta data) and temporal object (“*OggettoTemporale*”, which contains the temporal aspects). Though XML only supports single inheritance (in contrast to UML which offers multiple inheritance), there are standard methods to implement this in XML, one is called the “copy down” approach by using so-called “group” tags in XML.

Collection with or without membership restrictions

This aspect raises the discussion to the design goal of an elegant and readable form which is in harmony with the UML schemas from GML. In both the final technical UML model and the final GML/XML schema it was tried to make a readable and elegant design. Due to the different requirements (conceptual schema, actual data available, GML/XML rules) this was not always very easy. One complicating factor is that in order to have explicit control over the members of a set, the GML rules require explicit *FeatureAssociation* classes. One additional difference between the “non-strict” technical UML model (without explicit membership restrictions) and the “strict” technical UML model, is that the latter introduces restrictions to the values of the attributes. Now we consider only “non-strict” technical model in the analysis of Civil Emergency scenario.

Civil Emergency GML prototype

We discussed some of the issues that arose when translating the conceptual model for the Civil Emergency scenario into a technical data model. We will focus on the step from the UML models to the GML prototype files. We start with some general remarks about GML and XML Schema.

GML, XML and XML Schema

XML documents must respect some basic rules: they must be “well formed” (have corresponding begin and end tags) and “valid”. An XML document is valid when its tags and structure conform to the definitions and declarations in either a DTD document (=Document Type Description) or an XML Schema document. The Civil Emergency GML prototype are based on version 2.0 of the GML specification. With version 2.0 the OpenGIS consortium decided only to use XML Schema to convey the structure of GML files.

While XML and GML documents are “easy to look at”, because of the begin and end tags that wrap the data, XML Schema proved not that easy at first. There are four basic constructs:

- declaration of elements
- definition of types (*simpleTypes* and *complexType*s)
- the possibility to define subtypes by extending or restricting *superTypes*
- use of aliases (*substitutionGroup* mechanism)

The GML specification is basically a set of XML Schema documents with element declarations and type definitions plus an hierarchical structure for the relationships between types. In this way the specification offers a framework that can be used by organisations to make their specific XML application schemas for their GML implementations. Technically this is accomplished by including the XML Schemas *feature.xsd*, *geometry.xsd* (both from the OpenGIS consortium) and *xlinks.xsd* (from the W3C Consortium) in the user's application schema.

For the geometry elements and types users can decide to use the ones already present in *feature.xsd* and *geometry.xsd*. The only thing user organisations always have to do is to define their own feature elements and feature types. This is done by extending the OpenGIS basic features types. Also a root level feature collection must be defined.

GML has some “rules” that must be respected when users construct their own XML application schema:

- all user defined feature types must (directly or indirectly) inherit from *AbstractFeatureType*
- all user defined geometry property types must (directly or indirectly) inherit from *GeometryPropertyType*
- all user defined geometry types must (directly or indirectly) inherit from *AbstractGeometryType* or *GeometryCollectionType*

Catastrofe schema definition

We want to give some ideas of our schema definitions: we define it in our workspace named “prociv”. The `<import>` tag, imports other schema definitions that are used. In this case the schema definition `feature.xsd` is included, and from that other schema definitions are included again: `geometry.xsd` and `xlinks.xsd`. So we have a schema definitions header like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
  <!-- File: prociv.xsd -->
    <schema targetNamespace="http://www.protezionecivile.it/prociv"
      xmlns:prociv="http://www.protezionecivile.it/prociv"
      ...
  <!-- import constructs from the GML Feature and Geometry schemas -->
  <import namespace="http://www.opengis.net/gml" schemaLocation="feature.xsd"/>
```

Root element

The first `<element>` in the schema definition defines the root element of the GML file. This means that the first open tag in the GML file will be `<Temi_Catastrofe>`. As can be seen in the definition the `substitutionGroup` for this element is `gml:_FeatureCollection`. This means that `<Temi_Catastrofe>` is a collection of objects.

```
<element name="Temi_Catastrofe" substitutionGroup="gml:_FeatureCollection">
  <complexType>
    <complexContent>
      <extension base="gml:AbstractFeatureCollectionType">
      </extension>
    </complexContent>
  </complexType>
</element>
```

Shared definitions

There are some attributes that are shared by many objects in the PROCIV definition. In the UML structure these attributes are stored in multiple superclasses. Since XSD does not support multiple inheritance, we modelled shared definitions with the group construct. We have groups for `OggettoTemporale` and `OggettoMetaData`:

```
<group name ="OggettoTemporale">
  <sequence>
    <element name="iniziodata" type="string"/>
    <element name="finedata" type="string"/>
  </sequence>
</group>

<group name ="OggettoMetaData">
  <sequence>
    <element name="ospedaletype" type="string"/>
    <element name="nomeospedale" type="string"/>
    <element name="prontosoccorso" type="boolean"/>
    <element name="indirizzo" type="string"/>
    <element name="codiceprociv" type="integer"/>
  </sequence>
</group>
```

```

    </sequence>
</group>

<complexType name="OggettoCatastrofeType" abstract="true">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="Catastrofe_id" type="integer"/>
        <group ref="prociv:OggettoTemporale"/>
        <group ref="prociv:OggettoMetaData"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

Feature definitions

After all preliminary constructs we finally get the definitions of the feature Objects. These are the objects that are really visible on the map. Below only the definition of “Edificio” is given

```

<element name="Edificio" type="prociv:EdificioType"
substitutionGroup="gml:_Feature"/>
<complexType name="EdificioType">
  <complexContent>
    <extension base="prociv:OggettoCatastrofeType">
      <sequence>
        <element name="tipo" type="string"/>
        <element name="funzione" type="string"/>
        <element name="proprietario" type="string"/>
        <element name="numPersone" type="integer"/>
        <element name="stato" type="string"/>
        <element ref="gml:geometryProperty"/>
        <element name="numpiani" type="integer" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

Catastrofe document

This section contains an annotated GML file. It starts with the standard XML header in which the character encoding is mentioned:

```

<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!-- File: prova.gml -->

```

Then we get the root element of the GML document. In this header no default namespace is declared. This means that all tags in the gml document are prefixed by their namespace (“gml:”, “prociv:”).

```

<prociv:Temi_Catastrofe

```

```

xmlns:prociv="http://www.protezionecivile.it/prociv"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.protezionecivile.it/prociv prociv.xsd">

```

The <gml:boundedBy> tag contains the bounding box of all the objects in the GML file. The srsName attribute that is given in the geometry of the bounding box (and every other geometry in the GML document) is a reference to the EPSG database. In the GML2.0 specification the srsName should be provided, but a standardized format for the different Spatial Reference Systems is currently not part of the standard; it can be any URI. We know that GML3.0 release has a module for specifying Spatial Reference Systems.

```

<gml:boundedBy>
  <gml:Box srsName="EPSG:7408">
    <gml:coordinates>
      190000,446000,0.0 193000,449000,0.0
    </gml:coordinates>
  </gml:Box>
</gml:boundedBy>

```

After the bounding box of all collections we get the different collections one by one. First the collection is opened, then we get the bounding box of this sub-collection and after that there are the objects of the collection itself. Once we have had the objects of the first collection that collection is closed and we continue with the following collection until all collections are described. Notice that all co-ordinates have three dimensions.

```

<gml:featureMember>
  <prociv:OggettiStrutturali>
    <gml:boundedBy>
      <gml:Box srsName="EPSG:7408">
        <gml:coordinates>
          190000,446000,0.0 193000,449000,0.0
        </gml:coordinates>
      </gml:Box>
    </gml:boundedBy>
    <gml:featureMember>
      <prociv:Tratto_Ferrovia fid="PROTEZIONECIVILE.4200001">
        ...
      </prociv:Tratto_Ferrovia>
    </gml:featureMember>
  </prociv:OggettiStrutturali>
</gml:featureMember>
<!--other featureMember-->
...
</prociv:Temi_Catastrofe>

```

An example of one complete feature is given below.

```

<gml:featureMember>
  <prociv:Tratto_Ferrovia fid="PROTEZIONECIVILE.4200001">
    <prociv:Catastrofe_id>4200001</prociv:Catastrofe_id>
  </prociv:Tratto_Ferrovia>
</gml:featureMember>

```

```

<prociv:iniziodata>06 Jul 2001 08:08:24</prociv:iniziodata>
<prociv:finedata/>
<prociv:tipomateriale/>
<prociv:nomeospedale/>
<prociv:prontosoccorso/>
<prociv:indirizzo/>
<prociv:codiceprociv>4233</prociv:codiceprociv>
<prociv:type>urbano</prociv:type>
<prociv:stato>agibile</prociv:stato>
<prociv:veicolo>Tram</prociv:veicolo>
<prociv:nome_tratta>tuscolana</prociv:nome_tratta>
<prociv:tipo_porte>Automatiche</prociv:tipo_porte>
<prociv:numero_binari>1</prociv:numero_binari>
<prociv:funzione>struttura trasporto persone</prociv:funzione>
<prociv:TensioneElettrica>tensionepresente</prociv:TensioneElettrica>
  <gml:geometryProperty>
    <gml:Polygon srsName="EPSG:7408">
      <gml:outerBoundaryIs>
        <gml:LinearRing>
          <gml:coordinates>
            191008.456,447232.635,0.0 190990.713,447236.938,0.0
            190972.849,447239.952,0.0 190955.904,447235.469,0.0
            190940.491,447231.646,0.0 190923.831,447229.355,0.0
            190924.668,447229.093,0.0 190942.211,447223.787,0.0
            190944.282,447224.343,0.0 190957.89,447227.719,0.0
            190973.223,447231.776,0.0 190989.103,447229.096,0.0
            191006.57,447224.861,0.0 191008.456,447232.635,0.0
          </gml:coordinates>
        </gml:LinearRing>
      </gml:outerBoundaryIs>
    </gml:Polygon>
  </gml:geometryProperty>
  <prociv:numpiani>0</prociv:numpiani>
</prociv:Tratto_Ferrovia>
</gml:featureMember>

```

IMPLEMENTATION

Client Hardware

- Advancements in hardware performance have allowed the development of small, low powered devices suitable for mobile geographic applications. Higher-end systems offer a quarter VGA screen (320 by 240 pixels), 8-bit color, 32 MB of RAM, and a 200MHz, or more, processor. The choice of operating systems includes Windows CE, Palm OS, and Linux.
- Typically, but not necessarily, to be useful as mobile geographic clients these devices must be geographically aware, that is, it must be possible to locate the device quickly and with reasonable accuracy.

It is therefore a natural choice to adopt a GPS receiver into the client-side hardware.

To test Internet connectivity we turned to a GSM cellphone with the understanding that future applications will make a good use of the expanding GPRS technology. While the cellphone can be considered a “standard” tool in the field we also considered the usability of OEM GSM/GPRS modules. These modules, with a very small form factor can be integrated with a OEM GPS receiver module to build a very compact unit. This integration can also be advantageous to overcome the operational limitations imposed by the availability of a single serial (or USB) port in the PDA. Currently this limitation makes impossible the simultaneous access to the cellphone and the GPS receiver. This is however a minor problem since the application philosophy foresees only occasional contacts with the server: during the contact the user position is not updated; when the contact ends, the GPS resume feeding data to the PDA.



Figure 5: Hardware testing application phases.

Client Software

A part from a few low level drivers which may be required in some PDA implementations, the only software under development for the mobile user PDA is the Geo-Browser. The technology of choice in this case is Java, for its extensive availability on small platforms, which renders porting of the application particularly easy. Also the Java framework make easier the management of the object-oriented geographical representation, as well as the handling of XML formatted data and includes all the GUI element required for the graphical aspects of the application.

In line with the intent to keep all the software effort in the realm of freely available software we adopted a high successful free JVM optimized for PDA devices called SuperWaba.

Geo-Browser

Geo-Browser is a Java-based client-centric, relatively lightweight, self-contained, application that runs on Pocket PCs and PalmOS devices and offers several important capabilities:

- Pen-based GUI
- Symbolize multiple layer types
- Edit vector objects and higher level features
- Edit feature attributes
- Composition of geographic and attribute queries
- Serial/USB interface to GPS receivers and other local data sources
- TCP/IP interface to geographical servers
- User customization of presentation



Figure 6: Geo-Browser testing.

With reference to the previous figure, the main functionalities of the Geo-Browser area accessible by means of small buttons on top of the “map area”. Those are:

- P Pan
- Z Zoom
- F Find (a feature, or attribute data)
- S Show (selection of features)
- L Layer (switching)
- N New (feature)
- E Edit (feature characteristics or attributes, delete)
- M Measure (distances, bearings, ...)
- C Configure (map)
- ? Info (about the selected feature)

At the global application level the Geo-Browser interface is complemented by a menu bar, whose content might be platform-dependent.

Viewer with simple GML parser

The Geo-Browser is completely written in Java and can easily be extended. This has been done by writing a new class, referencing to a kDOM parser to build GML tree from current data stream, which displays graphical objects associated to kDOM objects. The main process to render streaming data is GML parsing that assign structure to sentences with a given grammar. There are two basic models for XML document parsing: SAX and DOM. We want to perform a general comparative analysis of DOM and SAX, and to explain through this the choice of model used. The models are not in direct competition with each other; each has strengths and weaknesses and can even be combined in certain applications. The most important difference between SAX and DOM is that SAX analyzes XML document as a serialized event stream with a sequence of calls to an handler function where each chunk of XML syntax is recognized. Instead DOM, after all XML syntax is recognized, builds a tree with a root element.

The greater disadvantage of SAX approach is that it does not support random-access handling of the document to access the data after parsing is done. But this disadvantage becomes an advantage where the user can discard information that will not be needed. Therefore the use of SAX can result in reduced memory overhead compared to DOM, which requires to retain the complete document as a tree in memory. Contrarily to how it can be thought we are using DOM (in particular kDOM) for rendering the GML data. It is easy and efficient to traverse the tree generated by parsing using the DOM and render a GML feature map on the basis of node information extracted from the tree. At API level Geo-browser directly takes GML data from the server side and renders the map in the client. This approach is flexible and allows client side processing like query and integration of data from different servers. DOM model is appropriate because of the choice to implement Geo-browser and display GML document returned from data stream by building a tree of objects which is used by a Java API to display GML data on the PDA device.

kXML parser

While the J2SE/J2EE platform offers excellent support for XML processing, the limited environment of a PDA is too “tight” for it.

Third-party groups offer many XML parsers that require little processing power. Most, such as NanoXML and TinyXML, only support the simpler serial SAX-style parsing. The kXML package developed by Enhydra.org offers both SAX and DOM style parsers. Since a fully W3C-compliant DOM parser is too heavy for small devices, kXML uses a lightweight alternative called kDOM and offers packages specialized for SOAP (kSOAP) and XML-RPC (kXML-RPC) message processing.

kXML, which is especially suited for Java applications running on mobile devices, offers the following features:

- XML Namespace support
- "Relaxed" mode for parsing HTML or other SGML formats
- Small Memory footprint
- A Pull-based parser for simplified parsing of nested / modularized XML structures
- XML writing support including namespace handling
- A lean document object model kDOM
- Optional WAP support (WBXML/WML)

Restrictions include the following:

- kXML does not support user defined (external) entities.
- The doctype declaration is not parsed. However, a corresponding "legacy event" is generated by the parser, so application programmers are able to parse the doctype declaration themselves.

The choice of limiting the scope to kDOM can be taken from JDOM: XML processing in JAVA should be as simple as JAVA itself. The problem with JDOM is that it depends on many classes that are not available in VMs for small devices. Whereas the goal of JDOM is simplification, kDOM tries to add a significant reduction of memory usage crucial on small devices like PDA.

Application phases

There are mainly three phases in the Geo-Browser application: location phase, connection phase, and portrayal phase.

In the location phase the user position is continuously obtained by reading the NMEA0183A stream coming from the GPS receiver connected to the PDA. This is performed by a GUI class and a GPS connection class which manages GPRMC sentences coming from GPS.

In the connection phase uploading of requests and downloading of data are managed by an HTTP connection class that handles XML formatted requests for the server through a serial socket (GSM). Downloaded data are stored in memory device and can be used persistently until they are deleted for any reasons. This is a crucial point of the Geo-Browser application because the connection is not permanent and the user may

require to redefine the representation more times. This also enable using the same data storage for GML data obtained from different servers and composing them in a single view or in multiple layers..

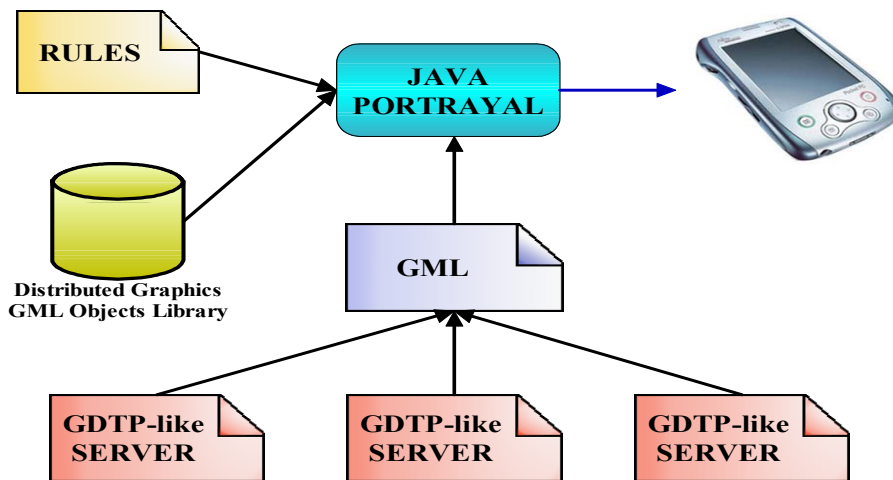


Figure 7: Portrayal phase overview

In the portrayal phase the java viewer reads GML features (eventually filtered) from the DOM tree storage and it interprets them, following a set of rules, like java objects within a java graphics package visualization. Currently we are at the level of a static graphics implementation but we are working for dynamic visualization to generalise the rendering and displaying API with Styled Layer Descriptor associated with GML documents returned from servers. Styled Layer Descriptor(SLD) is a standard syntax released from OGC for the specification of presentation; with SLD(XML-tags) it is possible to describe the symbolization of feature data that is transferred by servers.

The three phases are coordinated by user inputs on the pen-based GUI. The GUI manages also all the aspect related to the interaction with the graphical representation and offer additional features, like the possibility to input queries for the server.

All the above mentioned classes have been developed and run on the SuperWaba VM.

SuperWaba VM

The SuperWaba Virtual Machine environment defines a language, a virtual machine (VM), a class file format, and a set of foundation classes. Legally speaking, SuperWaba is not Java and the foundation classes are simplified in comparison to the standard Java classes as well as the VM which has certain limitations. Other than this, a programmer familiar with Java is able to write SuperWaba applications immediately, using conventional development tools. SuperWaba is currently gaining popularity among programmers, and a Linux port is in process.



Strength of SuperWaba

- SuperWaba is well suited for embedded and mobile devices, as well as embedded PCs (OS on one floppy)
- SuperWaba is free and open source (released under the GNU GPL)
- Applications written in SuperWaba can be run everywhere a JVM is available.
- The VM is written in standard ANSI C
- The VM and classes takes 293 Kb of executable code on Motorola 68K processors, and about double that on a Pentium

- It is the fastest VM able to run bytecode, hence very interesting for embedded systems with low-cost, low-end CPUs. The diagram below is a performances search for different VMs tested on CLIE PEG-770C (33 MHz)

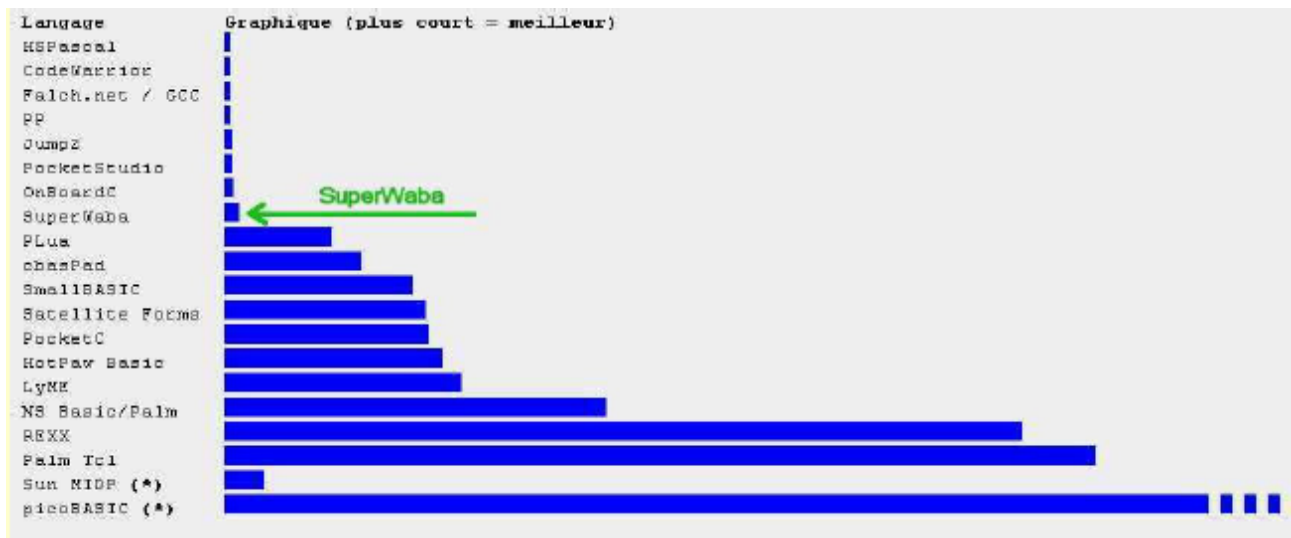


Figure 8: Search for real perfect number.

SuperWaba is first language with runtime and it is about three times faster than J2ME (working time: SuperWaba 48s/J2ME 135s)

- It can run standard byte code encoded in its standard format (the class file), produced by any Java compiler and it has a no-frills, deterministic garbage collector

SuperWaba is a serious competitor to Sun's J2ME (KVM), but with the advantage of being completely free.

Organization of SuperWaba VM software

The VM code is structured so as to ease porting onto different platforms such as PalmOS and PocketPC OS, and it is organized as follows:

- VM core, platform-independent
- OS abstraction routines, interfacing the core to the OS, include: class loader, startup and exit functions, file manipulations, windowing, sound, networking, etc. functions
- C implementation of the native routines, called by the core Java classes (themselves written in Java)

Currently, the foundation classes are:

- waba.fx*: this package provides graphics and drawing classes(Color, Font, FontMetrics, Graphics, ISurface, Image, Rect, Sound, SoundClip).
- waba.io*: this package includes input/output classes(Catalog, SerialPort, Socket, Stream, etc), including the important SerialPort class. This is a key ingredient for GPS applications requiring direct access with a GPS receiver via an RS-232 connection. *java.lang*: This package contains some classes (Object, String, StringBuffer, Throwable, etc) from the original java.lang, and those classes contains only a subset of the methods.
- waba.sys*: Classes that contains functions to deal with the underlying Operating System characteristics and configurations, and conversion classes(Convert, Time, Vm, CharacterConverter, etc). Convert which converts data types from waba data types to system types; Time, which gives you access to the current date and time; Vm which allows communication with virtual machine and CharacterConverter used to correctly handle international character conversions.
- waba.ui*: The most important package, with all user interface controls you need to create good and fast programs. Note that all user interface gadgets has two styles, original Palm OS and beautiful Windows

CE 3d controls, available in all devices. (Check, Container, Control, ControlEvent, Edit, Event, IKeys, KeyEvent, Label, Button, MainWindow, PenEvent, Radio, TabPanel, ScrollBar, ComboBox, ListBox, Timer, Welcome, Window, etc).

- *waba.util*: Utility classes, to deal with date, random number generation, and data structures (Vectors and Hashtables).

The overall result is an extremely reduced footprint, rich enough to enable writing interactive applications on a PDA (e.g. Palm). Many classes, though slightly different, resemble standard Java core classes but are actually much simpler. Some other classes, like the *io.Catalog*, were designed to exploit specifics of PalmOS and do not really have an equivalent in standard Java.

One can extend SuperWaba's functionality by supplying additional Java classes, or implementing a package including native methods (to speed execution when required).

Server Software

As already stated, the server solution has been subject to a careful evaluation for different solutions exist. In order to test a GDTP-based concept without having to develop a full GDTP-server, we adopted a mixed approach where GDTP-like requests (that is, expressed as short xml files) are sent to a CGI application accessed through a standard HTTP server. While this solution is not efficient and incurs in all the limitations of HTTP, it permits to develop the Geo-Browser concept with good confidence of its behaviour in the "real world". In the future the server side of the application will be reviewed, pending also possible advancement in the WFS architecture or a wider acceptance of GDTP.

Server Architecture

The server architecture is a classical one, based on the HTTP-server (Apache), the CGI application which interprets GDTP-like, XML-expressed, queries and a geographical data base (implemented with MySQL). It is important to note that the geographical data base may contain also many not geographical data which may be pertinent to the scenario. Using the generalized XML requests it is possible also to query those data using the same transport mechanisms.

This server architecture is therefore extremely simplified and, other than the data base and its related authoring tools (scripts), it requires the development of a single CGI application. The platform of choice for this development is, again, Java.

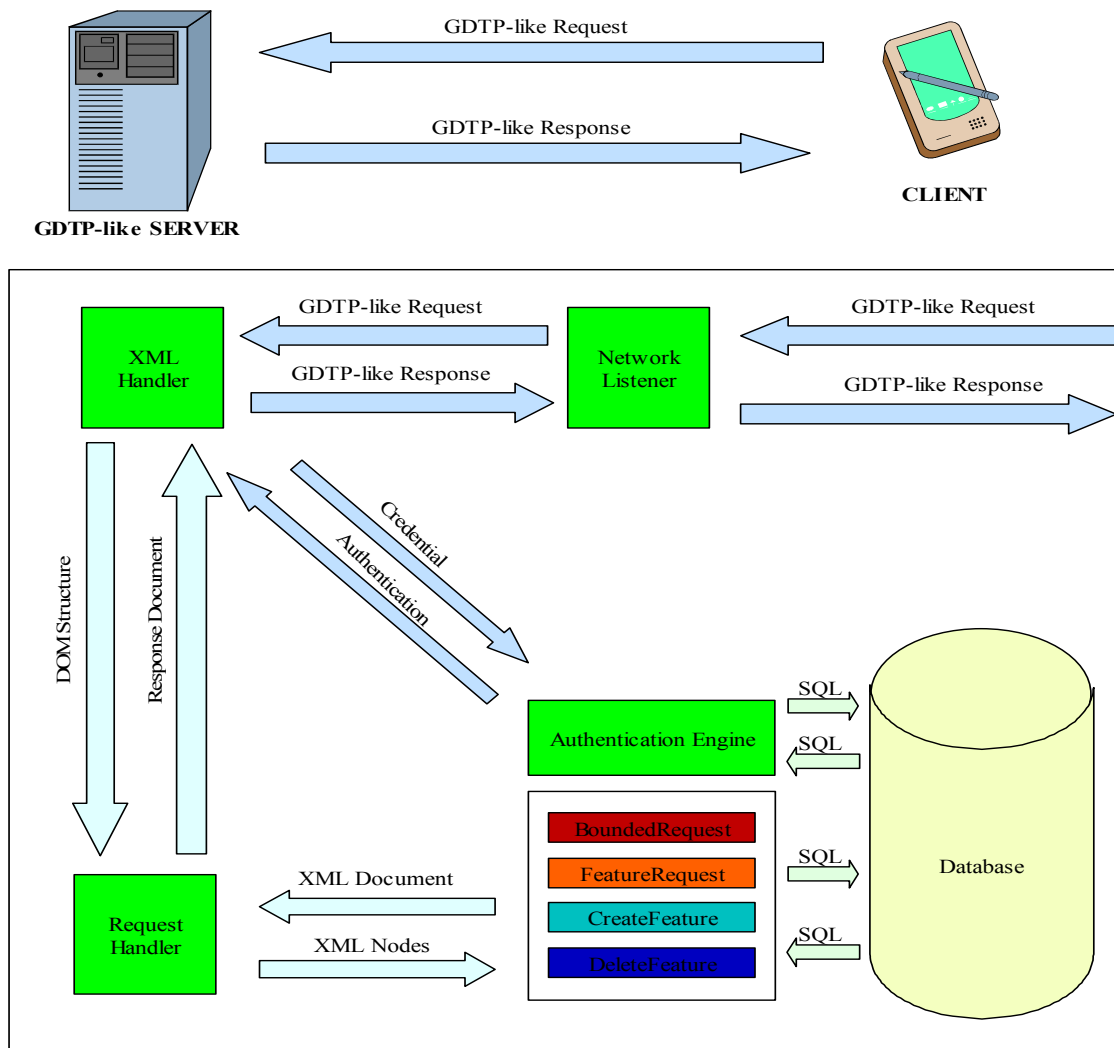


Figure 9: Server-side application for processing requests

GDTP-like requests

The Geographic Data Transfer Protocol (GDTP) is XML-based so request and response communication between client and server are encoded as an XML document. The request document is divided in two parts, a metadata part and a core request. Metadata contents of the request present information about user credentials, and other information useful to the server. Core request is a choice of one of the following request types:

- create a feature,
- request information on a specified feature,
- request for all features contained within a specified bounded area,
- delete a feature,
- modify a feature,
- request for a feature associating to a certain search criteria, and
- request server information.

The *boundedRequest* query type, one of the most useful, allows to search for a collection of features contained within a certain geographic area. This request type can be combined with filters to get powerful location data. For read-only requests, username and password credentials are optional, and access permissions depends only by features associated to the request. On the other side write access, such as

creating or deleting a feature, requires controlled access by user credentials. A response returned by the server contains answers about success or failure of an operation, and the data requested as a GML document.

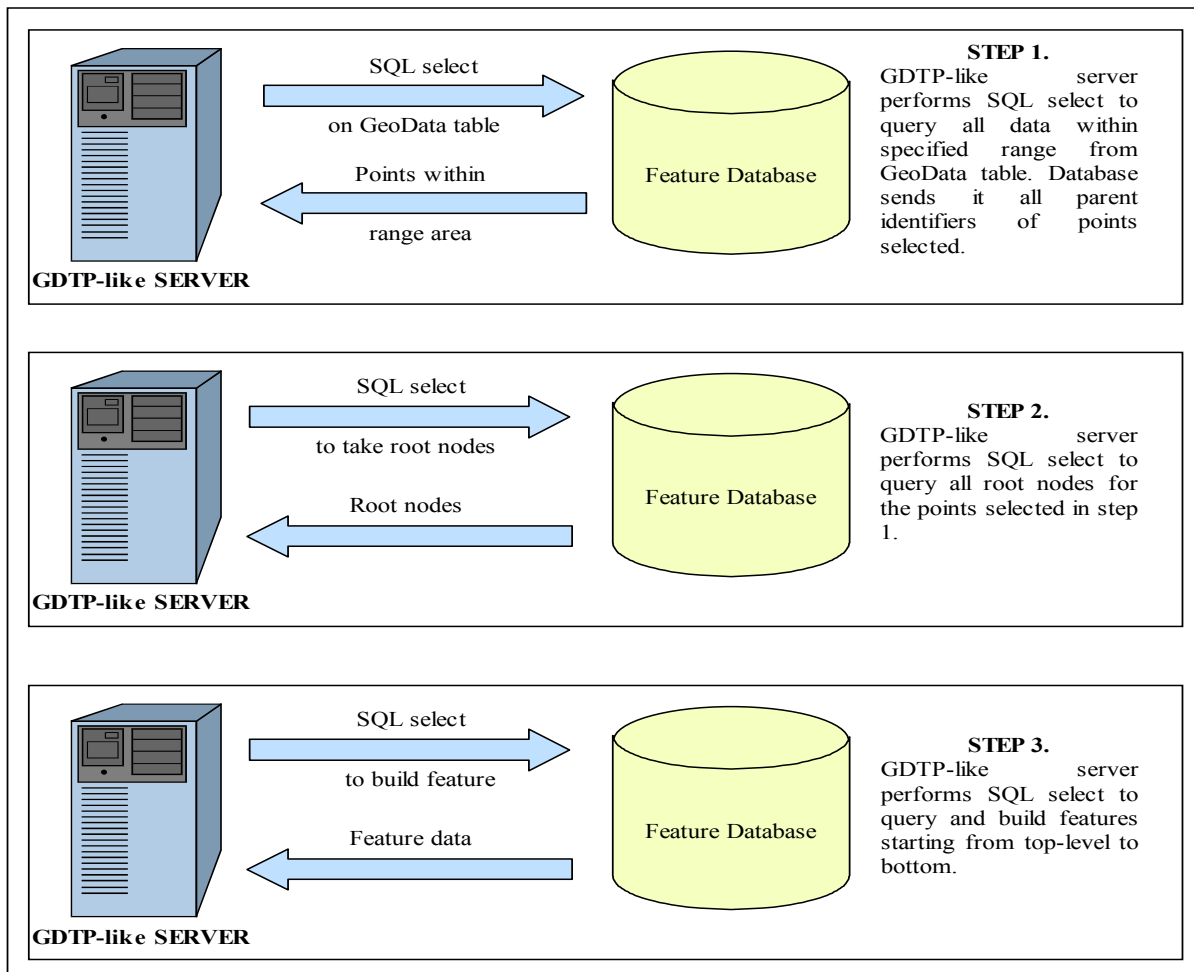


Figure 10: The communications between the GDTP server and the feature database are required to complete a boundedRequest.

Database Architecture

The feature data in MySQL database was separated between geographic and non-geographic contents. We have optimized tables with geographic contents through indexes to perform efficient spatial queries, and tables with related data required to build XML data trees were kept in other tables. Furthermore, server configuration data and access control framework information are contained within the database.

We have chosen a general relational database because MySQL offered many advantages with limited disadvantages, but in future implementations of this server, as spatial queries will become more complex and intensive, it could be necessary to switch to a spatial database.

The set of the tables within the database is formed by several operational data tables and two main content tables. Operational data tables contain configuration information, user definitions, group memberships, permissions, host access tables, and network access tables. This information is used to provide to access control, associating features with users, allowing users to form groups in order to share data, and what hosts and networks are allowed or denied access to the server.

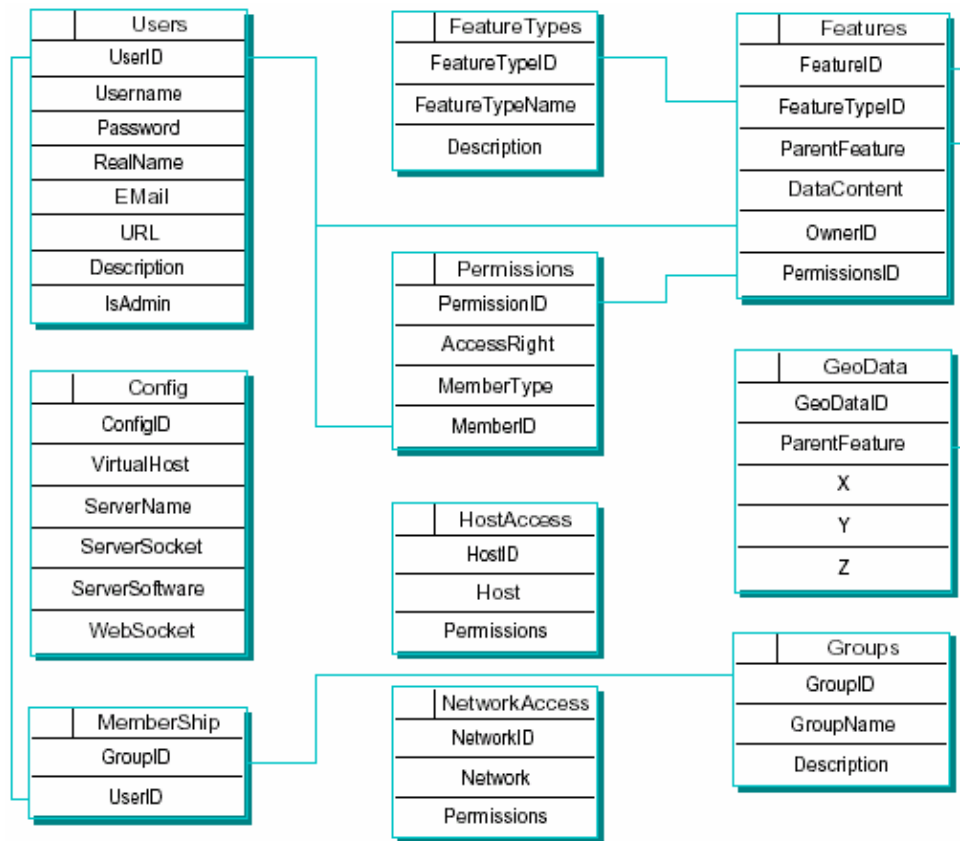


Figure 11: Data Base schema, implementation of the test scenario.

This database architecture works well for smaller to medium data sets, and it provides a good general interface for the Java components of the GDTP server to query and manipulate features efficiently, but there are still a few problems. First problem is that as the number of features in the database increases, the access time is degraded above all in insert operations because of database must rebuild indexes for each insert. Furthermore, an increasing period that decreases performance is used to bring data from the permanent storage into physical memory and this effect is more evident on a computer with limited physical memory.

As we said before this architecture allows to adapt itself to emerging spatial technologies resolving several problems.

CONCLUSIONS

The project describe in this paper is spawned by an engineering thesis and it is in a advanced state of implementation. All the single technologies, both hardware and software, have been tested and the first release of the client-software, the Geo-Browser, will permit single layer operations of multiple collections of data. The server-software analysis is complete and the development of the HTTP-based, GDTP-like, solution will be start as soon as the Geo-Browser is validated. Testing of the Geo-Browser will be performed with fixed files sent through HTTP.

In closing it is important to note that all the elements, hardware, software and formats/protocol, exist to implement geographical data distribution solutions that are based on not-proprietary representations of data. The capability to generate generic maps from different sources render the Geo-Browser concept both minimal and powerful. In addition, it shall be remarked that, like for many other GIS product/solutions, applications exists outside the strictly geographical context, as soon as the problem to be tackled lends to a spatial representation.

References

- [1] Chris Karr. *An Information Architecture for Sharing and Aggregating Geospatial Content*
- [2] OGC 01-029. *GML2.0 Specification*. <http://opengis.net/gml/01-029/GML2.html>
- [3] OGC 01-068r3. *Web Map Service Implementation Specification 1.1.1*. <http://www.opengis.org/techno/specs/01-068r3.pdf>
- [4] OGC 01-023. *Web Feature Server Specification*. <http://www.opengis.org/techno/discussions/01-023.pdf>
- [5] OGC 02-070. *Styled Layer Descriptor Implementation Specification*. <http://www.opengis.org/techno/specs/02-070.pdf>
- [6] Chris J. Karr, Dept. of Computer Science Princeton University. *Geographic Data Transfer Protocol User's Guide*. <http://gd.tuwien.ac.at/opsys/linux/sf/subcat/server/geoserver/gdtp-user-guide.pdf>
- [7] W3C. *Extensible Markup Language (XML) 1.0 (Second Edition)*. <http://www.w3.org/TR/REC-xml>
- [8] W3C. *Scalable Vector Graphics (SVG) 1.0 Specification*. <http://www.w3.org/TR/SVG/>
- [9] W3C. *XSL Transformations (XSLT) Specification Version 1.0*. <http://www.w3.org/TR/1999/WD-xslt-19990421.html>
- [10] M.E. de Vries, drs. T.P.M. Tijssen, drs. J.E. Stoter, drs. C.W. Quak and prof. dr.ir. P.J.M. van Oosterom. *The GML prototype of the new TOP10vector object model*. *GIS Report No. 9 Delft, December 2001*
- [11] Shashi Shekhar¹, Ranga Raju Vatsaval^{1j2}, Namita Sahay¹, Thomas E. But-k³, Stephen Lime³. *WMS and GML based Interoperable Web Mapping System*
- [12] Tjahjadinata Setiawan. *The Use of J2ME with a Campus Portal for Wireless Devices*
- [13] WDN journal. *Deliver Mobile Services Using XML And SOAP*. February 8, 2001
- [14] Jason Hunter. *JDOM Makes XML Easy*
- [15] The Enhydra.org project. *kSOAP Project and kXML Project*. <http://ksoap.enhydra.org/>
<http://kxml.enhydra.org/>
- [16] MySQL open source relational database system. *More information about MySQL is available online at* <http://www.mysql.org>.
- [17] SuperWaba VM. *More information at* <http://www.superwaba.com.br>