UNIVERSITY OF PIRAEUS

DEPARTMENT OF TECHNOLOGY EDUCATION & DIGITAL SYSTEMS

POST-GRADUATE STUDIES PROGRAM

NETWORK-CENTRIC SYSTEMS MODULE

Trajectory Data Visualization: The VisualHERMES Tool

MSc THESIS

by

IOANNIS S. GKOUTSIDIS

(Rec. No.: ME/0567)

Piraeus, June 2008

Disclaimer

This thesis is submitted as part requirement for the MSc Degree in **Network-Centric Systems Module** of **Technology Education & Digital Systems Department** at University of Piraeus, Greece. It is substantially the result of my own work except where explicitly indicated in text.

The thesis will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Author: **Ioannis S. Gkoutsidis**

Signature:

Date: **June 2008**

Abstract

Composition of time and space in a unified data framework results into spatio-temporal databases. Spatio-temporal Database Management Systems (STDBMS) are able to process, manage and analyze spatio-temporal data. HERMES provides spatio-temporal functionality to Oracle 10g Object-Relational Database Management System (ORDBMS). It introduces time-varying geometries that change their position and/or extend in space and time dimension either discretely or continuously, extending PL/SQL, the data definition and manipulation language of Oracle 10g, with spatio-temporal semantics. Currently, its main use is for representing moving objects (cars, trucks, people etc.) trajectories.

The problem of geospatial data interoperability has been an issue throughout the Geographic Information Systems (GIS) industry for a long time. The Open GIS Consortium (OCG) developed an eXtensible Markup Language based standard, Geography Markup Language (GML) with the intension to overcome this issue. GML is the standard for transport and storage of geographic information for those who need spatial and temporal sharing. GML, being a subset of XML, separates the content from presentation. Making maps from GML data involves a transformation of GML data into a display format that can be interpreted by viewer software. Keyhole Markup Language (KML) is an XML-based formatting standard that can be used to visualize GML data, using projection engines. The transformation of GML data into KML can be accomplished using an eXtensible Stylesheet Language Transformation (XSLT) stylesheet together with an XSLT processor. The XSLT stylesheet is an XML-based document that describes how data in a GML document is transformed into graphic elements in the KML document. By using different stylesheets, the same GML dataset can be visualized differently. In the same way, different datasets having a homogenous schema can use the same stylesheet. The possibility to use the same stylesheet for visualizing different datasets could be very useful in geographic data visualization.

This thesis is focused on two domains;

- In designing and implementing a wrapper for transforming raw HERMES's data into GML entities and process specialized GML queries sent to the database and
- Transforming these GML results into KML files for visualization purposes, via third-party projection engines.

Περίληψη

Η ενοποίηση του Χώρου και του Χρόνου σε ένα ενιαίο πλαίσιο έχει ως απόρροια τη δημιουργία χωρο-χρονικών βάσεων δεδομένων. Τα Χωρο-Χρονικά Συστήματα Διαχείρισης Βάσεων Δεδομένων (Spatio-Temporal Database Management Systems – STDBMS) είναι σε θέση να επεξεργαστούν, να διαχειριστούν και να αναλύσουν χωρο-χρονικά δεδομένα. Το HERMES παρέχει χωρο-χρονικές λειτουργίες στο Αντικειμενο-Σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (Object-Relational Database Management Systems – ORDBMS) της Oracle στην έκδοση 10g. Εισάγει χρονικά μεταβαλλόμενες γεωμετρίες, οι οποίες είναι σε θέση να αλλάζουν τη θέση ή/και το σχήμα τους στο πεδίου του χώρου ή/και του χρόνου ανά τακτά διαστήματα ή/και συνεχώς. Έτσι επεκτείνει το PL/SQL (Procedural Language/Structured Query Language), την γλώσσα ορισμού και χειρισμού δεδομένων της Oracle 10g, με χωρο-χρονικές έννοιες. Στην τρέχουσα κατάσταση, η βασική του λειτουργία αφορά στην αναπαράσταση τροχιών κινούμενων αντικειμένων (αυτοκίνητα, φορτηγά, άνθρωποι κλπ).

Το ζήτημα της δια-λειτουργικότητας των γεωγραφικών δεδομένων αποτέλεσε πρόβλημα στη βιομηχανία των Γεωγραφικών Συστημάτων Πληροφοριών (Geographic Information Systems - GIS) για αρκετό καιρό. Το Open GIS Consortium (OGC) ανέπτυξε ένα πρότυπο βασισμένο στην XML (eXtensible Markup Language), το GML (Geography Markup Language) με σκοπό να υπερκεράσει το πρόβλημα της δια-λειτουργικότητας. Το GML αποτελεί πρότυπο για τη μεταφορά και αποθήκευση γεωγραφικών πληροφοριών. Όντας υποσύνολο της XML, διαχωρίζει το περιεχόμενο από την παρουσίαση. Η δημιουργία χαρτών από GML δεδομένα περιλαμβάνει την μετατροπή των δεδομένων αυτών σε μια μορφή παρουσίασης, ικανή να αναλυθεί από το εκάστοτε λογισμικό παρουσίασης. Το Keyhole Markup Language (KML) αποτελεί ένα, βασισμένο επίσης σε XML, πρότυπο οπτικοποίησης των GML δεδομένων, κάνοντας χρήση μηχανών παρουσίασης. Η μετατροπή των GML δεδομένων σε KML επιτυγχάνεται με την χρήση ενός XSLT (eXtensible Stylesheet Language Transformations) συνόλου κανόνων σε συνεργασία με έναν κατάλληλο XSLT επεξεργαστή. Το σύνολο αυτό των κανόνων μορφοποίησης αποτελεί ένα XML αρχείο, το οποίο περιγράφει τον τρόπο με τον οποίο ένα GML έγγραφο μετατρέπεται σε γραφικές οντότητες εντός ενός KML εγγράφου. Χρησιμοποιώντας διαφορετικούς κανόνες μορφοποίησης, το ίδιο GML σύνολο δεδομένων δύναται να οπτικοποιηθεί διαφορετικά. Κατά τον ίδιο τρόπο, διαφορετικά σύνολα δεδομένων που μοιράζονται ένα ομοιογενές πρότυπο, μπορούν να χρησιμοποιήσουν τους ίδιους κανόνες μορφοποίησης. Η δυνατότητα χρήσης κοινών κανόνων μορφοποίησης για την οπτικοποίηση διαφορετικών συνόλων δεδομένων μπορεί να αποβεί ιδιαίτερα χρήσιμη για την παρουσίαση γεωγραφικών πληροφοριών.

Η παρούσα μελέτη επικεντρώνεται σε δύο βασικούς άξονες:

- Στον σχεδιασμό και την υλοποίηση ενός ενδιάμεσου μηχανισμού (wrapper) για τη μετατροπή των πρωτογενών δεδομένων, που αποστέλλονται ως απόκριση από το σύστημα HERMES, σε GML όρους αλλά και την επεξεργασία ειδικών GML ερωτημάτων προς τη βάση και
- Τη μετατροπή των GML αποτελεσμάτων σε KML αρχεία για τους σκοπούς της οπτικοποίησης, με την χρήση μηχανών παρουσίασης τρίτων κατασκευαστών.

Acknowledgements

The writing of this thesis has been a long and arduous journey of learning. From the initial to the final product, this thesis has seen the input from many individuals.

First and foremost, I would like to thank my supervisors, Professor Georgios Vasilakopoulos and Assistant Professor Yannis Theodoridis, for their continued support, constructive comments and encouragement. Without their excellent guidance, this work would not have taken the present form. I feel privileged to have got this opportunity to work with them.

I am grateful to Dr. Nikos Pelekis and Gerasimos Marketos. Their contribution both in my infancy and technical issues related to my work was invaluable. The patience shown in my endless questions was proverbial.

I would also like to thank Assistant Professor Emmanuel Stefanakis for his valuable information and guidelines at the beginning of my work.

Mum and Dad, I could not have come this far without you. You believed in me and gave me the liberty to choose my life. No words can express how grateful I am.

Dedicated to my little brother Nikos.

Table of Contents

ABSTRACT ..... **I**

ΠΕΡΙΛΗΨΗ ..... **II**

ACKNOWLEDGEMENTS ..... **III**

TABLE OF CONTENTS ..... **IV**

TABLE OF FIGURES ..... **VII**

LIST OF TABLES ..... **IX**

ABBREVIATIONS ..... **X**

1 INTRODUCTION **1**

1.1 Background and motivation 1
1.2 Problem definition 2
1.3 Research objectives 4
1.4 Methodology 4

2 MOVING OBJECT DATABASES **5**

2.1 Introduction 5
2.2 Mobility scenarios 5
2.3 Capabilities of spatio-temporal databases 5
2.4 Evolution of spatio-temporal models 8
2.5 Spatial change in Moving Object Databases 9
2.6 Trajectory Representation 10
2.7 Spatial Database Management Systems 11
2.8 Spatio-temporal Extensions 12
2.8.1 HERMES-MDC 13
2.9 Conclusion 14

3 THE EXTENSIBLE MARKUP LANGUAGE (XML) 15

3.1 Introduction 15
3.2 Evolution of XML 15
3.3 The eXtensible Markup Language (XML) 17
3.3.1 XML document structure 17
3.3.2 Declarations 18
3.3.3 Elements 18
3.3.4 Comments 19
3.3.5 Character references 19
3.3.6 Processing instructions 19
3.4 Document Type Definition (DTD) 19
3.5 XML schema 20
3.5.1 Simple type 21
3.5.2 Attributes 21
3.5.3 Restrictions and extensions 21
3.5.4 Complex type 21
3.6 XML namespaces 22
3.7 Document Object Model (DOM) 22
3.8 Unicode system 22
3.9 Viewing XML documents 22
3.10 Stylesheets 23
3.11 XML based markup languages 23
3.12 Conclusion 23

4 THE GEOGRAPHY MARKUP LANGUAGE (GML) 24

4.1 INTRODUCTION 24
4.2 BACKGROUND AND EVOLUTION OF GML 24
4.3 GML FEATURES 25
4.3.1 Simple features 25
4.3.2 Geometry elements 25
4.3.3 Time elements 28
4.4 CORE GML SCHEMAS 28
4.5 ENCODING GEOGRAPHIC INFORMATION WITH GML 29
4.6 GML APPLICATION SCHEMAS 29
4.7 STRUCTURE OF AN APPLICATION SCHEMA 30
4.8 STRUCTURE OF GML DOCUMENTS 32
4.9 VALIDATION OF GML DOCUMENTS 32
4.10 VIEWING GML DATA 33
4.11 CONCLUSION 33

5 THE KEYHOLE MARKUP LANGUAGE (KML) 34

5.1 INTRODUCTION 34
5.2 VISUALIZATION OF GEOGRAPHIC DATA 34
5.3 EVOLUTION OF KML 34
5.4 KEYHOLE MARKUP LANGUAGE (KML) 34
5.4.1 Creating KML files 35
5.5 KML FEATURES 35
5.6 KML DOCUMENT STRUCTURE 37
5.7 VIEWING KML DATA 40
5.7.1 Google Earth 40
5.7.2 Google Maps 41
5.7.3 Other viewing applications 42
5.8 KML CRITIQUE 42
5.9 CONCLUSION 42

6 THE EXTENSIBLE STYLESHEET LANGUAGE TRANSFORMATIONS (XSLT) 43

6.1 INTRODUCTION 43
6.2 EXTENSIBLE STYLESHEET LANGUAGE (XSL) 43
6.2.1 XSL Transformation (XSLT) 43
6.2.2 XSL Formatting Objects (XSL-FO) 43
6.2.3 XML Path Language (XPath) 44
6.3 TREE AND NODES 44
6.4 XSLT STYLESHEET 44
6.5 XSLT STYLESHEET STRUCTURE AND ELEMENTS 44
6.5.1 Templates 44
6.5.2 Matching nodes 45
6.5.3 Selecting nodes 45
6.5.4 Named templates 45
6.5.5 Content of output 45
6.5.6 Output methods 46
6.6 COMBINING STYLESHEETS 47
6.6.1 Importing 47
6.6.2 Inclusion 47
6.7 EMBEDDING STYLESHEETS 48
6.8 CREATING AN XSLT STYLESHEET 48
6.9 XSLT PROCESSORS 50
6.10 CONCLUSION 50

7 VISUALHERMES WRAPPER 51

7.1 Introduction 51
7.2 Requirements 51
7.3 Design 53
7.3.1 Application architecture 53
7.3.2 Data Access Layer 53
7.3.3 Business Logic Layer 55
7.3.4 Presentation Layer 56
7.4 Application modules 56
7.5 Data flow 58
7.6 Development environment 59
7.7 Access to data 59
7.8 Implementation 60
7.8.1 System logon 61
7.8.2 Query building 61
7.8.3 Results construction 67
7.9 Compression features 70
7.10 Problems in implementation 72
7.11 Conclusion 72

8 CASE STUDY 73

8.1 Introduction 73
8.2 Dataset characteristics 73
8.3 Trajectory query type 73
8.4 Spatial Intersection query type 76
8.5 Temporal Intersection query type 80
8.6 Average Speed query type 83
8.7 NotSet query type 86
8.8 Upload GML query file 87
8.9 Conclusion 91

9 CONCLUSIONS 92

9.1 Open issues 92

BIBLIOGRAPHY 94

APPENDIX 98

A VisualHERMES GML Schema Definition File 98
B VisualHERMES eXtensible Stylesheet Language Transformations File 101
C HERMES TO_CLOB() Function 104
D Εκτενής Περίληψη στην Ελληνική 105

Table of Figures

Figure 1-1: Moving objects 1
Figure 1-2: Raw locations & reconstructed trajectories 1
Figure 1-3: Moving Object Databases architecture (Marketos, et al., 2008) 2
Figure 1-4: HERMES-based Trajectory Data Warehouse 3
Figure 1-5: Wrapper extension on HERMES-based Trajectory Data Warehouse 3
Figure 2-1: Spatio-temporal path for ongoing moving objects as a set of points 11
Figure 2-2: Simplified architecture of spatial databases 12
Figure 2-3: Entity-Relationship diagram for Oracle Spatial SDO_GEOMETRY (Kothuri, et al., 2007) 13
Figure 2-4: HERMES system architecture (Pelekis, et al., 2006) 14
Figure 3-1: Markup language concepts 16
Figure 3-2: Options for displaying XML documents 23
Figure 4-1: GML Abstract Feature Model (OGC, 2004) 26
Figure 4-2: GML Geometry Model (OGC, 2004) 29
Figure 5-1: Google Earth UI 40
Figure 5-2: Google Maps UI 41
Figure 6-1: Axes provided by XPath (Sun Microsystems, 1994) 46
Figure 7-1: GML query and SQL result 52
Figure 7-2: KML result 52
Figure 7-3: VisualHERMES as an alternative interface 53
Figure 7-4: VisualHERMES 3-tier architecture 54
Figure 7-5: Business Logic Layer double role 55
Figure 7-6: Business objects class diagram 56
Figure 7-7: Business logic layer class diagram 57
Figure 7-8: Data access layer class diagram 58
Figure 7-9: VisualHERMES data flow diagram 58
Figure 7-10: Data providers 59
Figure 7-11: Disconnected DB access architecture 60
Figure 7-12: VisualHERMES query screen 60
Figure 7-13: VisualHERMES logon screen 61
Figure 7-14: HERMES intersection operation 63
Figure 7-15: Dataset compression ratios 71
Figure 8-1: Trajectory web interface 73
Figure 8-2: Trajectory SQL query 73
Figure 8-3: Trajectory GML query 74
Figure 8-4: Trajectory results page 74
Figure 8-5: Trajectory GML results 74
Figure 8-6: Trajectory KML results 75
Figure 8-7: Trajectory visualized KML data 76
Figure 8-8: Spatial Intersection web interface 77
Figure 8-9: Spatial Intersection SQL query 77
Figure 8-10: Spatial Intersection GML query 78
Figure 8-11: Spatial Intersection results page 78
Figure 8-12: Spatial Intersection GML results 79
Figure 8-13: Spatial Intersection KML results 79
Figure 8-14: Spatial Intersection visualized KML data 80
Figure 8-15: Temporal Intersection web interface 80
Figure 8-16: Temporal Intersection SQL query 81
Figure 8-17: Temporal Intersection GML query 81
Figure 8-18: Temporal Intersection results page 81
Figure 8-19: Temporal Intersection GML results 82
Figure 8-20: Temporal Intersection KML results 82

Figure 8-21: Temporal Intersection visualized KML data 83
Figure 8-22: Average Speed web interface 83
Figure 8-23: Average Speed SQL query 84
Figure 8-24: Average Speed GML query 84
Figure 8-25: Average Speed results page 84
Figure 8-26: Average Speed GML results 85
Figure 8-27: Average Speed KML results 85
Figure 8-28: Average Speed visualized KML data 86
Figure 8-29: Manual build of SQL query 86
Figure 8-30: Uploading a GML query file 87
Figure 8-31: Complex SQL query 88
Figure 8-32: Complex GML query 88
Figure 8-33: GML query file results page 89
Figure 8-34: Complex query GML results 89
Figure 8-35: Complex query KML results 90
Figure 8-36: GML query file visualized KML data 90

List of Tables

Table 4-1: Basic geometric properties 28
Table 7-1: Trajectories as business objects 68
Table 7-2: Dataset sizes comparison 71

Abbreviations

ADO:	ActiveX Data Objects
API:	Application Programming Interface
ASP:	Active Server Pages
BLL:	Business Logic Layer
BO:	Business Object
CML:	Chemical Markup Language
CSS:	Cascading Style Sheets
DAL:	Data Access Layer
DB:	DataBase
DBMS:	DataBase Management System
DLL:	Dynamic Link Library
DM:	Data Mining
DSSL:	Document Style and Semantic Specification Language
DTD:	Document Type Definition
EPSG:	European Petroleum Survey Group
GIF:	Graphics Image Format
GIS:	Geographic Information System
GML:	Geography Markup Language
GUI:	Graphical User Interface
HTML:	HyperText Markup Language
JPEG:	Joint Photographic Experts Group
KML:	Keyhole Markup Language
MO:	Moving Object
MOD:	Moving Object Database
MOT:	Moving Object Trajectory
ODBC:	Open DataBase Connectivity
OGC:	Open Geospatial Consortium
OGP:	Oil and Gas Producers (International Association)
OLE DB:	Object Linking and Embedding DataBase
ORDBMS:	Object-Relational DataBase Management Systems
PDA:	Personal Digital Assistant
PDF:	Portable Document Format
PGML:	Precision Graphics Markup Language
PNG:	Portable Network Graphics
SDBMS:	Spatial DataBase Management System
SGML:	Standard Generalized Markup Language
SQL:	Structured Query Language
sRGB:	standard Red Green Blue
SRID:	Spatial Reference IDentifier
STDBMS:	Spatio-Temporal DataBase Management Systems
SVG:	Scalable Vector Graphics
VML:	Vector Markup Language
W3C:	World Wide Web Consortium

WebCGM:	Web Compute Graphic Metafile
WGS:	World Geodetic System
WML:	Wireless Markup Language
XHTML:	eXtensible HyperText Markup Language
XML:	eXtensible Markup Language
XSL:	eXtensible Stylesheet Language
XSLT:	eXtensible Stylesheet Language Transformations

1 Introduction

1.1 Background and motivation

Nowadays, mobile users are more than ever before. In addition, advances in modern Database Management Systems (DBMS) have made it possible to store all kinds of information relating to various sources of Moving Objects (MO). For example, using Global Positioning Systems (GPS) and mobile-wireless communications it is possible to store information related to the geographical position of a moving object depending on time (Figure 1-1).

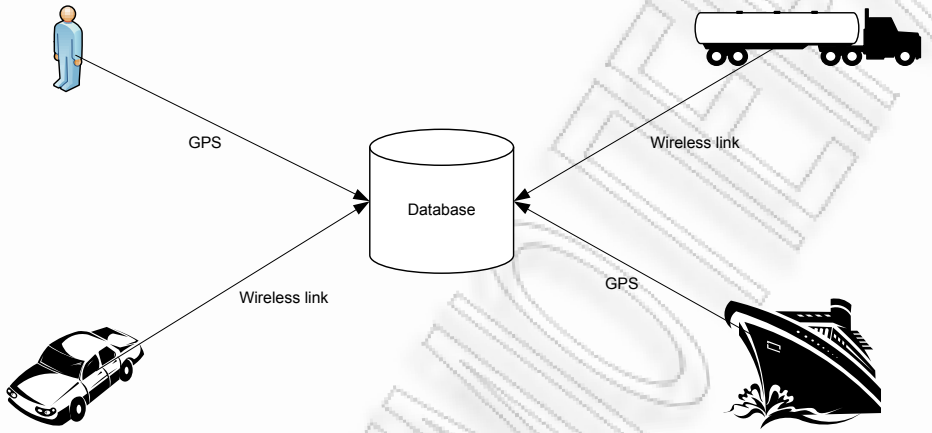


Figure 1-1: Moving objects

At a second level, is a combination of such information from the perspective of the moving object, creating the so-called Moving Object Trajectories (MOT). Of course, the process of producing trajectories from a range of points depending on time requires custom software, which operates on top of a database management system and is able to model the required data structures supporting spatial and temporal concepts (Figure 1-2).

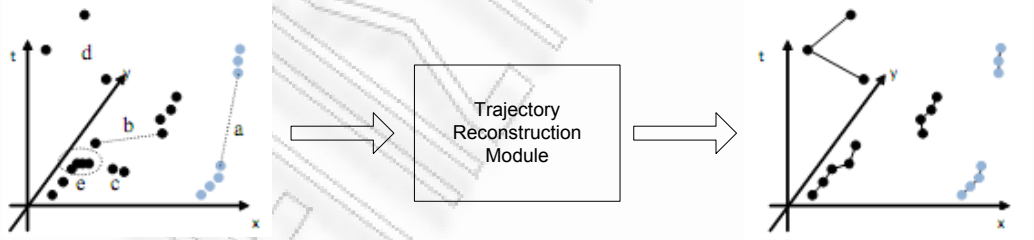


Figure 1-2: Raw locations & reconstructed trajectories

The next stage is to feed the Moving Object Database (MOD) with all the information necessary to facilitate the various functions of Data Mining (DM) and taking into account the geographical context, as well as the various geographical levels, enabling export useful conclusions and ultimately knowledge in this field of research (Figure 1-3).

It is understandable that, mining knowledge is only one sector, in a wider field of research related to spatial and temporal information of moving objects. Other fields in this area may include patterns extraction, visualization of information etc.

A critical point for the operation of a system, which manages both trajectories of moving objects, and acts as a source of such information to third parties, is the transfer of data. More specifically, the need for interoperability at the level of migrant data to and from a Spatial Database Management System (SDBMS) is imperative. Interoperability in this case refers to the possibility of such a system to send data formatted into a self-described notation, which in turn is identifiable by each party.

Such an approach requires mediation, of custom software, which will aim to convert outbound, from the database management system, data in a commonly acceptable format such as eXtensible Markup Language – XML (Bray, et al., 2006). In addition, all incoming data are urged to comply on the basis of a

predefined schema, so as to obtain the acceptance by the database management system for further processing. A competitive advantage arising from the use of XML standard is that because it is all about spatial data management, a more specific description can be used, which is directly associated with geographic concepts. This standard is Geography Markup Language – GML (OGC, 2004). With even the most recent additions to 3.1.1 version of GML, it is possible to model both spatial and temporal characteristics, as well as handle moving objects, through their respective extensions.

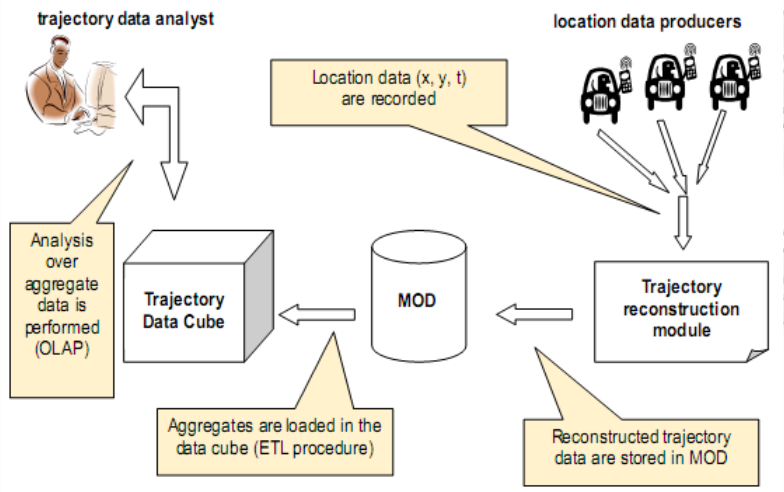


Figure 1-3: Moving Object Databases architecture (Marketos, et al., 2008)

The introduction of mechanisms for formatting data (wrapping) certainly is not limited to the use of XML and GML standards. It is possible to model data on other known and commonly accepted standards, such as that of Keyhole Markup Language – KML (Google, 2008), which bears several similarities with that of GML, but also the implementation of custom standards representation, in accordance with their respective needs.

The use of generally accepted standards based on XML, such as those of GML or KML, outside the known benefits such as self-description, scalability, easy processing and handling, platform independence, etc., has some disadvantages, which lie mainly in nature of the spatial and temporal data and the way they are formatted. As with the XML format of data, it should be noted that GML documents tend to be large in size, mainly because of the existing extensive hierarchy GML 3 has. This makes data more difficult and complex to be processed, particularly when it comes to describe large volumes of data. One solution to this issue could provide some method of compression (i.e. Deflate algorithm (Deutsch, 1996)), having a direct impact as increasing complexity of the system and processing times.

It is conceivable that the use of XML wrappers and/or parsers, who are involved in the process of exchanging data to and from a database management system, is a fairly good solution to the issue in interoperability, with the ultimate objective of creating a loosely-coupled system for this purpose.

1.2 Problem definition

HERMES (Pelekis, et al., 2006) is a flexible computing framework, capable to support the design and development of spatial and temporal databases. In addition, it provides the necessary infrastructure for posing queries to a database with moving objects, whose location, shape and size vary over time.

This framework has been developed as an extension, which provides spatial and temporal capabilities to the Oracle's object-relational database management system on version 10g (Oracle Corp., 2003). In addition, it is designed in such a way that someone can use it either as a purely spatial or purely temporal system, but its main contribution is to support management of continuously moving objects.

In the current version, HERMES serves as a repository for trajectory data, through which a user comes to have a trajectory set for a moving object of interest in an object-relational form. In addition, the user can execute a series of queries to gauge data (using spatio-temporal operators or not) and retrieve the corresponding results. With its role as a repository of moving object trajectories, HERMES is used as a main data source in a series of individual applications as well, serving specific purposes, for example data loading software, data mining software, data visualization software, etc. (Figure 1-4). According to the

current system's implementation, it directly rises a need for formatting the data exchanged between the repository and its various applications, using a commonly acceptable standard, with the ultimate objective of creating a more flexible and expandable system.

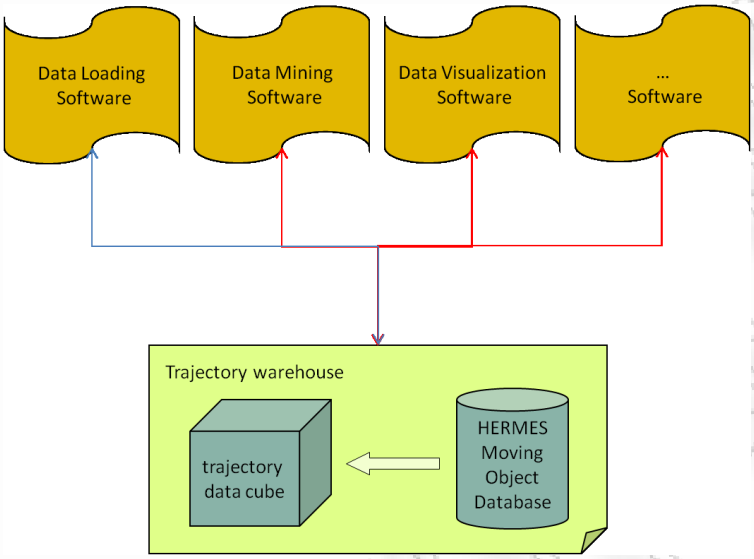


Figure 1-4: HERMES-based Trajectory Data Warehouse

With the main objective of developing a more independent and flexible system, it is necessary to achieve a higher level of processing both incoming data and outgoing results, with the use of GML standard. Based on the latest version of this standard (3.1.1), we are not limited in representing statically geographic concepts, but also moving objects. The proposal builds on the idea of implementing an additional mechanism, which will be deployed between HERMES and each application (Figure 1-5) and will be able to:

1. Provide a default GML standard, which will be used by the user for sending spatial and temporal queries to the database management system; these queries will then be converted into SQL queries, so that they can be further processed and
2. Receive SQL results before they are sent back to the user and transform them into GML-based structures, giving thus to the end user a self-described set of results in relation to the query which performed.

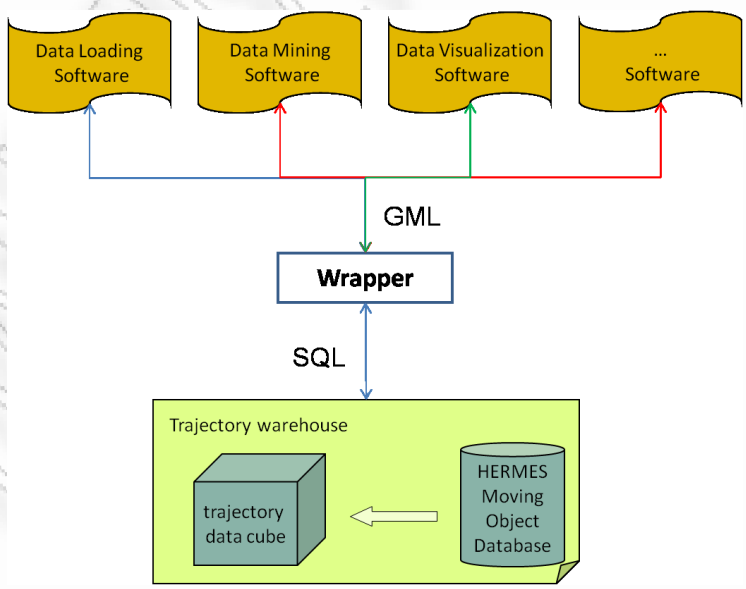


Figure 1-5: Wrapper extension on HERMES-based Trajectory Data Warehouse

Given that the GML standard is able to describe both spatial and non-spatial concepts, it becomes an ideal solution for creating trajectories of moving objects (using *LineStrings* in its most basic form), and also to accompany such data with other non-spatial information, which are necessary for describing each

item. Furthermore, the fact that the GML standard is based on XML grammar for data formatting makes possible the conversion of GML data in any XML-based visualization format, such as Vector Markup Language - VML (Mathews, et al., 1998), KML etc. This process may be accomplished in either the server's domain or the client's one. In this way, an immediate visualization of results returned from HERMES is achieved, via third-party projection engines, as Microsoft Virtual Earth (Microsoft Virtual Earth, 2008), Google Maps (Google Maps, 2008), Google Earth (Google Earth, 2008) etc.

1.3 Research objectives

The main objective of this thesis is to explore how accompanying representation technologies, such as XML, GML and KML can be integrated into a unified wrapper, which will provide HERMES with this ability to operate in a more interoperable fashion, while exchanging data. Another goal is to study the use of GML-based queries meeting a predefined application schema.

Based in the objectives above, the following research questions have to be defined:

- How is GML data structured?
- How can GML data be visualized?
- How can XSLT be used to transform GML data into KML?
- How is GML data converted into KML format?
- How all the already mentioned technologies can be implemented in the nature of an interfering mechanism between a spatio-temporal DBMS as HERMES and an end user?
- How this mechanism can be deployed in a client-server environment, such as the Internet?

1.4 Methodology

This thesis starts with a brief overview about the evolution of spatial databases and the accompanying temporal extensions of them. Thereafter, relevant literature on XML and XML-based technologies such as GML, KML and XSLT are reviewed. Finally, the design and implementation of a mechanism, in the nature of a web application, capable to act as a wrapper between these two worlds is studied. As a proof of evidence, a case study with real life spatio-temporal data are their capabilities to be represented in different formats is accomplished.

Chapter 1 of this thesis describes the research background and motivation, problem definition, research objective and issues, methodology and the thesis outline. In Chapter 2 we review MOD concepts, their evolution and describe HERMES-MDC extension. Chapter 3 is mainly focused on XML and related technologies. Geographic data representation in GML is described in Chapter 4. The 5th chapter provides information on what KML is and how it can be used to visualize GML data. Chapter 6 examines what XSLT is and how to use it in transforming XML-based data (GML) to another visualization XML-based format (KML). Chapter 7's goal is to provide information about a prototype's design and implementation for transforming raw data into GML and then visualize them, using KML transformations and projection engines. The material in Chapter 8 builds upon the overall system's understanding by covering the prototype's querying methods and transformation results based upon a real life dataset. Finally, in Chapter 9 this thesis conclusions and possible future work are presented.

2 Moving Object Databases

2.1 Introduction

This chapter examines various moving object databases concepts. Conceptual definitions and evolution information on spatial and spatio-temporal databases are provided. Finally, a brief description of HERMES-MDC concludes the chapter.

2.2 Mobility scenarios

Any physical object's existence brings naturally and automatically with it that, at any point in time, it is located somewhere. In the dynamic world in which we live, space is a property that varies over time. Space and time have become such indispensable elements of human beings' daily life, that we almost never think about them, and sense them while having difficulty in describing them.

In a rapidly evolving world, mobility is an important factor of people's life. The Internet, wireless networks, positioning technology as well as personal devices such as PDAs and cell phones and their related services are being advanced and improved day by day. Over the past few years, rapid advances in miniaturization and personalization of electronic devices have taken place, which consequently have resulted in major price reductions. Performance improvement of general computing technologies on the other hand has made it possible to introduce services that previously were even impossible to think of. The ultimate goal of all these advances is to satisfy the consumers' rising expectations. This can be achieved if information can be timely provided in the right place. In the coming years, delivering appropriate timely (personalized) services based on the position of mobile consumers will become increasingly important. Provision of such information will benefit the consumers in various ways, for instance, in better awareness of their surroundings, in identifying potential problems and bottlenecks, which in turn helps them to more efficiently and accurately plan to tackle them, in better management of available resources and planning for possibly sharing them for efficiency reasons.

Despite some success in fulfilling consumers' requirements, there is still a long way to go and new serious challenges are ahead. One of these challenges is the lack of database support at present. This stems from the fact that existing databases, which are one of the key elements in making more practical and accurate information available, are at their best good in handling static situations, while the concept of mobility brings up a new set of requirements, dealing with dynamic situations.

The central issue in any mobility scenario is the object whose position continuously changes, i.e., the moving object. Although the concept of moving object is rather new in the area of spatio-temporal databases, the variety of applications that may benefit from it is enormous. Urban traffic, especially commuter traffic, and rush hour analysis; fleet management and car theft protection; monitoring animal migration; analysis of shopping behavior (in a mall or city center); patient tracking in a hospital; location-based services, such as tourist information, localized advertising, emergency services; these are just a few examples to mention. The potential is simply enormous.

2.3 Capabilities of spatio-temporal databases

In a world facing information explosion, positioning technology is rapidly making its way into the consumer market, not only through the already ubiquitous cell phone but soon also through small, on-board devices in many means of transport and in types of portable equipment. It is thus to be expected that all these devices will start to generate an unprecedented data stream of time-stamped positions for the agents that carry them. This development does not depend on GPS technology alone: in-house tracking technology applies various techniques for up-to-date positional awareness, and adaptable antenna arrays can do accurate positioning on information obtained from calls by cell phones (Cooper, 2003).

Thanks to these advances in positioning technology, which makes data about moving objects easily available, soon these objects have become one of the focuses of the spatio-temporal database community. However, in spite of its simple looks, the moving object concept has become a practical challenge in applications dealing with mobility, Internet technology, and Geographical Information Systems (GIS).

Databases have not very well accommodated moving object data in the past, as their design paradigm was always one of *snapshot representation*. Their present support for spatial time series is at best rudimentary. Consequently, database support for moving object representation and computing has become an active research domain; see for instance (Abdelguerfi, et al., 2002), (Zhu, et al., 2002), (Guting, et al., 2000), (Saltenis, et al., 2000) and (Agarwal, et al., 2002).

Database management systems (DBMSs) have a potential foundation for moving object applications; however, they are currently not used for this purpose and at the moment the aforementioned moving object application domains lack database support. The reason is that moving object databases require a set of critical functionalities to be integrated, and built on top of existing DBMSs (Wolfson, et al., 1999).

What is needed in current real-world spatio-temporal applications is a small, robust, and highly expressive set of predicates, suitable for implementation based on off-the-shelf DBMS technology. The query processing schemes for the predicates and the accompanying indexing schemes should be supported by the implementation (Vazirgiannis, et al., 2001). Over the years, various issues were identified as a set of capabilities that should be provided by a DBMS to effectively and efficiently support mobility and moving objects. Despite their individual nature, these required capabilities are somehow related and any improvement or problem in one will affect the rest.

Following is an enumeration of important challenges that current DBMSs are facing, concerning moving objects:

1. Data modeling and representation

   Data modeling aims at defining the data types, operations, and relations between them (Guting, 1994) to support application design. In other words, data modeling is the common name for the design effort of structuring a database.

   This process involves the identification of the kinds of data that will be stored in the database, as well as the relationship among these data types (ITC Educational Textbook Series, 2001). The requirements of moving object modeling are not fully covered by current data models. We illustrate this below.

   Regarding data modeling, an important question is how to represent a moving object. The efficiency of indexing and query processing methods is highly affected by the chosen method to represent the continuous nature of the moving object. Since computer systems cannot easily represent continuous phenomena, such phenomena must be approximated using finite structures.

   The approximation methods should faithfully represent the object movement and provide a basis for further analysis, especially because an inappropriate technique will increase the uncertainty. The data model has direct effects on storage space, performance, and access time. The data model defined for modeling continuously changing locations should be simple, though expressive, and be easy to implement. The more expressive a model is, the closer to the real world the application will be, and the more semantics will be captured. However, the more expressive a data model, the more complex it may be (Tryfona, et al., 1997). Furthermore, since the moving object field of research is young, there is room for new concepts and techniques. Therefore, the data model should allow for possible extensions. On the other hand, the possibility of designing new data models based on already existing ones should be investigated.

   In addition, the moving object concept brings a new dimension to the definition of operations defined in traditional data models. For instance, the traditional definition of distance between two objects often concerns the Euclidean distance. However, distance in moving object scenario is a time dependent function, rather than a constant value. On the other hand, for network-constrained moving objects the use of plain Euclidean distance is not advisable, since the underlying network poses extra conditions on maneuverability of objects. This means that either existing operations should be revised and adapted to accommodate the moving object concept or new operations should be defined. In addition to the basic spatial and temporal data types and operations, which are supported by traditional data models, extra spatio-temporal data types and operations are needed. However, the question is what these extra data types and operations are, to fully support moving objects. This is an important issue since more powerful data models are the ones with more complete and expressive data types and operators and have the better strategy to reach the closure under the defined operations set. Furthermore, there is the issue of integrating such data types and operators with the DBMS. Data types can be integrated with the DBMS in three different ways, which from tightest to loosest are as follows: integration of data

types into the DBMS kernel, using a database extension, and implementing data types as a layer on top of DBMS. The choice of integration method is a crucial one.

2. Query processing

   Most existing query languages are non-temporal and limited to accessing a single database state (Deng, et al., 2002). Traditional query languages such as SQL were not designed for querying time-varying spatial aspects. Processing moving object data requires new sets of spatial, temporal, and spatio-temporal operators to be used in query processing. These new operators should be applicable for any kind of object, i.e., those with free movements as well as movement constrained objects. Movement of an object is either constrained by other objects or by the medium via which it is travelling. The question is how to design a query system on top of an existing query system to deal with the dynamic aspects of moving object data. Furthermore, moving object applications often have to use different databases to answer queries. This means that the query processing technique should account for delay, overhead, and inaccuracy (Wolfson, 2002).

   On the other hand, the mobility of objects leads to the invalidity of query answers, simply because some or all of the spatial and temporal criteria that were true at the time of posing the query will be violated very frequently. Therefore, the posed query should be reprocessed every now and then. Therefore, the question of when and how often the query should be re-evaluated arises.

3. Indexing

   Moving object databases often have huge amounts of data. Therefore, examining the location of each moving object in the database to answer queries results in high performance overhead. Thus, the location attribute should be indexed. However, straightforward use of spatial indexing is not feasible due to the fact that continuous change of the locations implies that the spatial index has to be continuously updated (Wolfson, et al., 1998). Constant updating of the indices is not feasible if not impossible due to huge computing resources required (Jensen, et al., 2001).

   Therefore, spatio-temporal indexing techniques are required. Previous work on indexing spatio-temporal data concerns either past or present and future data. However, most of these approaches deal with spatial data changing discretely over time (Pfoser, et al., 2003). Therefore, an important question is how to index the continuously changing moving object data with sufficed performance and acceptable cost.

4. Uncertainty handling

   The locations of moving objects are inherently imprecise (Pfoser, et al., 1999). This inherent uncertainty has various complications for database modeling, querying and indexing. The more accurate the record of moving objects, the better query results. However, this in turn may result in poor query performance. Therefore, a moving object data model must properly represent moving objects with reasonable degree of precision, which does not harm query performance (Jensen, 2002). On the other hand, moving object applications may need query-imprecision support, due to imprecision associated with other notions, (e.g., traffic jam) used in the query definition (Wolfson, 2002). One of the main research questions in this direction is how to build up new modeling and spatio-temporal capabilities needed for moving objects to handle the inherently imprecise data and their related query analysis, considering the fact that lowering uncertainty is costly.

5. Data mining and prediction

   One important database challenge is to find valuable information hiding in large amounts of data, such as moving object data. Moving objects often have repeated patterns of movement, which can be used for planning and management purposes. The objects' movement can have periodically repeated patterns, e.g., animal migration patterns, commuters, shopping patterns, or sudden patterns. Identifying both these patterns is the key for successful planning in the objects' environment. On the other hand, this identification can be used for classification of objects and consequently, providing appropriate services to objects, which share some profiles. However, due to the multidimensional nature of moving object data and dependency of its data to other objects as well as the underlying network conditions, if applicable, current data mining methods used in databases are not suitable for moving objects (Jensen, 2002).

6. Keeping information up-to-date

It is often assumed in existing databases that data does not change unless it is explicitly modified (Wolfson, et al., 1998). However, in mobility scenarios the moving object's location continuously changes even if the database is not directly updated. Due to continuously changing nature of moving object data, keeping such data up-to-date is a must. Continuous update of the database is impractical since the location is updated very frequently. On the other hand, the answer of queries may be outdated if data is not properly updated. Furthermore, assuming that the moving objects themselves are responsible for transmitting location-oriented data updates via wireless networks, frequent updating would also impose a serious bandwidth overhead (Wolfson, et al., 1999). In addition, strategies are needed to handle possibility that a moving object becomes disconnected and cannot send updates, which results in incomplete and inaccurate data set. This will bring the attention to an important issue of how to deal with the trade-off between the updating overhead and incomplete and inaccurate data set. One also should pay attention to the fact that too few updates leads to information loss and too frequent updates gives rise to storage and transmission issues. In short, due to continuously changing nature of moving object data, keeping such data up-to-date is a must.

7. Efficient storage mechanism

   Moving object applications have many objects to monitor, the monitoring process may be continuous, and the respective data acquisition rates may be high. Thus, an efficient storage mechanism is required. On the other hand, since not all the acquired data may be necessarily informative, to avoid wasting storage space, some compression mechanisms should be utilized.

8. Visualization

   Graphical visualization is a strong power in moving object applications. However, considering the continuous change of the object data, there is a great concern on how query answer can be visualized. Another question is what dimension to use for the visualization purpose. Considering a 3D environment increases the moving object challenges in all aspects. Working in a 3D environment is not only 3D visualization. Obviously it needs spatio-temporal analysis of the 3D data types, which are not supported in the existing DBMSs. On the other hand, mobile consumers want to see their graphic display updated all the time, with local information.

2.4 Evolution of spatio-temporal models

For quite some time, members of the database community considered spatial databases and temporal databases to be new trends. Many database scientists identified and solved related problems. Spatial database research focused on modeling, querying, and integrating geometric and topological information in databases. Temporal database research concentrated on modeling, querying, and recording the evolution of facts under different notions of time and, thus, on extending the knowledge stored in databases about the current and past states of the real world (Erwig, et al., 1997). Despite the many great results achieved, the satisfaction of database community did not last long. It soon realized that in reality, space and time are rarely, if at all, independent. Soon, new efforts were directed towards integrating both concepts in one database. Both the spatial database community and the temporal database community individually tried to extend their databases to support the missing concept. Integration of space and time means dealing with time-varying geometries, which soon became the main focus of a new branch of database, i.e., the spatio-temporal database. Looking at history of spatio-temporal data models reveals how this branch of databases has evolved (Pelekis, et al., 2004).

By December 1997, the first attempt to integrate space and time into a relational database was reported by Tryfona & Hadzilacos. The modeling requirements of spatio-temporal applications at the logical level of design were discussed and the essential elements of a spatio-temporal application and the interconnections among them were represented. After identifying key features required for handling spatio-temporal phenomena that are currently lacking in relational data models, they proposed an extension of the relational data model, called the Spatio-Temporal Relational Model (STRM), providing a small set of representation constructs (Tryfona, et al., 1997).

In 1999, Wolfson and his colleagues identified a set of functions that are needed to handle moving objects and consequently they proposed a data model to support such capabilities. In their proposed prototype, called DOMINO, they aimed at integrating these required functionalities in a layer on top of existing DBMSs. They introduced a system architecture that consists of three levels. The first level is an object-relational DBMS, which stores moving object data in a form of a sequence of time-stamped

positions. The second level is a GIS that is responsible for storing, querying, and manipulating spatial objects. The third layer, DOMINO, contains temporal predicates and offers support for inherent uncertainty of moving object data (Wolfson, et al., 1999). A comprehensive approach to integrate these supports in a commercial DBMS was also proposed (Wolfson, et al., 1998). Later, the data model was modified to also support uncertainty of moving object data and deviation between a moving object's actual location and its location as stored in the database. In the new data model, which assumed constrained movements on predefined networks, point objects were either mobile or stationary. If the object is stationary, its location attribute is an (x, y) coordinate pair. However, if the object is mobile, its location attribute has six sub-attributes, namely, the pointer to a line object representing the network segment on which an object is moving, location and time at which the object started its movement, the direction in which the object travelled, the (presumed constant) speed at which the object travelled, and, finally, an uncertainty measure, which could have been either constant or a function of time, representing the threshold of the location deviation. Another contribution of this work was a probabilistic model and an algorithm for query processing. In this model the location of a moving object is a random variable. The density function of this variable is determined using object location at any point in time and uncertainty derived from the database (Wolfson, et al., 1999).

Pelekis et. al. in 2002 presented an integrated and comprehensive design of spatio-temporal data types in the form of an Oracle Data Cartridge (Oracle Corp., 2003). HERMES Moving Data Cartridge (HERMES-MDC) (Pelekis, et al., 2006) and (Oracle Corp., 2003), integrates two other data cartridges, namely the TAU temporal data cartridge and Oracle's spatial data cartridge. It introduces time-varying geometries that change their position and/or extent in space and time dimension, either discretely or continuously. HERMES-MDC extends PL/SQL, the data definition and manipulation language of Oracle10g, with spatio-temporal semantics. The current thesis is based on HERMES-MDC and makes extensible use of it.

2.5 Spatial change in Moving Object Databases

We have seen that many developments in spatio-temporal data models are aimed at supporting the representation of spatial change over time. Two types of spatial change may be distinguished, namely discrete change and continuous change. Cadastral applications are well-known examples of the former, in which discrete changes are relatively easy to keep track of in a database. This can be achieved by frequently updating the database and recording history (Guting, et al., 2000). However, with continuous change, it is not feasible to constantly update the database for each such change.

The essence of spatio-temporal data models is to accommodate continuous change over time. Although this phrase almost immediately brings the terms moving object and movement to mind, it was only last years that researchers came up with formal definitions of both terms in context of spatial data handling.

A moving object is an object whose position and/or extent changes over time (Erwig, et al., 1999). The focus of this work is on moving point objects. Therefore, from now on whenever it is referred to a moving object, a moving point object is considered, unless it is mentioned otherwise. Any change in location will be viewed as movement. Simply speaking, movement is a mapping of time into space, indicating location at different points in time. The sequence of time-stamped locations visited by a moving object, form that object's trajectory. A trajectory is an ordered historic trace of locations of a moving object that can be depicted (simplified) as a line, however, the temporal characteristics are important semantic parameter in definition of trajectory. In fact, a trajectory represents the path taken by an object together with the time instants at which the object was at every position along the path (Vazirgiannis, et al., 2001).

In contrast to static objects, moving objects are difficult to represent in a database. Currently, applications dealing with moving objects are being developed in an ad hoc fashion. Despite of all the work on databases, situations in which the whereabouts of objects are constantly monitored and stored for future analysis are an important class of problems that present-day database users will find hard to tackle satisfactorily with their systems. The reason is that special functions needed by such applications are currently lacking. Therefore, there is an essential set of functions that has to be integrated, and built on top of existing DBMSs to support moving objects (Wolfson, et al., 1999). Such functions constitute a wide domain ranging from data models, data structure and operations, indexing methods, query processing techniques, and visualization methods that can handle continuous change in moving objects

and their large amounts of data. Applications dealing with time-varying data can be classified in one of the following categories (Tryfona, et al., 1997), based on the type of change they accommodate:

- Applications that are concerned with changes of non-spatial characteristics of objects, e.g., land parcels in a cadastral information system,
- Applications in which the position of objects continuously changes, e.g., cars moving in a road network and
- Applications with objects that integrate changes in the above case as well as changes in their extent. This case mostly happens in environmental applications, e.g., monitoring water pollution caused by oil spills.

While the first category deals with rather discrete phenomena, the other categories handle continuous change. Since our focus is on continuous change of object positions, here a fundamental issue arises, namely the *medium* via which the objects are thought to travel. That medium may or may not impose restrictions on movement. This results in at least three scenarios of object movement:

- Free movement in 2D or 3D space, e.g., animal migration,
- Restricted movement in 2D or 3D space, e.g., ships along coastlines and
- Restricted movement on 2D or 3D networks, e.g., car movements.

Obviously, there are scenarios, in which movements occur in combination, for instance, movements in shopping malls. Although these can be seen on the one hand as free movements, since people can freely move in space in any desired direction and any manner, on the other hand their movements are still restricted to the corridors and spaces between shelves. Another important observation is that situations in which objects are stationary are special cases of movement and should not be ignored. Stationary situations may occur due to physical obstacles (accidents, traffic), permanent constraints on the movement (stops at traffic lights), or personal decisions (waiting for someone in a parked car).

2.6 Trajectory Representation

All points, which are traversed by a moving object, make up the trajectory of that object.

From users' viewpoint, the concept of trajectory is rooted in the evolving position of some object moving in some space during a given time interval. Thus, trajectory is by definition a spatio-temporal concept. But while *moving* may be seen as a characteristic of some objects that differentiates them from non-moving objects (i.e. buildings, roads etc.), the concept of moving object implies that its movement is intended to fulfill a meaningful goal that requires moving from one place to another. Moving for achieving a goal takes a finite amount of time (and covers some distance in space); therefore trajectories are inherently defined by a time interval. This time interval is delimited by the instant when the object starts a movement (called *begin*) and the instant when the movement terminates (*end*). Identifying *begin* and *end* within the whole time-frame where the object is moving is an application decision, i.e. a user-driven specification. So, we can express the following definition, which formally defines a moving point trajectory in a database perspective.

A trajectory is the user defined record of the evolution of the position (perceived as a point) of an object that is moving in space during a given time interval in order to achieve a given goal. (Damiani, et al., 2007).

$$\text{trajectory} : [\mathit{begin}, \mathit{end}] \rightarrow \text{space}$$

The definition above settles trajectories as semantic objects. The time space function is defined by the user and is not necessarily the one provided by the data acquisition mechanism. The latter is the raw data, whose form usually is as a sequence of (sample point/time) pairs (Figure 2-1). Raw data often needs to undergo a cleaning process to correct errors and approximations in data acquisition. In addition, the application may be interested in only a subset of the cleaned sample points, e.g. skipping points acquired during the night to only retain daylight movement or replacing a sequence of irrelevant (from the application perspective) points with a single representative point (e.g. for representing stops as a single point).

The original sense of the term trajectory denotes the changing position of an object in geographical space, be it a 3D space (e.g. the trajectory of a plane) or a 2D space (e.g. the trajectory of a rolling ball in a bowling game). We say a trajectory is spatio-temporal if spatial coordinates are used to express the position of the traveling object. Most frequently, the moving object is geometrically represented as a point

(e.g., a person, an animal, a car, a truck, a plane, a ship, a train). Yet the moving object may have a surface or volume geometry (e.g. clouds, floods, air pollutions, oil spills, avalanches), in which case both change in position and change in shape may concur to define the trajectory. In the current thesis we only consider modeling spatio-temporal trajectories generated by objects represented as points.

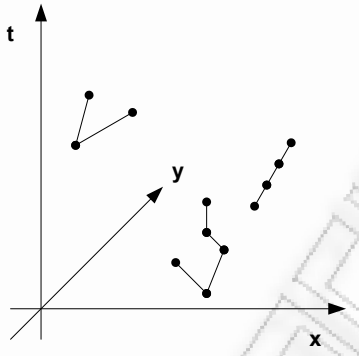


Figure 2-1: Spatio-temporal path for ongoing moving objects as a set of points

A trajectory has two facets:

- *The geometric facet*: This is the spatio-temporal recording of the position of the moving point. It is a delimited segment (i.e., a single continuous subset) of the spatio-temporal path covered by the object's position during the whole lifespan of the object. From the conceptual modeling perspective, we can basically rely on the definition given above and represent the geometric facet as a continuous function from a given time interval into a geographical space (the range of the function): trajectory: [*begin*, *end*] → space.

  However, the modeling structure should also include the sample points (and the interpolation functions) that are used to discretely capture the trajectory function. The geometric facet could be modeled using the moving point data type.
- *The semantic facet*: This is the information that conveys the application-oriented meaning of the trajectory and its related characteristics.

2.7 Spatial Database Management Systems

In various fields, there is a need to manage data related to space. The space of interest can be, for example, the 2-D abstraction of (parts of) the earth's surface (i.e., geographic space, the most prominent example). Other examples are a man-made space (e.g., the layout of a Very Large Scale Integration - VLSI design), a volume containing a model of the human brain, or another 3-D space representing the arrangement of chains of protein molecules. At least since the advent of relational database systems, there have been attempts to manage such data in database systems. Characteristic for the technology emerging to address these needs is the capability to deal with large collections of relatively simple geometric objects, for example, a set of 100,000 polygons. This is somewhat different from CAD databases (e.g., solid modeling) where geometric entities are composed hierarchically into complex structures, although the issues are certainly related (Hilton, 2007). A simplified architecture of current spatial databases is presented in Figure 2-2.

Oracle Spatial is one of the most powerful spatial DBMSs on the market. Oracle series began to support spatial data in its option Oracle Spatial since Oracle 8i. Partially compliant with Simple Features Specification for SQL (Oracle Corp., 2003), Oracle Spatial supports several spatial types specified in SFS. Oracle Spatial is considered partially compliant with OGC specifications but not completely compliant because in Oracle Spatial there are no separate data types for *point*, *linestring*, *polygon*, etc., but there is uniform data type: *SDO_GEOMETRY* to represent all spatial data types. Beside spatial data types, a large number of spatial functions are available in Oracle Spatial as well.

Unlike other spatial DBMSs in which different spatial types represent different geometries, there's only one spatial type: *SDO_GEOMETRY*. Before explaining the geometry types that can be represented with *SDO_GEOMETRY*, the Entity Relationship diagram in Figure 2-3 would be helpful to understand how *SDO_GEOMETRY* works.

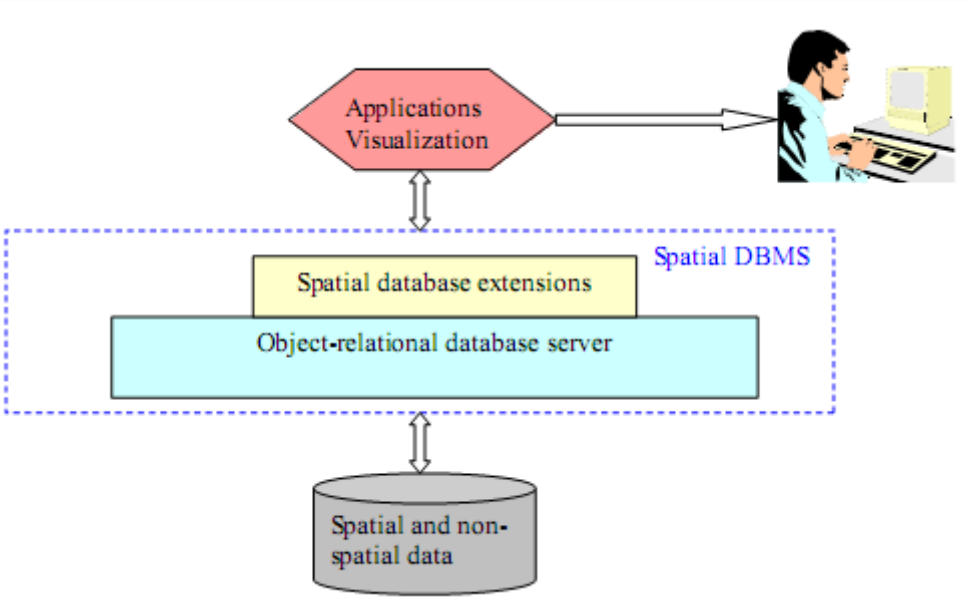


Figure 2-2: Simplified architecture of spatial databases

Oracle enables users to define new data types (user-defined data types) which are made up of several attributes. These attributes can be of basic data types such as numbers, varchar2, date, or other existing user-defined data types. *SDO_GEOMETRY* is created as a user-defined data type. Several kinds of geometries can be represented by setting attributes of *SDO_GEOMETRY*.

Oracle Spatial support a large number of operators and functions for spatial data types. Spatial operators in Oracle Spatial provide optimum performance because they use the spatial index, which is an *R-tree* (Guttman, 1984) organizing *MBRs* (Minimum Bounding Rectangles) of geometry shapes, on spatial columns. Spatial operators must be used in the *WHERE* clause of a query. Spatial functions in Oracle Spatial differ from spatial operators in that they do not require that a spatial index be defined, and they do not use a spatial index if it is defined. These spatial functions can be used in the *WHERE* clause or in a sub-query. Both spatial operators and spatial functions are included in the Spatial PL/SQL Application Programming Interface (API).

There are several spatial DBMSs on the market, such as Ingres, Informix, PostGIS, and Oracle Spatial. All of them have spatial data types and spatial functions, but the differences in their spatial functionalities are significant, which can be revealed from the following comparison. Oracle Spatial is also included in this comparison as a reference spatial DBMS:

- Informix (IBM, 2008), PostGIS (PostGIS, 2008) and Oracle Spatial (Oracle Corp., 2008) support OGC specifications (OGC, 2008), whereas Ingres (Ingres, 2008) does not,
- The spatial types can only be 2D in Ingres, but they can be 3D or 4D (with measure values) in other ones,
- Supported spatial types are different. Oracle Spatial supports most of the data types specified in Simple Features Schema – SFS (OGC, 2008), and unknown geometry types can be stored; PostGIS supports the same spatial data types as Oracle Spatial, but unknown geometries cannot be stored; Informix does not have Multi-Geometry type, which is a collection of different basic geometry types; Ingres does not support anyMulti- types, likeMultiPoint, MultiPolygon, etc. and
- Numbers of spatial functions are different. Oracle Spatial and PostGIS give more useful spatial functions than Informix and Ingres.

2.8 Spatio-temporal Extensions

The concept for spatial data managing is in many cases solved by current DBMSs via the corresponding off-the-shelf extensions. The same is not true as far as temporal, and more precisely spatio-temporal, data are concerned.

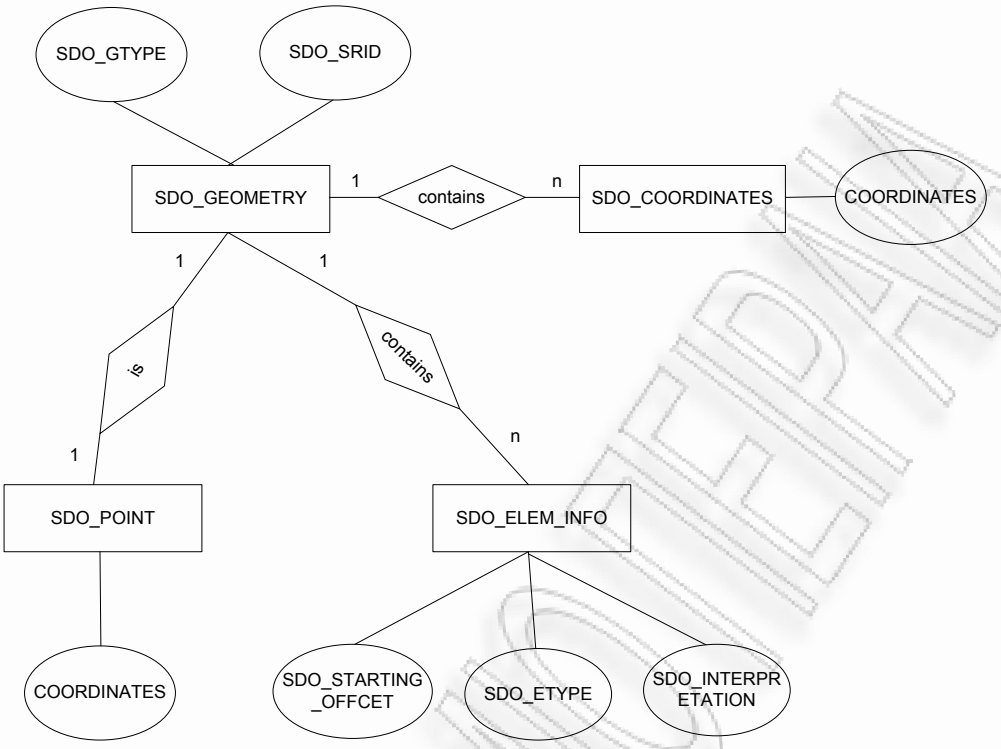


Figure 2-3: Entity-Relationship diagram for Oracle Spatial SDO_GEOMETRY (Kothuri, et al., 2007)

The approach can be integrated into an Oracle ORDBMS using database extender technology, which provides a mapping between the API of the index and query structure, and the underlying database tables and index structures. The need to provide RDBMS support for spatio-temporal data is a natural extension of the stance taken for purely spatial or purely temporal database support.

2.8.1 HERMES-MDC

In this direction, Pelekis et. al presented HERMES (Pelekis, et al., 2006). HERMES is developed as a system extension that provides spatio-temporal functionality to Oracle10g's ORDBMS. The system is designed in a way that it can be used either as a pure temporal or a pure spatial system, but its main functionality is to support the modeling and querying of continuously moving objects. Such a collection of data types and their corresponding operations are defined, developed and provided as an Oracle data cartridge. HERMES Moving Data Cartridge (HERMES-MDC) is the core component of the HERMES system architecture. HERMES-MDC provides the functionality to construct a set of moving, expanding and/or shrinking geometries, as well as time-varying base types. Each one of these moving objects is supplied with a set of methods that facilitate the cartridge user to query and analyze spatio-temporal data. Embedding this functionality offered by HERMES-MDC in Oracle's DML (Oracle Corp., 2008), one obtains an expressive and easy to use query language for moving objects.

In order to implement such a framework in the form of a data cartridge they exploit a set of standard data types together with the static spatial data types offered by the Spatial option of Oracle10g (Oracle Corp., 2008) and the temporal literal types introduced in a temporal data cartridge, called TAU Temporal Literal Library Data Cartridge (TAU-TLL) (Pelekis, et al., 2006). Based on these temporal and spatial object data types HERMES-MDC defines a series of moving object data types. The overall HERMES architecture can be seen in Figure 2-4.

A straightforward utilization scenario for a HERMES-MDC user is to design and construct a spatiotemporal object-relational database schema and build an application by transacting with this database. In this case, where the underlying ORDBMS is Oracle10g, in order to specify the database schema, the database designer writes scripts in the syntax of the Data Definition Language (DDL), which is the PL/SQL, extended with the spatiotemporal operations previously mentioned.

To build an application on top of such a database for creating objects, querying data and manipulating information, the application developer writes a source program in *Java/C#* wherein she can embed

PL/SQL scripts that invoke object constructors and methods from HERMES-MDC. The *JDBC pre-processor* integrates the power of the programming language with the database functionality offered by the extended PL/SQL and together with the *ORDBMS Runtime Library* generate the application's executable. By writing independent stored procedures that take advantage of HERMES functionality and by compiling them with the *PL/SQL Compiler*, is another way to build a spatio-temporal application (Pelekis, et al., 2006).

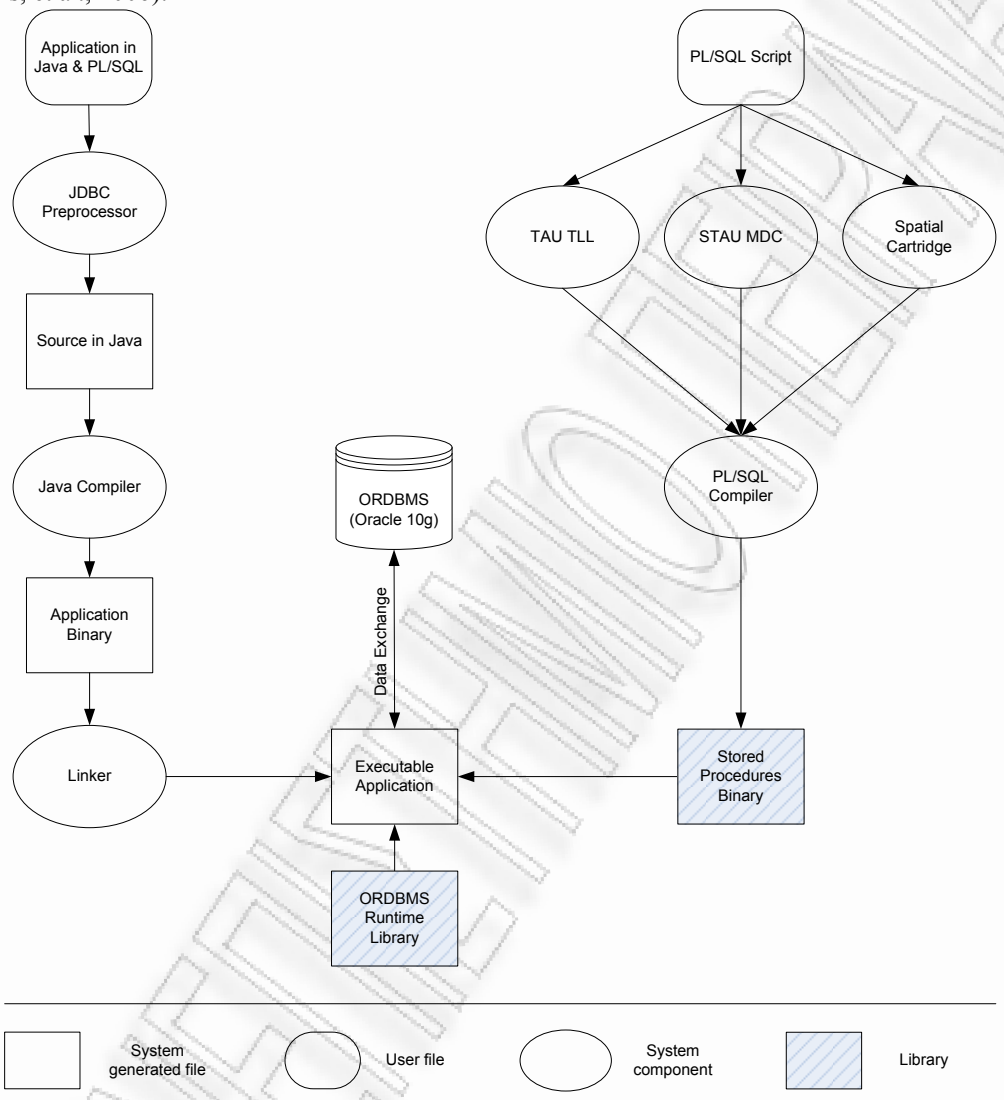


Figure 2-4: HERMES system architecture (Pelekis, et al., 2006)

2.9 Conclusion

Geographical data glut modern DBMSs every day. It seems, moreover, the importance they have placed in this field of supporting large vendors in their systems. Nevertheless, spatial, in conjunction with temporal data is not yet supported directly by current DBMSs. Thus, custom extensions are being developed, such as the HERMES-MDC.

3 The eXtensible Markup Language (XML)

3.1 Introduction

This chapter presents Extensible Markup Language (XML), its advantages and the basics behind it. It also describes the XML structure, and the relation with Document Type Definition, Schema and Document Object Model which are common in any XML application.

3.2 Evolution of XML

Structured content in general needs to be represented if it is to be used for machine-based processing. This means that there must be a formal and machine-readable way of representing document structures. Basically, a markup language is a text-based way of mixing structural information (the markup) and content, and the rules for how this is done are defined by every language, or by a framework which enables users to define their own structures.

The most prominent languages for defining own structures are presented, which are the Standard Generalized Markup Language (SGML) and the Extensible Markup Language (XML). Even though these languages are the most popular ones, there are also some non-markup approaches to representing structured documents.

The idea of structural markup (marking up content structures rather than layout information) was first presented by William W. Tunnicliffe at a conference in 1967. GML extended Tunnicliffe's general ideas with a well-defined nesting scheme of document structures, and the idea of document types, which define the allowed structures for a class of documents.

Interestingly, what later became known and successful as GML was called Text Description Language first (as an internal name), and this still captures the essence of what markup languages are about: describing a text's contents on the structural level, making it possible to have different applications using these descriptions for application-specific purposes.

Continuing the work on GML and joining forces with Tunnicliffe and Reid, the GML team moved on to develop the Standard Generalized Markup Language – SGML (International Organization for Standarization, 1986), which became the most important markup language and an important foundation for many document processing environments, because it was a standard rather than a product. This meant that using SGML, customers would not be bound to a specific software or vendor, and (at least theoretically) it would be possible for everybody to implement an SGML system by simply implementing the ISO standard.

However, even though SGML has had some problems with complexity and interoperability, it was the single most successful format for structured documents, and big organizations started recognizing the importance to store their documents in a well-defined and reusable way. For example, the U.S. Army required contractors to submit their documentation in SGML, which made it reusable and independent from any specific software or vendor.

When marking up document content, the question is what structural concepts do exist, and how can they be combined. For example, if there are sections which consist of headings and paragraphs or lists, would it also be allowed to use paragraphs within a list? These questions are addressed with SGML's concept of a Document Type Definition (DTD), a definition of a class of documents which adhere to a well-defined set of rules. Using this concept, first a DTD is designed, and then instances (i.e., actual documents) of this DTD can be created, which must adhere to the rules defined in the DTD. Seen this way, SGML is not a markup language in itself, but a mechanism to define vocabularies for marked up documents. This is shown in Figure 3-1, which also shows some of the more recent relatives of SGML.

In the beginning of the 90s, markup languages became the driving force of the globalization of the Internet with the Hypertext Markup Language – HTML (Raggett, et al., 1999), the content language of the World Wide Web (WWW). In its first versions, HTML was more inspired by SGML than being based on it formally, but this was changed later and HTML became a document type of SGML, which in SGML terminology is called an *SGML application*.

HTML may not be perfect from the markup design point of view, but its simplicity makes it easy to create. HTML's balance of simplicity and expressive power obviously was a one of the major reasons of the Web's success, and even though many HTML creators today do not know anything about markup languages and their concepts, they use an SGML-based markup language, as shown in Figure 3-1.

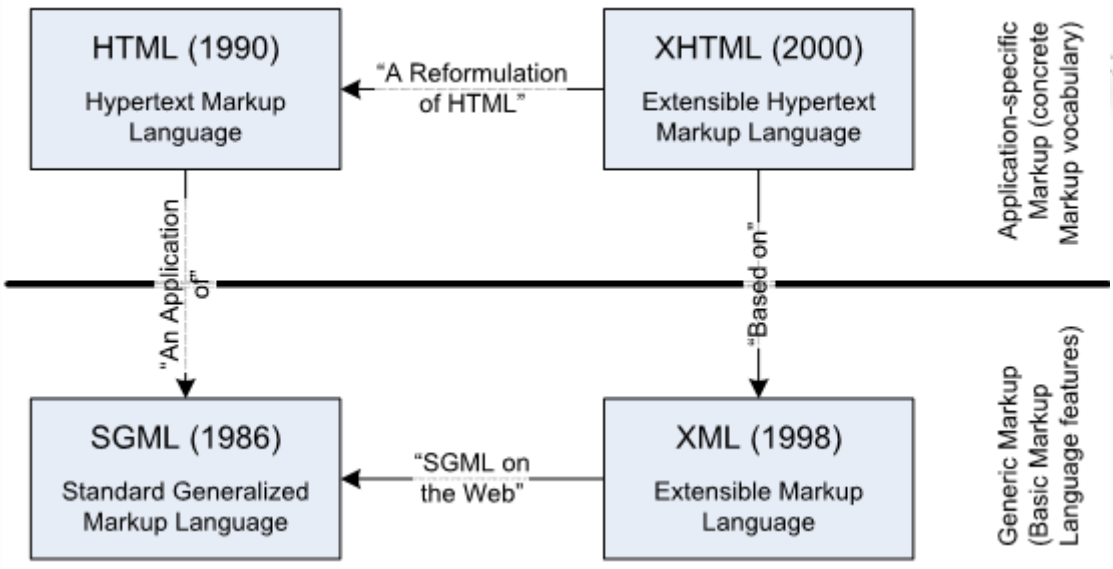


Figure 3-1: Markup language concepts

Even though HTML is well-suited for marking up documents for Web publishing, it is much less suited for marking up documents for general-purpose document processing. HTML basically is a rather simple page description language for Web pages, and as such has many limitations, such as missing support for paginated media, better layout control in general, or more detailed text structures apart from the simple model of paragraphs and simple lists. In the 90s, the Web grew at an astonishing rate, and there was a clear demand from content providers to have a better way of representing their content than just HTML. In particular, the trend towards mobile computing made it clear that contents should be kept in a device-independent way, and HTML would only be one way of publishing content (other possible channels being formats for mobile devices, or formats for print-quality publication).

XML supports the main concept of SGML, the Document Type Definition – DTD (Raggett, et al., 1999) which enables users to define their own classes of documents. However, XML leaves out many of the features that were thought of as being of only limited use. In fact, even though XML is radically simplistic in comparison to SGML, from today's point of view probably some more features would probably be left out, and a cleaner processing model would be defined.

However, even though XML may not have been the perfect markup language (in fact, it also is a generic markup language because it is a mechanism for defining vocabularies and not a markup language in itself), it has had a lot of success. Some of the reasons for this are political (it did not come from one of the major vendors and as such is free of any operating system or programming language legacy), and others are technical (the Internet needed a format for exchanging structured information). XML's success has been astonishing, and availability of a universally accepted format for exchanging structured information often vastly outweighs any minor concerns users might have with some details of the language.

It is interesting to note that even though XML was designed and thought of as a document structuring language (in the same way as SGML was intended for documents), the majority of XML users today are using XML for exchanging data rather than documents. Data in this case is more general than documents, in the sense that documents are a certain type of data characterized by properties such as instance variability and semi-structured models, while data can be anything, but in practice often are just well-defined structures such as tabular material for typical business-to-business applications exchanging business data such as orders and invoices.

The last building block shown in Figure 3-1 is the Extensible Hypertext Markup Language (XHTML) (Pemberton, 2002), which has been first defined in 2000 and revised in 2002. Because HTML is based on SGML, and XML is supposed to be the foundation for all machine-readable data in the future, the W3C

decided to create a new version of HTML. This new version is XHTML, and it is important to understand that XHTML does not add new features to HTML; it simply defines XHTML as being based on XML, so that XHTML documents can be processed using normal XML tools. This way, XHTML fits nicely into the overall picture of structured documents being XML documents, rather than using the older SGML syntax of HTML, which is much harder to process.

- Paragraphs without Emphasis:

```
<phtml>
    <p>This is the first paragraph.</p>
    <p>This is the second paragraph.</p>
</phtml>
```

- Paragraphs with illegal Emphasis (not proper Markup):

```
<phtml>
    <p>This is the <em>first paragraph.</p>
    <p>This is</em> the second paragraph.</p>
</phtml>
```

- Paragraphs with Tree-like Emphasis:

```
<phtml>
    <p>This is the <em>first paragraph.</em></p>
    <p><em>This is</em> the second paragraph.</p>
</phtml>
```

3.3 The eXtensible Markup Language (XML)

The XML 1.0 specification describes XML as a simple dialect (or *subset*) of SGML with the goal to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. For this reason, XML has been designed for ease of implementation, and for interoperability with both SGML and HTML.

XML is a markup language, designed to structure, store and send information over the Web. It provides no predefined tags, as in the case of HTML, but provides standards so that the user can define his own tags and document structure. Hence XML is free and extendible. Furthermore, as XML is in plain text format, it provides a software and hardware independent way of sharing data. It enables data to be accessed by all kind of *reading machines* or processors.

Since XML documents are in plain text and structured (encoded) with user defined tags, it does not do anything without some kind of software. Therefore it has to follow some standards in encoding data to enable decoding by some other programs. For this XML adheres to the standards specified by the XML specification. At present, XML Specification 1.0 (Forth Edition) is the W3C's implementation recommendation. XML documents must follow standard rules including the syntax for marking up and the meaning behind the markup. What a valid markup is defined by a Document Type Definition (DTD) or alternatively by an XML Schema.

3.3.1 XML document structure

There are certain rules that have to be adopted while authoring XML documents and these can be summarized as follows:

- XML documents need a declaration at the top to signal what they are;
- Every XML document must have a root element (tag) that encloses the content;
- Every start tag must have a closing tag;
- Tags must nest cleanly;
- Empty tags have a different form to make it clear that these are tags with no closing tags;
- All attribute values must be in quotation marks;
- Tags are case sensitive and must match.

The XML documents must be precise; those not complying with these rules cannot be processed by the XML parsers embedded in browsers or standalone processors. An XML document that conforms to these rules specified in XML specification, as determined by an XML parser is classified as well-formed. An XML Parser is a software program that creates the Document Object Model in the computer memory (Houldling, 2001).

An XML document mainly consists of two parts; prolog and body. The prolog contains the declaration, and the body contains the actual marked up document. The content of both parts (whole document) is composed of declarations, elements, comments, character references, and processing instruction, all of which are indicated in the document by explicit markup (W3Schools, 2008).

An example XML document is given below:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE MScThesis SYSTEM "Document.dtd">
<!-- An XML Example -->
<document>
    <title>VisualHERMES:Extending HERMES Interoperability</title>
    <pub_date>2008-05-20</pub_date>
    <production id="80" media="paper"/>
    <chapter>
        Introduction to XML
        <para>Introduction</para>
        <para>Evolution of XML</para>
        <para>The eXtensible Markup Language (XML)</para>
    </chapter>
    <chapter>
        XML syntax
        <para>Elements must have a closing tag</para>
        <para>Elements must be properly nest</para>
    </chapter>
</document>
```

3.3.2 Declarations

The first line of an XML document is a declaration which notifies that the document has been marked up as an XML document. The XML declaration itself is a processing instruction and therefore it begins with `<?` and ends with `?>`. The `version` attribute indicates the version of XML specification that the document complies with and the `standalone` attribute specifies whether the document has any markup declarations that are defined in a separate document. Thus, value `yes` implies no markup declarations in external documents and `no` leaves the issue open. The document may or may not access external documents. The `encoding` attribute denotes the character encoding system used in the document. The `DOCTYPE` declaration (second line in the example) declares the name, type and location of the related Document Type Definition (DTD).

3.3.3 Elements

Elements are the basic unit of XML content. An element consists of a start tag and end tag, and everything in between. Anything between a < sign and a > sign is a tag except that is inside a comment or a `CDATA` section. Relationships in XML elements are expressed in terms of parents and children. In the given example `document` is the root element. `Title`, `pub_date`, `production`, and `chapter` are child elements of `document` element and thus it is the parent element. `Title`, `pub_date`, `production`, and `chapter` are siblings (or sister elements) because they have the same parent.

Elements can have different content types such as elements that contains other elements like `document` element, mixed that contains both text and other elements as `chapter` element, simple that contains only text like `para` element in the example, or empty that contains no information like `production` element. In case of empty elements no closing tag appears.

The name of an element cannot contain space and must not start with a number or punctuation character or with the word `xml`. XML elements can have attributes in name/value pairs which provide additional information about elements and the attribute values must always be enclosed in either single or double quotes.

3.3.4 Comments

The comments are the character data in an XML document that XML processor ignores. The comments follow the syntax of `<!-- content -->`. The content of the comment should not have `-` or `--` characters that might confuse XML parser and also a comment should not be placed within a tag and cannot be nested.

3.3.5 Character references

Character data includes any legal character except `<` which is reserved for the start of a tag. XML provides a set of entity references that helps to avoid the ambiguity in specifying character data against markup. In XML, `>`, `<`, `&`, `"` and `'` characters can be substituted by `&gt`, `&lt`, `&amp`, `&quot`, and `&apos` respectively. `CDATA` blocks in XML provides a convenience measure to include large blocks of special character data. For example, a internal Cascading Style Sheet can be defined in `CDATA` block. `]]>` is not allowed within a `CDATA` block as it signals the end of a `CDATA` block.

3.3.6 Processing instructions

A processing instruction is a bit of information meant for the application using the XML document. That is, they are not really of interest to the XML parser. Instead, the instructions are passed intact, straight to the application using the parser. The application can then pass this on to another application or interpret itself. All processing instructions follow the generic format of `<?name_of_application instruction_is_for Instructions?>` and all processing instructions must be in lowercase. For example, `xml` declaration is a processing instruction, which name is `xml`.

3.4 Document Type Definition (DTD)

In XML the definition of a valid markup is handled by a Document Type Definition – DTD. It is a file (or several files to be used together) with `dtd` extension, written in XML's Declaration Syntax, which contains a formal description of a particular type of document. DTD sets out what names can be used for element types, where they may occur, and how they all fit together. For example the DTD of the above XML document is as follows:

```
<!ELEMENT Document (Chapter+ | ANY)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Pub_date (#PCDATA)>
<!ELEMENT Production (EMPTY)>
    <!ATTLIST Production
        id NMTOKEN #REQUIRED
        media CDATA #IMPLIED>
<!ELEMENT Chapter (Para+ | #PCDATA)>
<!ELEMENT Para (#PCDATA)>
```

In DTDs all keywords must be in uppercase, such as `ELEMENT`, `ATTLIST`, `#REQUIRED` etc. However user defined elements and attributes can be in any case the user chooses, as long as they are consistent. A DTD can be either included as part of a well-formed XML document

(standalone="yes"), or it can be referenced from an external source, (standalone="no"). When external DTD is referenced, the SYSTEM attribute whose value indicates the location of DTD has to be added to the DOCTYPE declaration. Thus, in order to reference an external DTD, both xml and DOCTYPE declarations have to be changed. An XML document that conforms to the rules of a DTD is called a *valid* document. A *valid* document is necessarily well-formed. When XML document is parsed by a processor, the DTD is being firstly parsed and then read the document to identify where every element type comes and how each relates to each other. Using a DTD when editing documents preserve them consistent and valid (W3Schools, 2008).

3.5 XML schema

XML Schemas is a W3C Recommendation for defining the structure, content and semantics of XML documents. It is an alternative to DTD written in Schema Definition Language, which is an XML language for describing and constraining the content of XML documents. In general terms, an XML Schema describes how data is marked up and these files are given with xsd extension. XML Schema offers many advantages over DTD. One of the greatest advantages is the support for data types. Since DTDs are designed for use with text, they have no mechanism for defining the content of elements in terms of data types. Therefore a DTD cannot be used to specify numeric ranges or to define limitations or checks on the data content. The XML Schema provides a means of specifying element content in terms of data type, so that document type authors can provide criteria for validating the content of elements as well as the markup itself. Some other strengths of a Schema are; they are written in XML thus avoids the need for another processing software; supporting namespaces and extensibility to future additions (W3Schools, 2008).

XML Schema of the above example can be defined as below:

```
<?xml version="1.0" encoding="utf-8" ?>
 <xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.unipi.gr"
  xmlns="http://www.unipi.gr"
  elementFormDefault="qualified">
  <!-- An XML Schema Example -->
  <xsd:element name="document" type="docType" />
  <xsd:complexType name="docType" mixed="true">
   <xsd:sequence>
    <xsd:element name="title" type="xsd:string" />
    <xsd:element name="pub_date" type="xsd:date" />
    <xsd:emement name="production" type="productionType" />
    <xsd:complexType name="productionType">
     <xsd:complexContent>
      <xsd:restriction base="xsd:integer">
       <xsd:attribute name="id" type="xsd:positiveInteger" />
       <xsd:minInclusive value="0" />
       <xsd:maxInclusive value="1000" />
      </xsd:restriction>
     </xsd:complexContent>
     <xsd:attribute name="media" type="xsd:string" />
    </xsd:complexType>
    <xsd:element name="chapter" type="chapterType" />
    <xsd:complexType name="chapterType" />
     <xsd:sequence>
```

```
        <xsd:element name="para" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

The schema element is the root element of every XML Schema where xsd is the namespace prefix. The fragment `xmlns:xsd="http://www.w3.org/2001/XML Schema"` indicates that the elements and data types used in the Schema come from the `"http://www.w3.org/2001/XMLSchema"` namespace. The `targetNamespace` attribute indicates that the elements defined by this schema are from `"http://www.unipi.gr"` namespace and `xmlns` attribute gives the default namespace. The `elementFromDefault= "qualified"` fragment indicates that any element used by the XML instance document which will declare in this schema must be namespace qualified. All namespaces used in the schema must be declared in prolog and all elements and data types must be defined in the body. An XML Schema document consists of four basic constructs: declaration of elements, definition of types (`simpleTypes` and `complexTypes`), the possibility to define sub-types by extending or restricting super-types, and use of aliases (`substitutionGroup` mechanism).

3.5.1 Simple type

An XML element that contains only text (data in any type) is a simple type element. It cannot contain any other elements or attributes. But it can have a default value or fixed value set. Simple element is defined by `<xsd:element name="name_of_element" type="data_type" default="default_value"` (or `fixed="fixed_value"`) `/>`.

Common data types in XML Schema are `string`, `decimal`, `integer`, `boolean`, `date` and `time`. The type of a simple element is defined by `<xsd:simpleType>`.

3.5.2 Attributes

In Schema an attribute is always declared as a simple type and an element with attributes always has a complex type definition. An attribute is defined by, `<xsd:attribute name="attribute_name" type="data_type"/>`. Attributes also can have a `default` or a `fixed` value specified. Even though all attributes are optional by default, with use attribute it can be explicitly specify whether it is optional or required.

3.5.3 Restrictions and extensions

Defining a type for XML element or attribute imposes a restriction for the element or attribute content. With XML Schemas, it is able to add own user restrictions to user XML elements and attributes. In the example restriction has imposed on `id` attribute. The `id` can have only integer values between `0` and `1000` (including that numbers). Likewise to limit the content of an XML element to a set of accepted values, the `enumeration` constraint can be used. All restrictions that can be applied to data types are given in XML Schema Specification. Moreover types can be defined by extending existing types.

3.5.4 Complex type

An XML element that contains elements, mixed content, empty content or attributes is considered to be a complex type element. For example, document element is a complex one. Its type has been defined as `docType` separately. The child elements of document element are surrounded by the `<xsd:sequence>` indicator. Defining complex types separately offers more flexibility, because these types can be used by other elements too.

3.6 XML namespaces

Since XML uses no fixed element names, the same element name could be used in different documents to describe different types of elements. If such XML documents are added together, there would be an element name conflict. XML namespaces is a method to avoid these element name conflicts. It solves the name conflicts using a `prefix` and the namespace attribute (`xmlns`) alone with element name. The namespace attribute is placed in the start tag of an element and all child elements with the same prefix are associated with the same namespace. The W3C Namespace specification states that the namespace itself should be a Uniform Resource Identifier (URI) which is a string of character that identifies an Internet Resource. The most common URI is the Uniform Resource Locator (URL) which defines an Internet domain address.

3.7 Document Object Model (DOM)

XML standards include specifications of how an XML document should be parsed and represented within any computer irrespective of type or operating system. This internal representation (tree representation) of an XML document, which is generated within a computer by an XML parser, is called the Document Object Model – DOM (Le Hegaret, et al., 2004). DOM allows a single document to be accessed in the same way by different applications running on different computer platforms through XML tag references. In a software context, the DOM is a programming API (Application programming Interface) for an XML document, which defines the logical structure of documents, and the way a document is accessed and manipulated. It details the characteristic properties of each element of a document, thereby detailing how these components can be manipulated and, in turn, manipulating the whole document. Therefore with the DOM, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content. As a W3C specification, one important objective for the DOM is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM can be used with any programming language and provides precise, language-independent interfaces.

3.8 Unicode system

In HTML, a document is in one particular language, whether English, Japanese or Arabic. Applications that cannot read the characters of that language cannot do anything with the document. But XML uses the Unicode standard, which is a character encoding system that supports intermingling of text in the world's major languages. Because of that, applications that read XML can properly deal with any combination of these character sets. Thus XML will enable exchange of information in different languages. The character set used in the XML document encoding is specified in xml declaration with encoding attribute. Since XML is fully internationalized for both European and Asian languages, with all conforming processors required to support the Unicode character sets in UTF-8, UTF-16 and ISO-8859-1.

3.9 Viewing XML documents

Since XML documents are text based, they can be viewed in any text editor. But they will not view any hierarchical structure of the document elements. The browsers like Internet Explorer 5.0 (and higher), Netscape 6 or any other XML editors like XMLSpy can be used to view XML documents. Any error in an XML document will be reported by those browsers or editors. As mentioned above, since XML elements are user defined the browser has no idea how to display the content other than just viewing whole document as it is. Therefore XML documents are associated with stylesheets which provides GUI (Graphic User Interface) instruction for a processing application like a web browser. There are different ways to visualize the content of an XML document using these stylesheets and they are presented in Figure 3-2. In addition, JavaScript (or VBScript) can be used to import data from an XML document and display them inside an HTML page. The use of eXtensible Stylesheet Language family – XSL (Adler, et al., 2001) in XML data visualization will be discussed in the next chapters.

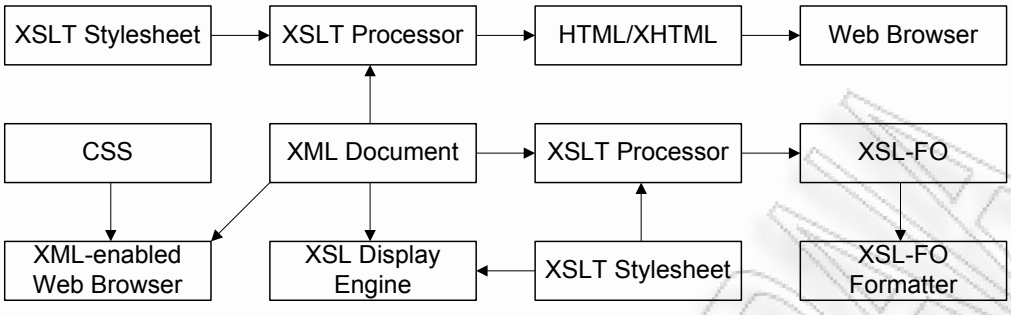


Figure 3-2: Options for displaying XML documents

3.10 Stylesheets

The stylesheets are the documents that provide information on data presentation. The international standard for stylesheet for SGML document is DSSL, the Document Style and Semantic Specification Language. This provides Schema-like languages for stylesheets and document conversion, and is implemented in the Jade formatter. Cascading Stylesheet provides a simple syntax for assigning styles to elements, and has been partly implemented in some HTML browsers. Extensible Stylesheet Language has been created for use specifically with XML. It uses XML syntax but combines formatting features from both DSSL and CSS. The stylesheet of an XML document should be declared in the prolog section of the XML document in the format of `<?xml:stylesheet href="location_of_stylesheet" type="text/xsl"?>` in case of CSS the `type="text/css"`.

The separation of style from the content allows for the same data to be presented in different ways and it enables:

- Reuse of fragments of data: the same content should look different in different contexts.
- Multiple output formats: different media (paper, online), different sizes (manuals, reports), different classes of output devices (workstations, hand held devices)
- Styles tailored to the reader's preferences (e.g. accessibility): print size, color, simplified layout for audio readers.
- Standardized styles: corporate stylesheets can be applied to the content at any time.
- Freedom from style issues for content authors: technical writers need not be concerned with layout issues because the correct style can be applied later.

A stylesheet specifies the presentation of XML content using two basic categories of techniques. One is an optional transformation of the input document into another structure and the other is a description of how to present the transformed information.

3.11 XML based markup languages

Due to the flexibility and robustness of XML technology, a number of markup languages have been developed according to the XML standards and some of them are: Chemical Markup Language – CML (Murray-Rust, et al., 1995), a markup language to managing molecular information; Wireless Markup Language – WML (Engelschall, et al., 2001), used to markup Internet applications for handheld devices like mobile phones; MathML (Carlisle, et al., 2003), an XML application for describing mathematical notation and capturing both its structure and content; XSL, a language for expressing stylesheets; GML, for transport and storage of geographic information; and KML to project this geographic information using Google's Maps/Earth API.

3.12 Conclusion

XML is a simplification of SGML. It enables easy implementation, and interoperability with both SGML and HTML. Any valid XML document must have a DTD or Schema that describes the structure. XML separates data content from presentation information. Stylesheets provide presentation information for XML documents and XML elements can be manipulated through the DOM. There are number of markup languages developed in different domains based on XML standards.

4 The Geography Markup Language (GML)

4.1 Introduction

This chapter is focused on Geographic Markup Language, and discusses the use of GML and how it can be used to model geographic data. It also describes the possibilities of visualizing GML data. Through that it explains the structure of GML data and how to visualize them.

4.2 Background and evolution of GML

Geographic Information Systems have proved to be an efficient tool for decision making for various organizations dealing with geographic data. However, implementation of such a system is difficult and relatively costly. High production cost and importance of geographic data in mapping has increased the need of data sharing. Furthermore, geographic information is not confined to a specific system or an application domain. It resides in heterogeneous environments in various application domains and systems. When geographic data is shared between organizations dealing with different applications, there might be a heterogeneity problem if the organizations use different GIS platforms, hence, producing different digital formats of the data. Even if they have identical GIS platforms, and hence use the same database paradigm, e.g., relational, they might have different conceptual database schema, different data collection schemes, or different quality parameters (Bishr, 1997). In this scenario, information cannot be shared or transferred readily. We need to have file conversions and in the latter case, there should be a perfect mapping between the corresponding database schemas.

In the geographic domain, besides information sharing, there is a need of information integration. Geographic data describes the geographic features and phenomena on the Earth, and events on the Earth do not take place in isolation. For example, if an earthquake takes place, many objects like trees, roads, buildings, agricultural fields, and water bodies might become affected. To perform earthquake analysis, we need to have an integrated data set, which describes all the features affected.

Today, the Internet is the main platform for data sharing. The development of Internet has demanded that our technologies be extensible and comprehensible (Lake, 2001). Thus, to share and integrate geographic data in the Internet environment requires a standard data format, which is interoperable, extensible and suitable for Internet Technology. OGC, whose mission is to address the lack of interoperability between systems that process geo-spatial data, has established the XML-based standard, known as GML: *The Geographic Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the spatial and non-spatial properties of geographic features* (Cox, et al., 2004).

GML provides an XML-based encoding of geo-spatial data. It can be viewed as a basic application framework for handling geographic information in an open and non-proprietary way. Like any XML encoding, GML represents (geographic) information in the form of text, thus it can be readily intermixed with a wide variety of data types including text, graphics, audio and more (Lake, 2001). GML documents, like XML, are both human-readable and machine parsable. So, they are easier to understand and maintain than proprietary binary formats.

Like XML, GML also separates content of geographic data from its presentation. GML mainly describes the structure of geographic data without regard to how the data can be presented to a human reader. Since GML is based on the XML standard, it can readily be styled into variety of presentation formats including vector and raster graphics, text, and sound (Cox, et al., 2004). Graphical output such as a map is one of the most common presentations of GML.

OGC initially developed GML 1.0 (May 2000), which was based on a combination of XML DTDs and the Resource Description Framework (RDF). GML 2.0, which replaces GML 1.0, was developed and adopted in March 2001 by OGC. It is entirely based on XML Schema (Thompson, et al., 2004). Adoption of XML Schema in GML incorporates support for type inheritance, distributed schema integration, and namespaces (Lake, 2001). Moreover, it provides a rich set of primitive data types such as string, boolean, float, as well as construction of user-defined complex data types, data ranges and masks. GML 2.0 is based on linear geometry, it does support coordinates to be specified in three dimensions, but it does not provide direct support for three-dimensional geometric constructs.

GML 3.1.1 (OGC, 2004) has been extended to represent geo-spatial phenomena in addition to simple 2D linear features, including features with complex, non-linear, 3D geometry, features with 2D topology, features with temporal properties, dynamic features, coverage and observations. It also provides more explicit support for properties of features and other objects, of which the value is complex. Moreover, it represents spatial and temporal reference systems, units of measure and standards information.

4.3 GML features

GML is based on the geographic model developed by the OGC, which describes the world in terms of geographic entities called features. This geographic model is based on the OGC Abstract Specification, 4 which defines a geographic feature as *an abstraction of a real world phenomenon, it is a geographic feature if it is associated with a location relative to the Earth* (Cox, et al., 2004).

Thus, real world phenomena are represented digitally as a set of features. The state of a feature is defined by a set of properties, where each property has a name, value and type descriptions. Geographic features are those features whose properties may be geometry-valued. Properties of a feature may be simple properties or geometric properties. Properties with simple types (e.g., integer, string, float, boolean) are collectively known as simple properties and the properties that are geometry-valued, are known as geometric properties.

A feature can have multiple simple properties as well as multiple geometric properties. A feature can be composed of other features. Such a feature is termed as a feature collection. A feature collection has a feature type and thus may have its own distinct properties, in addition to the features it contains. A single feature like a city can be composed of other features like rivers, roads and colleges. So, a city can be represented as a feature collection in GML where the individual features in the collection represent roads, rivers and colleges.

4.3.1 Simple features

The GML 3.1.1 specification is concerned with the OGC 'simple feature' model. It is the simplified version of a more general model described by the OGC Abstract Specification. There are two major simplifications, which lead to the definition of simple features as:

- Features having either simple properties (integer, real string, boolean) or geometric properties and
- Features whose geometries are assumed to be defined in a two-dimensional SRS, using linear interpolation between coordinates.

The Abstract Feature Model for simple features is presented in Figure 4-1.

4.3.2 Geometry elements

In accordance with the OGC simple feature model, GML provides geometric elements corresponding to the following geometry classes:

- Point,
- LineString,
- LinearRing,
- Polygon,
- MultiPoint,
- MultiLineString,
- MultiPolygon,
- MultiGeometry

In addition there are `<coordinates>`, `<coord>`, `<pos>` and `<posList>` elements for encoding coordinates and `<Box>` element for defining extent.

Each geometry element is used to encode instances of the corresponding geometry class. Thus, a Point element is used to encode an instance of the Point geometry class, and a Polygon element is used to encode an instance of the Polygon geometry class and so on.

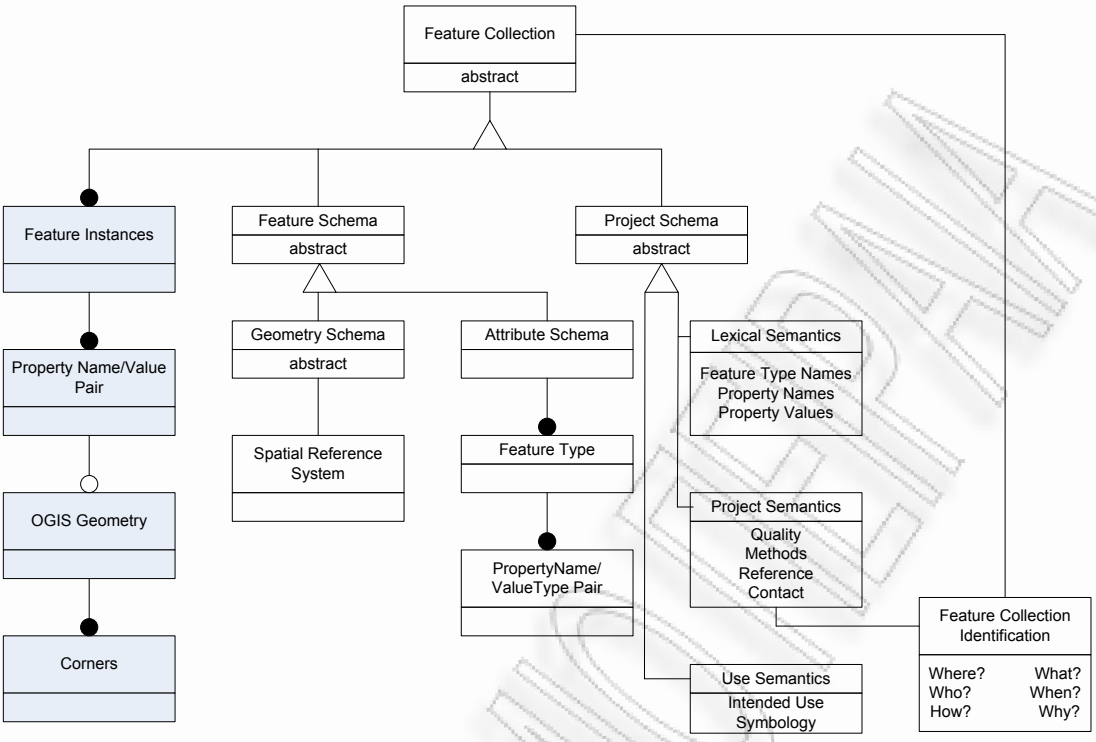


Figure 4-1: GML Abstract Feature Model (OGC, 2004)

The coordinates of any Geometry class instance (Point, Polygon) are encoded as a sequence of `<coord>` elements, or as a single string contained within a `<coordinates>` element. Examples of the use of `<coord>` and `<coordinates>` elements to encode coordinate information are shown below.

```
<Point srsName="http://www.openings.net/gml/srs/epsg.xml#4326">
    <coord><X>37.3</X><Y>23.8</Y></coord>
</Point>
<Point srsName=http://www.openings.net/gml/srs/epsg.xml#4326">
    <coordinates>37.3,23.8</coordinates>
</Point>
```

The Box element requires two coordinate tuples to define an extent. So, either a sequence of two `<coord>` elements or a `<coordinates>` element with exactly two coordinate tuples can be used. Since a Box cannot be contained by other geometry classes, the `srsNameattribute` must be provided, which identifies the coordinate system. An example is shown below.

```
<Box srsName="http://www.openings.net/gml/srs/epsg.xml#4326">
    <coord>
        <X>0.0</X>
        <Y>0.0</Y>
    </coord>
    <coord>
        <X>52.5</X>
        <Y>55.0</Y>
    </coord>
</Box>
```

Each Point element consists of either a single `<coord>` element or a `<coordinates>` element containing exactly one coordinate tuple. The `srsName` is optional because a Point element may be contained within other elements that specify a reference system.

A LineString is a piece-wise linear path defined by a list of coordinates that are assumed to be connected by straight line segments. At least two coordinate pairs are required to encode an instance of the LineString class.

Furthermore, there is the alternative of using a <posList> pair of tags, so as to define the coordinates. The main difference between these two declaration types is that, with the later, there is no need for the comma (,) separation.

```
<LineString srsName="http://www.openings.net/gml/srs/epsg.xml#4326">
    <coordinates>0.0,0.0 10.0,15.0 65.0,75.0</coordinates>
</LineString>
<LineString srsName="http://www.openings.net/gml/srs/epsg.xml#4326">
    <posList>0.0 0.0 10.0 15.0 65.0 75.0</poList>
</LineString>
```

A LinearRing is a closed, piece-wise linear path defined by a list of coordinates that are assumed to be connected by straight line segments. In order to encode an instance of the LinearRing class, at least four coordinate pairs are required and the last coordinate pair must coincide with the first coordinate pair so that it forms a ring. LinearRing is used in construction of Polygons, so the srsName attribute is not required here.

A Polygon is a connected surface of which the boundary is a set of LinearRings. The boundaries are characterized as interior and exterior boundaries. A Polygon must have at most one exterior boundary and zero or more internal boundaries. An example of a Polygon instance is given below.

```
<Polygon srsName="http://www.openings.net/gml/srs/epsg.xml#4326">
 <outerBoundaryIs>
  <LinearRing>
   <coordinates>0.0,0.0 60.0,0.0 60.0,60.0 0.0,0.0</coordinates>
  </LinearRing>
 </outerBoundaryIs>
 <innerBoundaryIs>
  <LinearRing>
   <coordinates>10.0,10.0 10.0,30.0 30.0,30.0 10.0,10.0</coordinates>
  </LinearRing>
 </innerBoundaryIs>
</Polygon>
```

A MultiPoint is a collection of Points; a MultiLineString is a collection LineStrings; a MultiPolygon is a collection of Polygons; and a MultiGeometry is a collection of any arbitrary geometry elements such as Points, LineStrings, Polygons and so on. Each of these collections uses an appropriate membership property to contain other elements. An example of a MultiLineString instance is shown below.

```
<MultiLineString srsName="http://www.openings.net/gml/srs/epsg.xml#4326">
    <LineStringMember>
        <LineString>
            <coordinates>25.5,0.5 55.5,60.0</coordinates>
        </LineString>
    </LineStringMember>
    <LineStringMember>
        <LineString>
            <coordinates>54.0,10.5 72.6,40.5</coordinates>
        </LineString>
    </LineStringMember>
</MultiLineString>
```

Thus, GML provides eight geometry elements corresponding to their geometry classes. Among these eight geometry elements, Point, Box, LineString, LinearRing and Polygon are primitive geometry elements, and MultiPoint, MultiLineString, MultiPolygon and MultiGeometry are geometry collections.

If a feature has properties that are geometry-valued, then these properties are termed as geometric properties. Geometry-valued properties describe position and shape of the feature. Since the OGC abstract specification defines a small set of basic geometries, GML defines a set of geometric property elements to associate these geometries with features.

There are three levels of naming geometry properties in GML: formal names that denote geometry properties based on the type of geometry allowed as a property value, descriptive names provide aliases or synonyms for the formal names, and application specific names chosen by user and defined in

application schemas based on GML. The formal and descriptive names for the basic geometric properties are listed in Table 4-1.

Formal Name	*Descriptive Name*	*Geometry Type*
boundedBy	-	Box
pointProperty	location, position, centerOf	Point
lineStringProperty	centerLineOf, edgeOf	LineString
polygonProperty	extentOf, coverage	Polygon
geometryProperty	-	*any*
multiPointProperty	multiLocation, multiPosition, multiCenterOf	MultiPoint
multiLineStringProperty	multiCenterLineOf, multiEdgeOf	MultiLineString
multiPolygonProperty	multiExtentOf, multiCoverage	MultiPolygon
multiGeometryProperty	-	MultiGeometry

Table 4-1: Basic geometric properties

4.3.3 Time elements

Recent advances in GML added the ability for describing a wide variety of features that are time dependent from the motion of a person or vehicle, to the development of a hurricane and the impact zone of an earthquake (Lu, et al., 2007). The description of time is handled by the temporal.xsd schema.

One way for representing temporal features in a GML document is by using the <TimePeriod> complex type. This type consists of two distinct simple types <begin> and <end>. Date and time information can be added inside these tags, following several prototyping methods (ISO 8601, CalDate, URI Reference, Decimal value). For the purposes of the current thesis, the ISO 8601 format, has been chosen, which can be seen in the following example.

```
<gml:TimePeriod>
        <gml:begin>2001/02/09T16:32.53</gml:begin>
        <gml:end>2001/02/09T16:33.23</gml:end>
</gml:TimePeriod>
```

According to ISO 8601 notation, every timestamp has to conform to the following format (Wolf, et al., 1997):

```
{Year/Month/DayTHour:Minute.Second}
```

4.4 Core GML schemas

GML 3.1.1 defines three base schemas for encoding spatial information. These schemas provide the building blocks for constructing GML application schemas. In other words, the base GML schemas provide a meta-schema or set of foundation classes, from which an application schema can be constructed.

The Feature schema defines the general feature-property model which describes both abstract and concrete elements and types. It supports feature collection (as feature type) and includes common feature properties such as *fid* (a feature identifier), name and description. The `<include>` element in the Feature schema brings in the definitions and declarations contained in the Geometry schema for use in defining feature types. The UML model of Abstract Feature schema is shown in Figure 4-1.

The GML Geometry schema includes the detailed geometry components that are type definitions for abstract geometry elements, concrete (multi) point, line and polygon geometry elements, as well as complex type definitions for the underlying geometry types. The Geometry schema targets the `gml` namespace. The `<import>` element in the Geometry schema brings in the definitions and declarations contained in XLink schema. The UML model of Geometry schema is shown in Figure 4-2.

As W3C states, the XLink allow elements to be inserted into XML documents in order to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of HTML, as well as more sophisticated links. The XLink schema in GML provides the XLink attributes to support linking functionality.

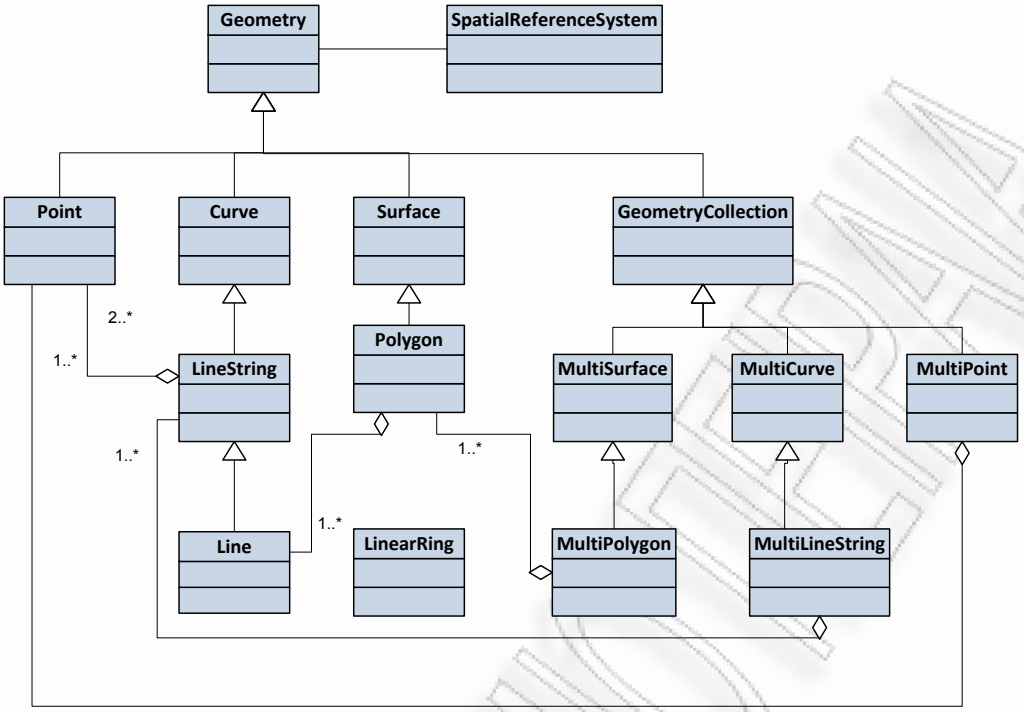


Figure 4-2: GML Geometry Model (OGC, 2004)

4.5 Encoding geographic information with GML

Using the three base XML schemas described above, it is possible to encode a wide variety of geospatial information such as; features with or without geometry, geometry, collection of features, and associations between features. GML 3.1.1 specification explains how these encodings are performed.

A geographic feature in GML is a list of simple and geometric properties. The features are captured as element names in GML and the feature class definition prescribes the named properties that a particular feature type should have. GML follows some conventions in encoding these. Type definitions take the corresponding class name and append the `Type` suffix. Type names are in mixed case with leading capital (e.g. A Road feature is coded as `<element name="Road" type="RoadType"/>`, the names of geometric properties and attributes are in mixed case with leading lower case character (e.g. `pointProperty`, `familyName`). The abstract elements' name is in mixed case with leading underscore (e.g. `Feature`, `FeatureCollection`).

4.6 GML application schemas

The three core XML Schemas (Feature Schema, Geometry Schema and XLink Schema) alone do not provide a schema suitable for constraining data instances; rather, they provide base types and structures which may be used by an application schema.

A GML application schema is an XML Schema document constructed with the components provided by the base GML schemas. It declares the actual feature types and property types of interest for particular domain, using components from GML in standard ways. Defining an application schema from components in base GML schemas, benefits from standardized constructs and guaranteed to conform to the OGC Feature Model.

Any GML application schema must adhere to the schema development rules described in GML 2.0 specification and must not change the name, definition, or data type of mandatory GML elements. But in application schemas, the abstract type definitions can be freely extended or restricted. Furthermore an application schema must target a namespace and that must not be `gml` namespace. And such an application schema must be made available to anyone receiving data structured according to that schema.

4.7 Structure of an application schema

As any other XML document, a GML application schema also consists of header (prolog) and body.

The header starts with a xml declaration which consists version and encoding attributes. The value of encoding attribute indicates the Unicode character set used in encoding GML.

```
<?xml version="1.0" encoding="utf-8"?>
```

The <schema> open tag must contain the namespaces that are used for the schema definition. The targetNamespace attribute defines the user namespace, and its value is a unique identifier for the GML namespace. For an example:

```
<schema
    targetNamespace="http://www.unipi.gr/gml"
    xmlns:gml="http://www.unipi.gr/gml"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns="http://www.w3.org/2000/10/XMLSchema"
    elementFormDefault="qualified"
    version="1.0">
    <annotation>
        <appinfo>Village.xsd v1.0 </appinfo>
        <documentation xsl:lang="en">
            GML application schema for Topographic data
        </documentation>
    </annotation>
    <!-- import constructs from the GML Feature & Geometry schemas-->
    <import
        namespace="http://www.opengis.net/gml"
        schemaLocation="feature.xsd" />
</schema>
```

The <annotation> element provides the information about the application schema and the <import> element imports other schema definitions that are used in this schema. In this case the GML feature schema definition (feature.xsd) is imported and it includes geometry and xlinks schemas. The schemaLocation attribute describes the location path of the importing schema.

The first <element> in the application schema defines the root element and becomes the open tag of the GML file. In schema definition, the root element is a substitutionGroup for the glm: FeatureCollection and an extension based on gml:AbstractFeatureCollectionType. For example, if the topographic information of a village is modeled in GML, it may consist of road, river, terrain, building, utility and boundary feature classes. The collection of these feature classes is the root element and if it is named as VillageModel, it can be defined in the schema as follows.

```
<element
    name="VillageModel"
    type="gml:VillageModelType"
    substitutionGroup="gml:_FeatureCollection" />

<complexType name="VillageModelType">
    <complexContent>
        <extension base="gml:AbstractFeatureCollectionType">
            <sequence>
                <element name="dateCreated" type="date" />
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

There are some properties that are shared by many objects in a schema definition. Such shared definitions are modeled with group definitions in XML Schema. For example, temporal and metadata information will be related to every feature in the model. To inherit these properties in all features, a

`complexType` can be defined that refers to group definitions and all feature types have to be defined as extensions of that type.

```
<group name="TemporalData">
    <sequence>
        <element name="begindate" type="string" />
        <element name="enddate" type="string" />
    </sequence>
</group>
<group name="MetaData">
    <sequence>
        <element name="source_type" type="string" />
        <element name="source_description" type="string" />
        <element name="accuracy" type="string" />
        <element name="actuality" type="string" />
        <element name="code_num" type="integer" />
    </sequence>
</group>
<complexType name="TopoDataType" abstract="true">
    <complexContent>
        <extension base="gml:AbstractFeatureType">
            <sequence>
                <element name="code_id" type="integer" />
                <group ref="gml:TemporalData" />
                <group ref="gml:MetaData" />
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

Features are the objects that are visible on the map. In the example below, `building` is a one feature type and the collection of building features are represented by `BuildingLayer` type. All such feature layers can be defined as a `TopoDataLayerType` which is an extension of `gml:AbstractFeatureCollectionType`.

```
<complexType name="TopoDataLayerType">
    <complexContent>
        <extension base="gml:AbstractFeatureCollectionType" />
    </complexContent>
</complexType>
<element
    name="BuildingLayer"
    type="TopoDataLayerType"
    substitutionGroup="gml:_FeatureCollection" />
<element
    name="Building"
    type="gml:BuildingType"
    substitutionGroup="gml:_Feature" />
<complexType name="BuildingType">
<complexContent>
    <extention base="gml:TopoDataType">
        <sequence>
            <element name="type" type="string" />
            <element name="function" type="string" />
            <element name="height_category" type="string" />
            <element name="height" type="string" />
            <element name="status" type="string" />
            <element ref="gml:geometryProperty" />
            <element name="heightlevel" type="integer" minOccurs="0" />
            <element name="name" type="string" minOccurs="0" />
        </sequence>
```

```
        </extension>
        </complexContent>
    </complexType>
```

All other feature layers can also be defined in the same way.

A property of feature can have a predefined number of allowed values. When such a property is `string` type, it is possible to make an enumeration type for those properties that lists allowed entries. The `function` property of above building feature can be altered as follows:

```
<element name="function" type="functionType" />
<simpleType name="functionType">
    <restriction base="string">
        <enumeration value="Municipality" />
        <enumeration value="Police office" />
        <enumeration value="Post office" />
        <enumeration value="Church" />
        <enumeration value="Hospital" />
        <enumeration value="Station" />
        <enumeration value="Storage tank" />
        <enumeration value="Other" />
    </restriction>
</simpleType>
```

4.8 Structure of GML documents

Any GML document starts with the standard XML header in which the character encoding and xml version are mentioned. Then the root element of the GML document is appeared with all namespace definitions used in the GML document and schema location. The `<gml:boundedBy>` element contains the bounding box of all the features in the GML document. The `srsName` attribute in the `<Box>` element indicates the spatial reference system where coordinates of features are based on. After the bounding box of all feature collections, each data layer (feature collection) is described alone with related bounding box elements. A GML document fragment based on the above application schema fragment is given below.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!--File: Model.gml-->
<gml:VillageModel
    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.unipi.gr/GMLModel.xsd">
    <gml:dateCreated>May 2008</gml:dateCreated>
    <gml:featureMember>
        <gml:boundedBy>
            <gml:Box>
                <gml:coordinates>
                    190000,446000 193000,449000
                </gml:coordinates>
            </gml:Box>
        </gml:boundedBy>
    </gml:featureMember>
</gml:VillageModel>
```

4.9 Validation of GML documents

The correctness of a GML document has to be checked through validation. First the document must be well formed (that is the document should comply with XML syntax). If it is well formed the next step is to check if the GML document is valid according to its Schema definition. It can be performed with packages like XMLSpy or custom applications using Java/.NET, implemented for this specific reason.

4.10 Viewing GML data

According to the principles of XML technology and GML development goals, GML should not contain presentation data. As a consequence, GML data files do not contain styling information. But there are ways to add styles to the GML features. One method is, when GML data is imported into GIS or other CAD application, styles can be added through that software. Another method is developing a viewer that can read GML data and generate cartographic view. The viewer software needs to have an interactive module to change the graphical properties of the features. It is also possible to provide a separate document that contains styling information along with GML data that viewer software can read both and generates a graphical display. But still this kind of viewers is not commonly available. There is another possibility, which is transforming GML into another XML graphic format, KML and using an already implemented API, the visualization concept is achieved. That is the main focus of this thesis.

4.11 Conclusion

GML is an XML based standard for describing geographic data. GML 3.1.1 is the latest specification in implementation level. GML models Geographic data as feature, and feature collections which are defined in application schemas using the basic constructs provided core GML schemas.

A GML document that is based on such a schema is a *valid* XML document that can be processed by any XML parser. GML provides no information about presentation and therefore data content has to be converted into another graphical format.

5 The Keyhole Markup Language (KML)

5.1 Introduction

Keyhole Markup Language is an XML geospatial data file format for presenting two- and three-dimensional graphics. It has many advantages over other graphics formats. This chapter describes its advantages, capabilities and suitability for viewing geographic data. It also examines the various flavors of users' applications, which are responsible for visualizing such information.

5.2 Visualization of geographic data

Geographic data provides information of man-made and natural features on earth's surface such as roads, hydrology, buildings, land cover, terrain relief, and boundaries etc. In the visualization, process these features have to be graphically represented. The basic graphic elements points, lines and areas can be used to create the visual designs irrespective of the medium on which it is displayed. Point elements convey a sense of position and are the most fundamental of these three types. Lines are linear array of points and exhibits direction as well as position. Area elements exhibit extent, direction, and position and are two-dimensional array of points (Robinson, et al., 1995).

According to the Bertin's introduction the shape, size, orientation, color (hue), value (tone) and texture are the graphic variables that can be used to make one symbol different from another (de By, et al., 2001).

In order to make the visualization more meaningful, one should identify and analyze the data to be presented before to symbolize. Text is used in visualization to transfer the information that is not possible to symbolize.

In this research the target presentation media is a Web browser and therefore the graphical format used for the visualization must suit the Web as well as the data content (geographic data).

5.3 Evolution of KML

Most of the graphics on the Web consist of images represented as a sequence of colored pixels. GIF (Graphics Interchange Format), JPEG (Joint Photographic Experts Group) and PNG (Portable Network Graphics) are examples of bit-mapped graphic formats which are based on this principle. An alternative approach for sending pixel values down the Web is sending instructions for drawing features like lines or curves (vectors) and filling these shapes, which offers great advantages over pixel based formats.

By now, a number of vector formats are being used on the Web; e.g. Flash, Precision Graphics Markup Language (PGML), Web Compute Graphic Metafile (WebCGM) and Portable Document Format (PDF).

Their implementation through plug-ins and difficulty in integration with the rest of the Web, prevented them of being used in all the places that natively supported raster graphics could be. Moreover, no single format was widely and well supported by the tools for creating Web pages, and in general there was a lack of cross-platform support and of accessibility and well internationalized solutions (Lilley, 2002).

Keyhole Inc., which was acquired by Google Inc. in 2004, developed a new standard format for vector graphics, KML that matches the needs of content providers and browsers like.

5.4 Keyhole Markup Language (KML)

KML is an XML grammar and file format for modeling and storing geographic features such as points, lines, images, and polygons for display in Google Earth, Google Maps, and Google Maps for mobile. Google's backing of KML makes it an important format for the exchange of geographic information.

KML 2.0 specification was issued by Google Inc. in 2006 and as it describes, it allows for three main types of graphic objects; vector graphic shapes (e.g. paths consisting of straight lines and curves), images and text.

In KML, graphical objects can be grouped, styled, transformed and composites into previously rendered objects. Furthermore, the feature set may include nested transformations, clipping paths, alpha masks, filter effects, template objects and both procedural and declarative animation. KML drawings can also be dynamic and interactive.

KML is a bridge between design and programming, because unlike traditional methods of creating graphics, graphics in KML are created through an XML based programming language and consequently integrates well with other W3C standards such as the DOM. KML is much like a vector based graphics program, with the exception of an associated graphical program interface. Instead, vector images are created through text based commands that are formatted to comply with XML specifications.

KML offers many important advantages over other formats.

- Zoomable: Images can be magnified without sacrificing sharpness, detail of clarity.
- Text stays text: Text in KML images remains editable and searchable. There are no font limitations and users will always see the image the same way it was created.
- Small file size: On average, KML files are smaller than other Web-graphic formats and are quick to download.
- Display independence: Images are always crisp on screen and print out at the resolution of the printer.
- Color control: KML offers a palette of 16 million colors, support for ICC color profiles, sRGB, gradients and masking.
- Interactivity and intelligence: Since KML is XML based it offers high dynamic interactive graphics far more sophisticated than bitmapped or even Flash images. Moreover KML images can respond to user actions with highlighting, tool tips, special effects, audio, and animation.
- OGC standard: The KML 2.2 specification has been submitted to the Open Geospatial Consortium to assure its status as an open standard for all geo-browsers. As of November 2007, the OGC has a new KML 2.2 Standards Working Group. Comments were sought on the proposed standard until January 4, 2008 and it became an industry standard on April 14, 2008.
- True XML based: Since KML is an XML grammar; it offers all the advantages of XML. Interoperability, Internationalization (Unicode support), Wide tool support, Easy manipulations through standard APIs such as DOM API.
- Easy transformation through XML Stylesheet Language Transformation (XSLT).

5.4.1 Creating KML files

KML documents can be hand-coded using any simple text editor or sophisticated XML editors like TextPad, Vim, or XML-Spy. XML-Spy offers syntax highlighting, elements and attributes completion, validation of KML content against the KML Schema. This method is feasible only for simple and small documents.

Another alternative is to use a custom application which will be fed by geographic data and in turn, it will produce the corresponding files. There are quite a lot applications accomplishing such tasks; to name a few, Google Earth (Google Earth, 2008), Google My Maps (Google Maps, 2008), Map Builder (Bidochko, et al., 2005) etc.

A more flexible, robust and advanced method of generating KML is through XSLT stylesheets. As discussed in the previous chapter, XSLT is designed for XML document transformation. Since KML is XML based, application of XSLT to generate KML from XML data is straightforward. This research emphasizes on this approach for visualizing GML data.

5.5 KML features

An application, which wants to exploit the use of cartographic services of Google, must utilize the API available from the vendor.

The Google Maps API is an interface, which was developed by the Google Company and allows us to integrate Google maps on our Web pages using JavaScript. Can anyone with the facilities provided to

design indicators over maps, or develop even more sophisticated applications. At present, however, services provided by Google are only available for web pages and cannot be used by another application. Not based on an open standard such as SOAP/XML, but uses JavaScript as mentioned above. For this reason, the only way of integrating them are websites.

The advantage of using these services is that they are provided free of charge (at least for web sites with a maximum number 50000 visits/day). The only thing that is needed to use these services is included in the Google system to grant a password to the API of Google (API key). But if the services are rendered free from Google should anyone who uses them not to make them available on the site of payment but also free to visitors of his website. Below is a brief presentation of the characteristics of the services provided by Google Maps API:

- `GMap` class: This is the basic class. An object of class `GMap` corresponds to a map on the page. A user can create as many instances of this class as she wants (one for each map on page). When creating a new snapshot map, we set an item on page which contains the map. The map then uses size as the size of the item which includes unless we define it differently. The `GMap` class provides methods for handling the center of the map and the level of zoom, and methods for adding or removing various overlays (such as instances of class `GMarker` and `GPolyline`). In addition, it provides methods that give us the possibility to open a "window information" which contains various information on the map.
- Events: Usage of the event listeners can introduce dynamic elements to our application. An object of this class provides a number of events (events) and our application's implementation can "listen" using static methods such as `GEvent.addListener` or `GEvent.bind`. So the program can, for example displays a message depending on a user's click on the map.
- The Info Window: Every map has a single "window information", which displays HTML content in a window above the map. The information window resembles a "bubble" in a book comic. It consists of a region with the content of the information which becomes thinner aside and become as an indicator showing at a designated point on the map. If anyone has used the Google Maps or Google Local, then chances are to have seen a "window information" when clicked on an icon (marker). Another feature of these windows is that nobody can display more than one simultaneously in a given map but can move the window and change its contents if this is desirable.

  The basic formula for a window of information is `openInfoWindow`, which takes an entry point and an HTML DOM element. The window information appears in the text given point of the map and displays the DOM element in the region comprising. `OpenInfoWindowHtml` method is similar, but takes an HTML string as the second argument instead of a DOM element. Similarly, `openInfoWindowXslt` gets one point, an XML DOM element and the URL of an XSLT file. It then applies the XSLT transformation in XML to produce the contents of the window. This method carries XSLT asynchronously if not already transferred from the browser user.

  Apart from the above, we can also display a window of information from an overlay for example, an icon (marker). To do that we set a pixel offset between the fixed point and the text of the window, as a third argument. The `GMarker` class includes `openInfoWindow` methods, which handle the pixel offsets automatically based on the size and shape of the icon, and therefore there is no need for the developer to worry about calculating offsets to its application.
- Overlays: The overlays are objects on the map which is "associated" to geographical coordinates, and so moved when the user moves the map or zoom to do this or when visibility is changing the way (Google offers several ways to display a map for example satellite visibility, road map etc). The Google Maps API includes two types of overlays, the markers are icons on the map and polylines lines that are comprised of a number of points.
  - Markers and Icons: The `GMarker` constructor takes as entering an icon and a point and produces a small set of events such as the "click" event, which can then be manipulated in our code. The most difficult part in creating a marker is the definition of the icon. This is difficult because a large number of different images which compose a simple icon in the Maps API. Nevertheless, if one wants a global icon can create a `GMarker` without defining an icon.

Icons are usually kind pins, with a tip that appears in the location specified by `GMarker` constructor. Each icon has (at least) one picture on the scene and an image as a shadow. There are more details about how it should be established icons and can be found in the documentation of the Google Maps API (Google Maps API, 2004).

- Polylines: `GPolyline` constructor takes as a table entry points and creates a series of segments that unite these points in turn given. They can also determine the color, thickness and brightness of the line. The color should be in hexadecimal form as in HTML. More details about the polylines can be found in the documentation of the Google Maps API (Google Maps API, 2004).

- Controls: To use control over the map, such as removal or zoom or any other check, there is a method `addControl`. The Maps API has incorporated the following checks we can use the map:
  - `GLargeMapControl`: A wide range of motion/zoom control used in Google Maps.
  - `GSmallMapControl`: One less motion control/zoom used in Google Local.
  - `GSmallZoomControl`: A small zoom control used in Google Maps to display instructions guidance.
  - `GMapTypeControl`: Check for enabling the user to change the various types of maps (for example Map and Satellite).
- XML and RPC: The Google Maps API offers a "factory method" `XmlHttpRequest` to create objects that "work" in recent versions of Internet Explorer, Firefox, and Safari. It is also the chance to take in parsing an XML document with a static method `GXml.parse`, which takes as input a XML string. Please note only that the Google Maps API does not require the use of XML or `XmlHttpRequest` to work together based solely on a "net" JavaScript/DHTML API.

5.6 KML document structure

Google Earth/Maps use KML as its external spatial and temporal data format. KML is XML-based and has a schema. Using KML, organizations can import geospatial data into Google Earth/Maps and, for instance, overlay it on top of the spatial data provided by Google. A sample KML code is presented. This code places a tethered placemark in a predetermined location. As seen here, KML has a tag-based structure, and Google Earth/Maps acts as a browser of KML files.

Besides an ability to situate placemarks, KML has capabilities to manipulate the following 2-dimensional geometric shapes:

- Points: can be visually represented on the Earth's surface by either icons or labels, or both and positioned at different altitudes.
- Lines: can be positioned at various altitudes similar to points. The support for polylines is included.
- Polygons: can be created in Google Earth/Maps, in 2- or 3-D and can be of solid form or with inner boundaries. As a result, complex 3-D shapes can be rendered at various altitudes.

The appearance of these geometric shapes can be manipulated using the following techniques: defining coordinates, extruding to make 3-D shapes, and grouping into collections. Defining coordinates technique is used by entering coordinates and an elevation above the seal level. Extruding works by positioning an element at the specified position and then using the `<extrude>` tag. Grouping into collection is done using `<MultiGeometry>` tag and is used for organization purposes.

In order to enhance the software package's displaying capabilities, image overlays are also supported. Using this technique, it is possible, for example, to overlay a 3-D blueprint of a construction site over the location site where the construction is planned to occur. Image overlays can be divided into two parts: ground overlays and screen overlays. Ground overlays are images that are static with respect to the Earth's surface whereas screen overlays are fixed to the computer screen and are independent of the underlying geography.

Moreover, KML allows one to define one's own schema if the default schema does not satisfy all needs. The schema requires the following elements:

- Parent element: a base KML schema for inheritance. Uses `<parent>` tag.

- Schema name: uses <name> tag.
- Field declarations: define individual elements.

The following example illustrates the concept of defining a new schema.

```
<Schema>
    <name>States</name>
    <parent>Placemark</parent>
    <SimpleField>
        <name>FIPS</name>
        <type>wstring</type>
    </SimpleField>
    <SimpleField>
        <name>STATE</name>
        <type>wstring</type>
    </SimpleField>
</Schema>
```

In this example two fields are declared: `FIPS` and `STATE`, both of type `wstring` (a UNICODE 16-bit string). The schema itself inherits from `Placemark` element and is, therefore, capable of working correctly with all the `Placemark` attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
    <Placemark>
        <description>
            Tethered to the ground by a customizable tail
        </description>
        <name>Tethethed placemark</name>
        <LookAt>
            <longitude>-122.0856375356631</longitude>
            <latitude>37.42240551227282</latitude>
            <range>305.8880792294568</range>
            <tilt>46.72425699662645</tilt>
            <heading>49.06133439171233</heading>
        </LookAt>
        <visibility>0</visibility>
        <Style>
            <IconStyle>
                <Icon>
                    <href>root://icons/palette-3.png</href>
                </Icon>
            </IconStyle>
        </Style>
        <Point>
            <extrude>1</extrude>
            <altitudeMode>relativeToGround</altitudeMode>
            <coordinates>
                -122.08562,37.42244,50
            </coordinates>
        </Point>
    </Placemark>
</kml>
```

From the sample code above, we can sort out the core elements of which a KML document is comprised.

The root element of a KML file is *kml*. This element is required. It follows the xml declaration at the beginning of the file. The <kml> element may also include the namespace for any external XML schemas that are referenced within the file.

A *Placemark* is a Feature with associated Geometry. In Google Earth, a Placemark appears as a list item in the Places panel. A Placemark with a Point has an icon associated with it that marks a point on the Earth in the viewer.

The *description* element is user-supplied text that appears in the description balloon. It supports plain text as well as a subset of HTML formatting elements, including tables.

In a similar manner, the *name* element is user-defined, text displayed in the viewer as the label for the object.

The *LookAt* element defines a virtual camera that is associated with any element derived from Feature. Its purpose is to position the *camera* in relation to the object that is being viewed. This element is comprised of several other sub-elements; some of them are:

- *longitude*: Longitude of the point the camera is looking at. Angular distance in degrees, relative to the Prime Meridian. Values west of the Meridian range from −180 to 0 degrees. Values east of the Meridian range from 0 to 180 degrees.
- *latitude*: Latitude of the point the camera is looking at. Degrees north or south of the Equator (0 degrees). Values range from −90 degrees to 90 degrees.
- *range*: Distance in meters from the point specified by `<longitude>`, `<latitude>`, and `<altitude>` to the `<LookAt>` position.
- *tilt*: Angle between the direction of the `<LookAt>` position and the normal to the surface of the earth. Values range from 0 to 90 degrees. Values for `<tilt>` cannot be negative. A `<tilt>` value of 0 degrees indicates viewing from directly above. A `<tilt>` value of 90 degrees indicates viewing along the horizon.
- *heading*: Direction (azimuth) of the camera, in degrees. Default=0 (true North). Values range from 0 to 360 degrees.

The *visibility* element is a boolean value, which specifies whether the feature is drawn in the viewer when it is initially loaded. In order for a feature to be visible, the `<visibility>` tag of all its ancestors must also be set to 1.

Style element defines an addressable style group that can be referenced by StyleMaps and Features. Styles affect how Geometry is presented in the viewer and how Features appear in the Places panel of the List view. Its sub-elements are:

- *IconStyle*: Specifies how icons for point Placemarks are drawn, both in the Places panel and in the viewer of Google Earth/Maps.
- *Icon*: Specifies the icon image
- *href*: An HTTP address or a local file specification used to load an icon.

Point elements represent geographic locations defined by longitude, latitude, and (optional) altitude. When a Point is contained by a Placemark, the point itself determines the position of the Placemark's name and icon. When a Point is extruded, it is connected to the ground with a line.

- *extrude*: Specifies whether to connect the point to the ground with a line.
- *altitudeMode*: Specifies how altitude components in the `<coordinates>` element are interpreted. Possible values are (`clampToGround`, `relativeToGround` or `absolute`).
- *coordinates*: A single tuple consisting of floating point values for `<longitude>`, `<latitude>`, and `<altitude>` (in that order). Longitude and latitude values are in degrees, where
    - $-180 \leq$ longitude ≤ 180
    - $-90 \leq$ latitude ≤ 90
    - altitude values are in meters above the sea level

As a final note, it has to be mentioned that some of the elements above, have no effect as far as the visualized data are concerned, in the case of Google Maps viewer. That happens because of the nature of the two viewers. Google Earth is a desktop application, which at least has more exotic features. Google Maps on the other hand, is a web application, so it has to be a more light-weight program running over a stateless web protocol (HTTP), in a simple web browser without any plug-ins that is without some features experienced in similar desktop applications.

5.7 Viewing KML data

Viewing a KML file requires some kind of a custom application. Google, as the main vendor of KML, offers two different types of applications, which a user can use in order to get her data visualized. Google Earth is a desktop application which comes with many features. On the other hand, Google Maps is a Web service, offered by Google, with which a user can use only hers' JavaScript-enabled web browser to get the results. As a web application, Maps has less features in regard to its Earth brother.

5.7.1 Google Earth

Formerly known as Earth Viewer, Google Earth was developed by Keyhole, Inc., a company acquired by Google in 2004. The product was renamed Google Earth in 2005 and is currently available for use on personal computers running Microsoft Windows 2000, XP, or Vista; Mac OS X 10.3.9 and above; Linux and FreeBSD. In addition to releasing an updated Keyhole based client, Google also added the imagery from the Earth database to their web based mapping software (Figure 5-1).

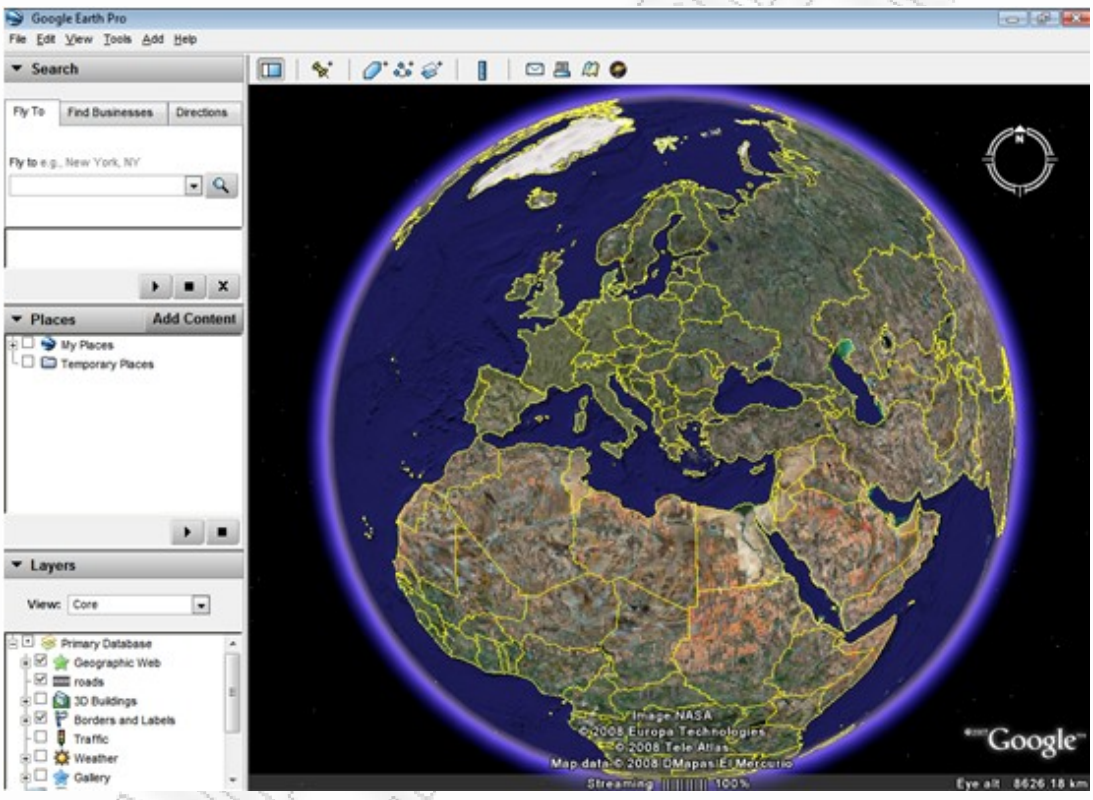


Figure 5-1: Google Earth UI

The viewer will show houses, the color of cars, and even the shadows of people and street signs. The degree of resolution available is based somewhat on the points of interest, but most land (except for some islands) is covered in at least 15 meters of resolution. Las Vegas, Nevada and Cambridge, Massachusetts include examples of the highest resolution, at 15 cm (6 inches). Google Earth allows users to search for addresses (for some countries only), enter coordinates, or simply use the mouse to browse to a location.

Google Earth also has digital elevation model (DEM) data collected by NASA's Shuttle Radar Topography Mission. This means one can view the Grand Canyon or Mount Everest in three dimensions, instead of 2D like other map programs/sites. Since 2006, the 3D views of many mountains, including Mount Everest, have been improved by the use of supplementary DEM data to fill the gaps in SRTM coverage. In addition, Google has provided a layer allowing one to see 3D buildings for many major cities in the US and Japan.

Many people using the applications are adding their own data and making them available through various sources, such as BBS or blogs. Google Earth is able to show all kinds of images overlaid on the surface of the earth and is also Web Map Service client. Google Earth supports managing three-

dimensional geospatial data through KML. It is available in a free version, and in licensed versions for commercial use.

Google Earth has the capability to show 3D buildings and structures (such as bridges), which consist of users' submissions using *SketchUp*, a 3D modeling program. In prior versions of Google Earth (before Version 4), 3D buildings were limited to a few cities, and had poorer rendering with no textures.

Many buildings and structures from around the world now have detailed 3D structures; including (but not limited to): U.S., Canada, India, Japan, United Kingdom, Germany, Pakistan, Amsterdam and Alexandria. Three-dimensional renderings are available for certain buildings and structures around the world via Google's 3D Warehouse and other websites.

Furthermore, Google Earth offers a time-slice bar, which is a very handy tool when a user has to visualize moving objects information. With this tool in hand and a corresponding KML file as source, it is possible to examine an object's movement in a time range.

5.7.2 Google Maps

Google Maps is a map service that a user views in your web browser. Depending on user's location, she can view basic or custom maps and local business information, including business locations, contact information, and driving directions. She can also view satellite image with or without map data of a desired location that can be zoomed and panned (Figure 5-2).

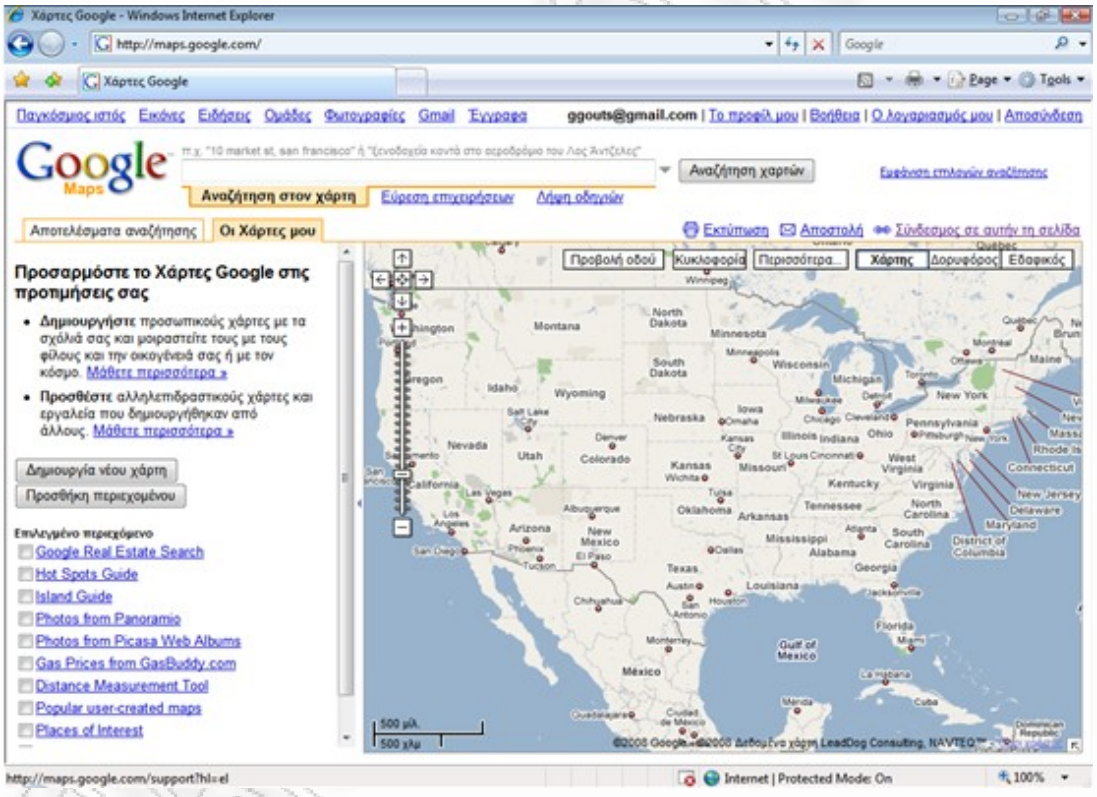


Figure 5-2: Google Maps UI

Google Maps is a first class solution for displaying the contents of a KML file. The easiest way to do so is to go to the Google Maps page and enter the URL of the KML file as though it were an address or other search term. Such a query results in the following URL:

```
http://maps.google.com?q={kml-url}
```

For example, feeding the KML for the bookstore map back into Google Maps, we get the following:

```
http://maps.google.com/maps?q=http:%2F%2Fmaps.google.com%2Fmaps%2➥
Fms%3Ff%3Dq%26om%3D1%26ie%3DUTF8%26msa%3D0%26output%3Dnl%26msid%3➥
D116029721704976049577.0000011345e68993fc0e7
```

Google Maps is an incredibly useful KML renderer. First, someone can test KML files without having access to Google Earth. Second, a user can let others look at the content of KML files without requiring them to have Google Earth installed. We should be aware, however, of two caveats in using Google Maps to render KML:

- Google Maps does not implement KML in total; so don't expect to replace Google Earth with Google Maps for working with KML and
- Google Maps caches KML files that a user renders with it. That is, if we are using Google Maps to test the KML that we are changing, be aware that Google Maps might not be reading the latest version of our KML file.

5.7.3 Other viewing applications

The acceptance of KML specification for geographic information is so great, which led many other companies, providing congener services, to adopt it as a secondary option. That is, Microsoft Virtual Earth and Yahoo! Maps, offers the ability to their users, to view KML files using their services.

Furthermore, a developer can always create a custom web application, so to visualize hers KML information, using the Google Maps API as a core component.

5.8 KML critique

KML is a powerful XML-type language used to extend the applicability and usefulness of Google Earth/Maps package. However, its disadvantage stems from the fact that it is based on the rigid schema that KML documents must conform to in order to work correctly in Google Earth/Maps. In addition, tag names are ambiguous because of the inherent vagueness of a human language. Therefore, KML is ill-suited to provide spatial data interoperability on the semantic level. M-Language, on the other hand, solves the ambiguity problem and is well positioned to offer spatial data interoperability solutions for disparate data sources.

5.9 Conclusion

KML is a really powerful technology for visualizing both spatial and temporal data in the Web. It is a perfect tool for publishing geographic data in the form of maps. Its XML nature in conjunction with a well developed API makes this kind of files really portable and easily to be processed. Finally, usage of free web or desktop applications for viewing this information completes the overall image.

6 The eXtensible Stylesheet Language Transformations (XSLT)

6.1 Introduction

The main focus of this chapter is examining the ways of visualizing XML data and how Extensible Stylesheet Transformations (XSLT) contributes in that process. At the same time it explains how XSLT is used to transform GML data into Keyhole Markup Language (KML).

6.2 EXtensible Stylesheet Language (XSL)

Since XML does not use predefined tags or include formatting information, a generic XML processor that reads an XML document has no idea what is *meant* by the document and the form which is desired to present it. Therefore, there must be an additional document that provides information on how to present or otherwise process the XML, and that is XSL.

XSL is a specification being developed within W3C for applying formatting to XML documents in a standard way. The specification defines that XSL is a language for expressing stylesheets. Stylesheets are used to describe how the content of a given structured document should be presented; that is how the source content should be styled, laid out, and paginated onto some presentation medium such as a window in a Web browser or a hand-held device, or a set of physical pages in a catalog, report, pamphlet or book (Joshi, 2007).

The Extensible Stylesheet Language consists of three component languages which are described by three W3C recommendations. These are XSL Transformation – XSLT (Clark, 1999), XSL Formatting Objects - XSL-FO (Adler, et al., 2001), and XML Path Language – XPath (Clark, et al., 1999). The XSL Transformation and XSL Formation Objects can function independently of each other (Adler, et al., 2001).

6.2.1 XSL Transformation (XSLT)

XSLT is the most important part of the XSL Standards. It provides elements that define rules for how one XML document is transformed into another XML, HTML or text document. If the transformed document is in XML, it may use the markup and DTD of the original document or it may use a completely different set of elements. XSLT can add new elements into the output file, remove existing elements, rearrange and sort elements, test and make decisions, and a lot more through appropriate stylesheets.

The transformation can be performed in three primary ways. First, XML document and associated stylesheet both can be served to the browser (formatter), which then transforms the document as specified by the stylesheet. Otherwise a server can apply the XSLT stylesheet to the XML document and send the transformed document to the user. And the other possibility is, an XSLT processor transforms the original XML document into specified format according to the stylesheet before the document is placed on the server. Here both server and user only deal with the transformed document.

Each of these three approaches uses different software, although they all use the same XML document and XSLT stylesheet. This thesis emphasizes on the third approach that is more suitable for achieving the research objectives.

6.2.2 XSL Formatting Objects (XSL-FO)

The XSL-FO is an XML application that describes how pages will look when presented to a reader on screen or paper. It describes a rendering vocabulary capturing the semantics of formatting information for paginated presentation. An XSLT stylesheet can be used to transform XML document in semantic vocabulary into a new XML document that uses the XSL-FO presentational vocabulary (Evjenet, 2007).

6.2.3 XML Path Language (XPath)

XPath is a language for referencing specific parts of an XML document, essentially for cases where it is needed to say exactly which of a document are to be transformed by XSLT. XPath is designed to be used by both XSLT and XPointer which defines an addressing scheme for individual parts of an XML document. XPath has an extensible string-based syntax that describes the *location path* between parts of a document or documents using common *path/file* file system syntax.

6.3 Tree and nodes

A tree is a data structure composed of connected nodes beginning with a top node called the root. Therefore every well-formed XML document is a tree. The root is connected to its child nodes, each of which is connected to zero or more children of its own, and so forth. The most useful property of a tree is that each node and its children also form a tree. Thus, a tree is a hierarchical structure of trees in which each tree is built out of smaller trees. XSLT models an XML document as a tree that contains seven kinds of nodes: The root, Elements, Text, Attributes, Namespaces, Processing instructions, and Comments. The DTD and document type declaration are specifically not included in this tree (Evjenet, 2007).

6.4 XSLT stylesheet

An XSLT stylesheet is basically a set of rules expressed in Extensible Stylesheet Language for transforming XML documents. In case of data visualization, the role of XSLT stylesheet is to transform the XML data content into a presentation format such as HTML/XHTML, KML, XSL-FO, text or any other structured format. However, traditional stylesheets encode information about the appearance of text and the layout of content. But in the context of GIS, XML data found in GML has to be presented in a graphical format like Keyhole Markup Language which is an XML based language for describing 2D graphics.

6.5 XSLT stylesheet structure and elements

An XSLT stylesheet consists of a set of rules called templates. A template contains a set of template rules which has two parts; a pattern which is matched against nodes in the source tree and a template which can be instantiated to form part of the result tree. This allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures.

As any other XML documents, an XSLT stylesheet begins with an xml declaration. The next line is either `<xsl:stylesheet>` element or `<xsl:transform>` element which are completely synonymous defines the start of stylesheet and the root element and declares the document to be an XSLT stylesheet. This element must have a version attribute to indicate the version of XSLT in which the stylesheet is based. The XSLT namespace attribute is given by `xmlns:xsl="http://www.w3.org/1999/XSL/Transform` (by convention `xsl` prefix is used to map the XSLT namespace). The elements that occur as a child of an `<xsl:stylesheet>` element are called top level elements. The basic structure of a stylesheet is as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- Templates go here -->
</xsl:stylesheet>
```

6.5.1 Templates

Template rules defined by <xsl:template> elements are the most important part of an XSLT stylesheet. Each <xsl:template> element contains rules to apply when a specified node is matched in source document. These rules describe the contribution that the matched elements make to the output document.

The rules may contain both texts that will appear literally in the output document and XSLT instructions that copy from the input XML document to the result.

Following template works on `gml:Point` nodes in the input document and extract the content of `coordinates` child node. It then creates a new `<Point>` element in output document with `coordinates`' contents. It uses another template to work on `gml:Name` node.

```
<xsl:template match="gml:Point">
    <xsl:element name="Point">
        <xsl:element name="coordinates">
            <xsl:value-of select="." />
        </xsl:element>
    </xsl:element>
    <xsl:apply-templates select="gml:name" />
</xsl:template>
```

The match attribute in `<xsl:template>` element specifies which node of the input document the template is instantiated for. It can also be used to define a template for a whole xml document. (i.e. `match="/"` defined the whole document). When the XSLT processor reads the input document, the root is the first node it finds and then the rules match that root node are carried out.

To get beyond the root, `<xsl:apply-template>` element have to be used. By including this element, the formatter is instructed to compare each child element of the matched source element against the templates in the stylesheet, and if a match is found, output the template for the matched node. The `xsl:apply-template` is supplied with select attribute to designate the children to be selected.

6.5.2 Matching nodes

The match attribute of the `<xsl:template>` element supports a complex syntax that allows to express exactly which nodes are needed and which are not needed to match. The match patterns enable to match nodes by element name, child elements, descendants, attributes, element id, comments, processing-instruction, text, and or operator and, as well as by making simple tests on some of these items.

6.5.3 Selecting nodes

The select attribute is used in `xsl:apply-templates`, `xsl:value-of`, `xsl:for-each`, `xsl:copy-of`, `xsl:variable`, `xsl:param` and `xsl:sort` to specify exactly which nodes are operated on. The value of this attribute is an expression written in the XPath language. The XPath language provides a means of identifying a particular element, group of elements, text fragment, or other part of an XML document.

The expressions are a superset of the match patterns mentioned above. They are not limited to specifying the children and descendants of the current node. XPath provides a number of axes that can be used to select from different parts of the tree relatives to some particular node in the tree called context node. In XSLT, the context node is normally initialized to the current node that the template matches, though there are ways to change this. Figure 6-1demonstrates the axes provided by XPath.

6.5.4 Named templates

The `<xsl:template>` element can have a name attribute by which it can be explicitly invoked, even when it isn't applied directly. Such templates are called named templates. Named templates are used to repeat a template rule inside other template rules and they enable to include data from the place where the template is applied. The `<xsl:call-template>` element is used to call a named template and the value of its name attribute provides the name of the named template.

6.5.5 Content of output

In an XML document transformation, it is often necessary to include new elements, attributes, processing instructions, comments, etc. in the output document in order to conform to a desired output structure. For instance, the output of an XSLT stylesheet designed to transform GML content into KML, should comply

with KML specifications. This is accomplished with the corresponding xsl elements such as `xsl:element`, `xsl:attribute`, `xsl:processing-instruction`, `xsl:comment`, and `xsl:text` elements and attribute value templates.

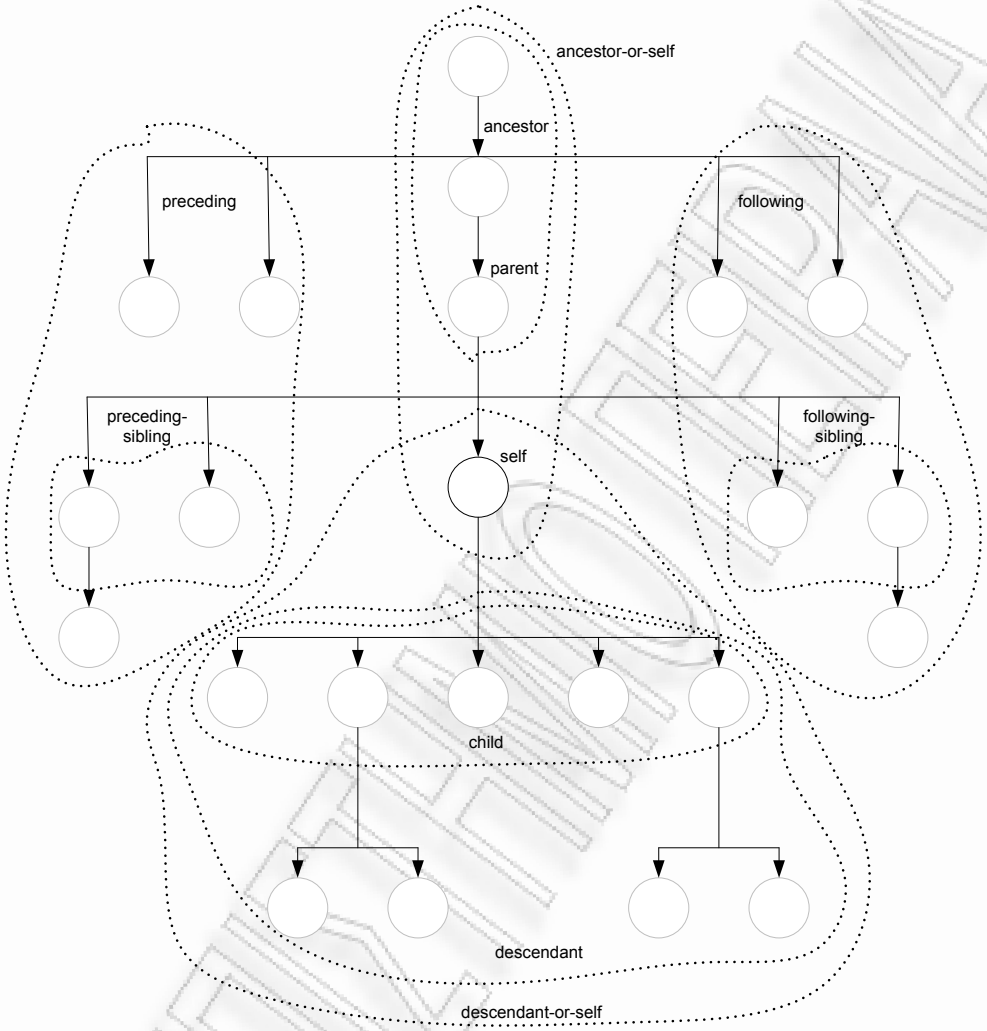


Figure 6-1: Axes provided by XPath (Sun Microsystems, 1994)

Attribute value templates copy data from the input document to attribute values in the output. The `<xsl:element>` element inserts an element into the output document. The name of the inserting element is given by the value of name attribute and the content by the content of the `<xsl:element>` element. The `<xsl:attribute>` element defines an attribute name and value and inserts them to the elements in output document. Therefore this must appears as a child of either an `<xsl:element>` or a literal element, before any other content in those elements. When the same group of attributes is applied in many different elements, such as an attribute set can be defined as a top level element with `<xsl:attribute-set>` and inserted wherever necessary with `<xsl:use-attribute-sets>`. The `<xsl:processing-instruction>` element places a processing instruction in the output document. The target of the processing instruction is specified by name attribute and the content of the output `<xsl:processing-instruction>` element become the contents of the processing instruction itself. The `<xsl:comment>` and `<xsl:text>` elements insert comments and text respectively to output document.

6.5.6 Output methods

Most of the XSLT processors support three types of output methods XML, HTML and Text. The XSLT processor behaves differently depending on which of these output methods stylesheet uses. The output

method is defined by the top-level <xsl:output> element and it's method attribute specifies the output method which is xml by default. It also has a number of attributes that allow changing the prolog, indenting, and CDATA sections in the output document as well.

The following four attributes in <xsl:output> element format the XML declaration in the output document in case the output method is xml. The omit-xml-declaration attribute can have the value yes or no and when the value is yes, no xml declaration is included in the output document. At present, the default version of the XML declaration is 1.0 and it's the only value allowed. The version attribute of <xsl:output> element allows to change the version used in XML declaration accordingly in the future. The encoding attribute sets the encoding system in output document and its value can be any encoding name registered with the Internet Assigned Numbers Authority. The standalone attribute can set the standalone attribute and the value yes or no in xml declaration of the output document.

The XSLT provides no elements for building an internal DTD subset for the output document. However, it provides two attributes of <xsl:output> element that can be used to include a DOCTYPE declaration that points to an external DTD. These are doctype-system and doctype-public. The first inserts a SYSTEM identifier for DTD and the second a PUBLIC identifier.

The indent attribute of <xsl:output> element has two values yes and no. When the attribute has the value yes, then the processor is allowed to insert extra white space into the output to make the document printable and more readable.

The standard XSLT does not allow inserting CDATA selections at arbitrary locations in XML documents produced by XSL transformations. However it can be specified that the text content of a particular element in input document to be placed as a CDATA section in output document by placing the name of the element whose text content should be wrapped in CDATA delimiters in the cdata-section-elements attribute of the <xsl:output> element. For example <xsl:output cdata-section-element="SCRIPT" /> says that the content of the SCRIPT element in input document should be wrapped in a CDATA section in output document.

The attribute media-type of <xsl:output> element specifies the MIME media type of the output document. Mostly this will have the value text/xsl, but could be text/html or text/plain for the HTML or text output methods. This is important to the environment in which the XML document exists, but not so to the XML document itself.

6.6 Combining stylesheets

XSLT provides two mechanisms to combine stylesheets; an inclusion mechanism that allows stylesheets to be combined without changing the semantics of the stylesheets being combined and an import mechanism that allows stylesheets to override each other.

6.6.1 Importing

The <xsl:import> element is a top level element whose href attribute provides the URI of a stylesheet to import. All <xsl:import> elements must appear before any other top-level element in the <xsl:stylesheet> root element. Rules in the imported stylesheet may conflict with rules in the importing stylesheet. If so, rules in the importing stylesheet take precedence.

6.6.2 Inclusion

The <xsl:include> is a top-level element that copies another stylesheet into the current stylesheet at the point where it occurs. Its href attribute provides the URI of the stylesheet to include. Unlike in above case, rules included by <xsl:include> elements have the same precedence in the including stylesheet.

6.7 Embedding stylesheets

An XSLT stylesheet can directly be embedded in the XML document it applies to. In such a case, the `<xsl:stylesheet>` element must appear as a child of the XSLT document element, rather than a root element itself and have an `id` attribute giving it a unique name. This `id` attribute would appear as the value of `href` attribute in the `xml-stylesheet` processing instruction following the fragment identifier separator # in the XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xml" href="#mystyle"?>
<Root_Element>
 <xsl:stylesheet
     version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     id="mystyle">
     <xsl:template match="/">
         <!-- Template content goes here -->
     </xsl:template>
     <!-- Other templates go here -->
     <!-- Don't display the style sheet itself or its descendants -->
     <xsl:template match="xsl:stylesheet"/>
 </xsl:stylesheet>
 <!-- Rest of xml data elements go here -->
 ...
</Root_Element>
```

6.8 Creating an XSLT stylesheet

An XSLT stylesheet must be a well-formed XML document and should comply with XSLT specification, which describes allowed syntax and vocabulary. The content of the stylesheet entirely depends on the input document structure (Schema) and the required output structure. The following general steps in XSLT stylesheet creation are based on the assumption that input is a GML document and output target is KML.

1. XML declaration.

   ```
   <?xml version="1.0" encoding="utf-8"?>
   ```

2. Root element of Stylesheet including version and namespace attributes (including all namespaces found in input document).

   ```
   <xsl:stylesheet
         version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
         xmlns:gml="http://www.opengis.net/gml"
         xmlns:msxsl="urn:schemas-microsoft-com:xslt"
         xmlns:ext="http://goutsidis.gr/extension"
         exclude-result-prefixes="gml msxsl ext"
   >
   ```

3. Declaring top-level elements.

   ```
   <xsl:output
         method="xml"
         version="1.0"
         encoding="utf-8"
         indent="yes"
         media-type="application/vnd.google-earth.kml+xml"
   />
   ```

   This stylesheet fragment outputs the following declarations in the output document.

   ```
   <?xml version="1.0" encoding="utf-8"?>
   ```

4. Template rule matching the root `<gml:featureCollection>` element of the input document. This may include many computations required to determine attribute values for the root element of the output document and other templates in the stylesheet.

```
<xsl:template match="/gml:featureCollection">
```

5. Create Root element and connected attributes in output document.

```
<xsl:element
    name="kml"
    namespace="http://earth.google.com/kml/2.2">
```

An example for the output of this stylesheet fragment is.

```
<kml xmlns="http://earth.google.com/kml/2.2">
```

6. Create Top-level elements in output document.

```
<xsl:element name="Document">
      <xsl:element name="Folder">
      <xsl:element name="name">
            <xsl:text>VisualHermes</xsl:text>
      </xsl:element>
      <xsl:element name="Style">
            <xsl:attribute name="id">
                  <xsl:text>blueLine</xsl:text>
            </xsl:attribute>
            <xsl:element name="LineStyle">
                  <xsl:element name="color">
                        <xsl:text>ffff0000</xsl:text>
                  </xsl:element>
                  <xsl:element name="width">
                        <xsl:text>3</xsl:text>
                  </xsl:element>
            </xsl:element>
      </xsl:element>
```

It creates the following series of elements in the output document.

```
<Document>
  <Folder>
    <name>VisualHermes</name>
    <Style id="blueLine">
      <LineStyle>
        <color>ffff0000</color>
        <width>3</width>
      </LineStyle>
    </Style>
```

7. Apply all the available templates, according to the elements, that can be found, that is matching, to the original document.

```
<xsl:apply-templates />
```

For example, the following template is applied if a `<gml:Point>` element is present in the GML document.

```
<xsl:template match="gml:Point">
      <xsl:element name="Point">
            <xsl:element name="coordinates">
                  <xsl:value-of select="." />
            </xsl:element>
      </xsl:element>
</xsl:template>
```

The output of such a template will be as below.

```
<Point>
    <coordinates>
        23.86301,37.99958
```

```
        </coordinates>
    </Point>
```

8. Close root element in output document.

```
</xsl:element>
```

9. Close Template rule matching root element of the input document.

```
</xsl:template>
```

For the current thesis' needs, there are more templates implemented in the corresponding XSLT instructions file. Furthermore, a series of custom C# functions had to be implemented, for the transformations to be accomplished.

6.9 XSLT processors

In order to perform the transformations in an XSLT stylesheet, another software program called XSLT processor has to be employed, because the source XML document and the stylesheet both are plain text documents. XSLT processor takes as input an XML document and style sheet to convert the XML document to whatever XML, HTML or Text format. In the transformation, the processor walks through the XML document tree, looking at each node in turn, compares it with the pattern of each template rule in the style sheet. When the processor finds a node that matches a template rule's pattern, it outputs the rule's template. At present, many XSLT processors, which conform to XSLT 1.0 specification, have been developed, and Instant SAXON (Kay, 2002) and XMLSpy (Altova, 2005) are among them. Among them, it is possible for a developer to create a custom processor using known development environments such as Java (Sun Microsystems, 1994), Microsoft .NET Framework (Microsoft Corporation, 2008) etc.

6.10 Conclusion

XML content can be presented in many ways. Extensible Stylesheet Language Transformation is the W3C specification for reformatting XML documents. XSLT stylesheet consisting transformation rules in templates accomplishes this transformation through an XSLT processor. GML data can be transformed into KML by this method. The structure of the XSLT stylesheet entirely depends on the structure of input and output documents.

7 VisualHERMES Wrapper

7.1 Introduction

In previous chapters, the XML-based open standards that are used to model geographic information and transforming them into visualizations were discussed. This chapter covers the implementation phase of the VisualHERMES wrapper, whose main purpose is to transform relational data (coming from HERMES) into GML and vice versa and in a final stage to produce the corresponding KML files via XSLT rules.

First, the set of requirements is discussed. Following, is a general description of the overall system as well as a more in depth discussion about the components the system is comprised of. Then the processing of an XSLT stylesheet through a XSLT processor is described. Finally, the process of GML queries is adduced, as far as the validation and parsing of them are concerned. Problems and limitations encountered during the implementation are discussed. In this chapter the basic skeletons of all the available files are provided. For more information on a real data set and produced files as well, consult the corresponding *Case Study* chapter.

7.2 Requirements

The standardization of data exchanged between different parts of the system, is made with the use of GML standard, given the fact that is geographic information. GML is an XML standard formatting of data, which allows the transfer and storage of geographic information, including both spatial and time elements of geographical entities, as well as handling for moving objects. The GML standard is designed to support the maximum cross-functional available and this is achieved through the provision of basic geometric labels (all systems that support GML standard, use the same geometric labels), a common data model (entities and properties) and a mechanism for creating and sharing schemes applications (application schemas); thus enabling the modeling of the semantics of spatial and temporal information.

The interoperability supported by the intermediate level manipulation of data to and from the trajectory DBMS, is completed around providing a shape (schema), with which both incoming and outgoing data is required to comply. At the level of development, this translates to distinguish the operation of the intermediate level in two parts:

1. Data, from which queries are comprised of and which in turn have to be executed from the DBMS, must:
    - Comply with the provided schema (GML Schema),
    - Be valid as far as semantics is concerned and
    - Be transformed into SQL clauses

    so as their execution to be successful and
2. Data, from which the results produced by queries' execution, are comprised of must:
    - Be transformed according to the provided, by the intermediate level, schema

    so as a client can apply any further analysis procedures to a dataset, which meets a well known and commonly accepted format (Figure 7-1).

Finally, for the results' visualization process and the presentation of them via third-party vendor engines (such as Google Earth/Maps) to be accomplished, it is essential to apply a valid transformation, so as the engines mentioned above, to be able to handle the incoming data. In the current thesis, results have to be transformed using the KML specification. Since both GML and KML data are based on the more general XML data tagging technology, this transformation can be accomplished via XSLT transformation rules (Figure 7-2).

Of course, all the operations mentioned above, from a development perspective, can be integrated under a web application. This application can provide a user with a simple and intuitive working environment, in which someone can construct and send queries, as well as retrieve the corresponding GML results and the visualized KML data, via a web browser, like Microsoft Internet Explorer, Mozilla Firefox etc. In this way, system's intrinsic operations are encapsulated, making the user capable to use the system without even being aware of SQL.

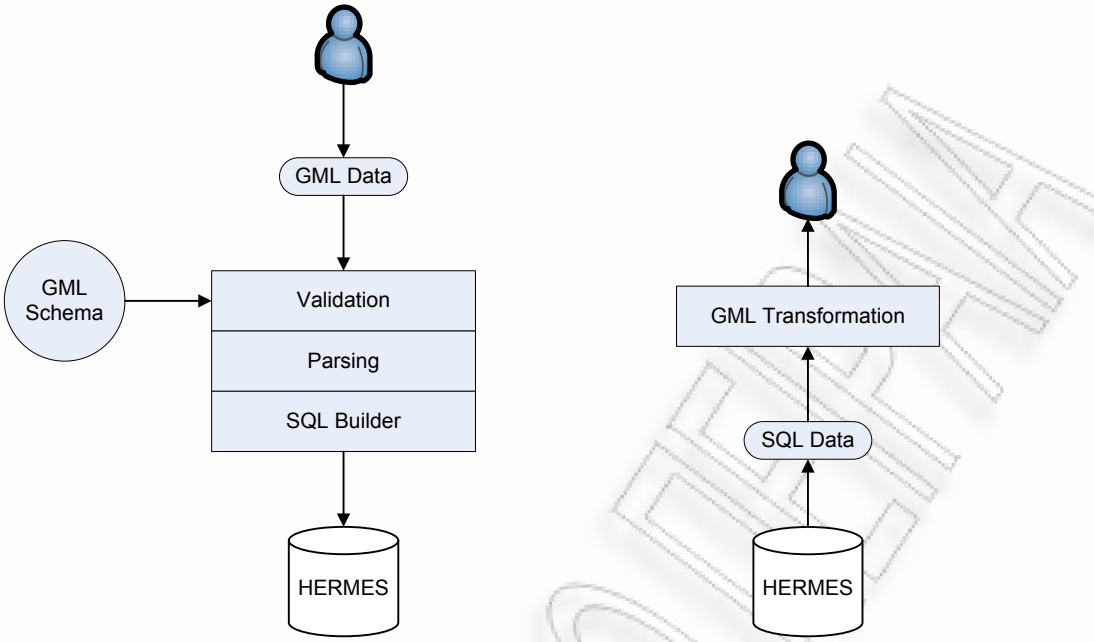


Figure 7-1: GML query and SQL result

As far as the queries' construction and sending are concerned, there are three possible ways available to the user:

1. Manually construct a SQL query
2. Automatic query construction through a bunch of selections on the appropriate option fields and
3. GML file upload, which meets the predefined schema and contains all the appropriate information.

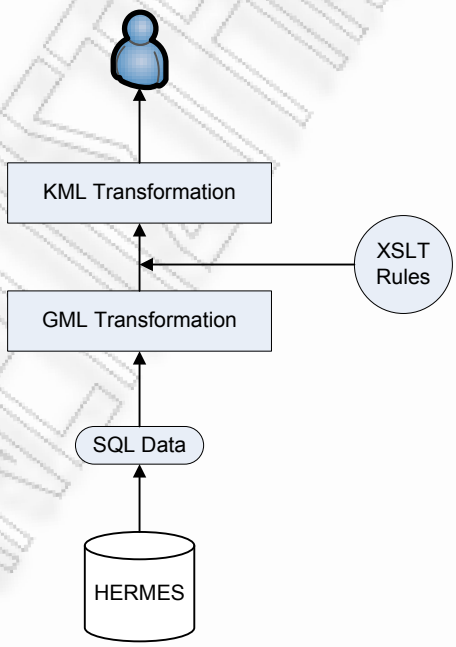


Figure 7-2: KML result

The available results a user is able to receive after a successful query's execution, using any of the above methods, are the following:

1. GML results file: User receives a hyperlink, through which she can download the final result, in her local computer and

2. KML results file: User receives a second hyperlink, which navigates her to another web page of the application. From there, the user can view the final result visualized on a map, using Google Maps (Google Maps, 2008) and/or Google Earth (Google Earth, 2008).

7.3 Design

The effort of extending HERMES's interoperability, led to a new tool-application, namely VisualHERMES. This tool is able to be installed and operate without the need of any modifications to the current system (Figure 7-3). In other words, VisualHERMES is an alternative interface to the current system, providing features such as results retrieval based on GML specification. It is able to visualize these results via Google's Maps/Earth engines too.

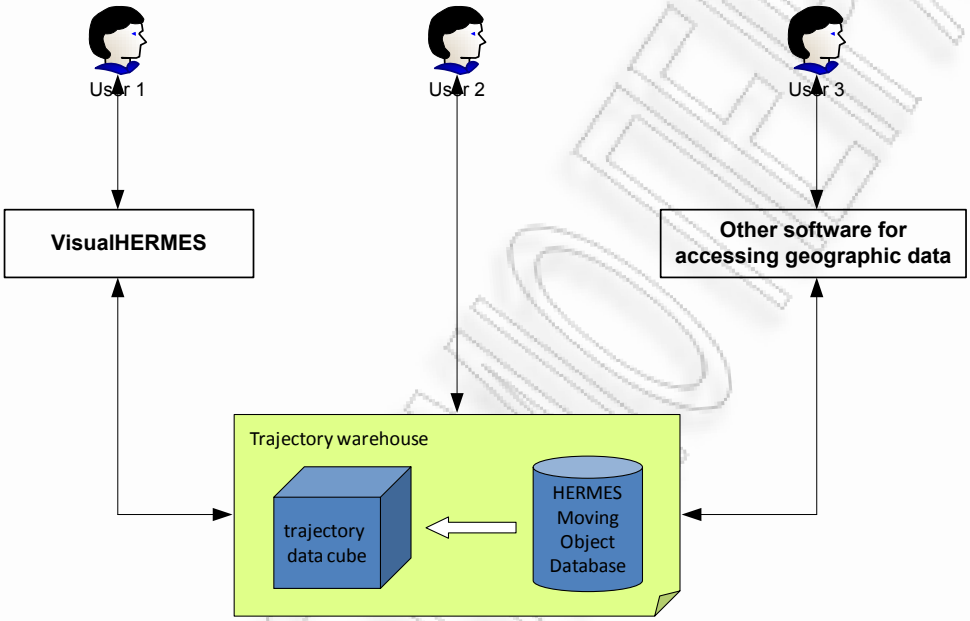


Figure 7-3: VisualHERMES as an alternative interface

7.3.1 Application architecture

VisualHERMES is based on the 3-tier architecture, for application development, strategy. The selected approach gives the system great capabilities for future extensions. Furthermore, any changes can be targeted to a specific layer, leaving the rest of the overall system intact. Another powerful advantage, the selected approach provides to the system, is that all levels are completely isolated from each other. That is, any layer can completely be replaced with a new and possible better one, without requiring any further modifications (Figure 7-4).

Each level's development, except the presentation's one, follows the Dynamic Linked Libraries (DLL) guidelines of Windows operating systems. This approach provides one more advantage to the overall system, as in this way, the presentation layer is loosely-coupled with the other two layers. Thus, other types of applications can be developed, such as desktop applications, Personal Digital Assistant device (PDA) applications[1], Web Services etc., which will be able to provide the same functionality, as the current wrapper does, although they will target to different platforms or even devices.

7.3.2 Data Access Layer

Data Access Layer (DAL) is responsible for accomplishing all the required operations for connecting the wrapper to the database and handling the SQL queries and results as well. In this application's layer, two important operations take place:

[1] Consult *Open issues* (Section 9.1).

1. Sending users' SQL queries to HERMES and
2. Getting the primitive results from HERMES's responses.

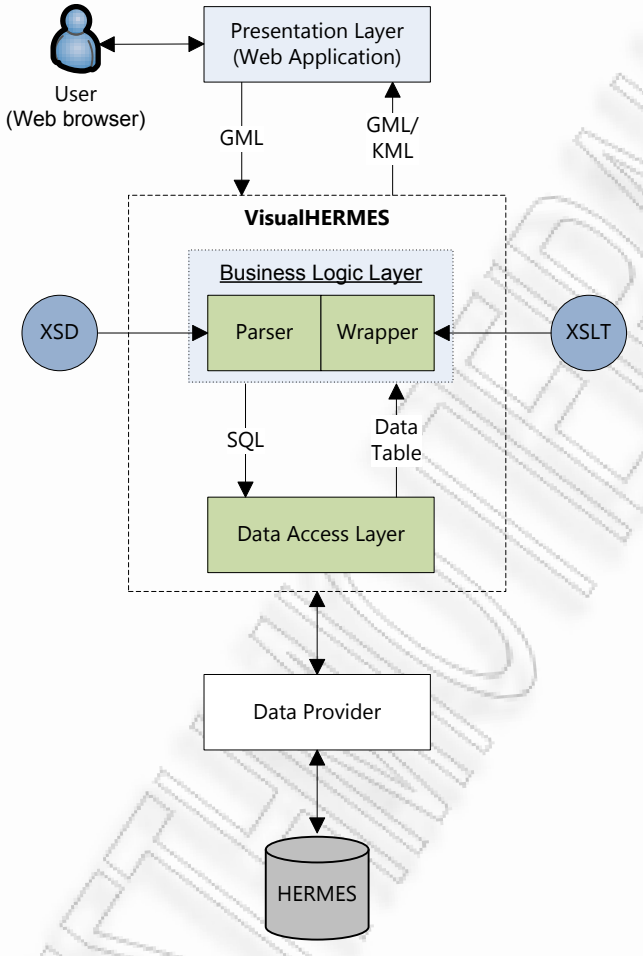


Figure 7-4: VisualHERMES 3-tier architecture

It is conceivable, that for the current layer to be able to communicate with HERMES system, it has first to establish the corresponding connection. Such an operation is accomplished by this layer using an appropriate data provider.

As far as the primitive results are concerned, it has to be mentioned that the wrapper receives moving object trajectories data from a semantic perspective. These trajectories are composed of two core components; the geographic coordinates and the time interval during which censure was made. Furthermore, we have to notice, that since these information are about trajectories the presence of a pair of points is required. That is, we need a point and the corresponding timestamp for the moving object's start and another point and timestamp for the stop. Based on the above, the wrapper gets a response with data, which meet the following pattern:

$$(X_1,\ Y_1)\ -\ (X_2,\ Y_2)\ \#\ TimeStamp_1\ -\ TimeStamp_2$$

Each response's result set may contain multiple pairs for each moving object; in this way a stream of subsequent points and timestamps is constructed.

The SQL queries the wrapper can send to HERMES are typical queries, based on the SQL grammar specification for building queries. They are able to make use of any spatial and/or temporal operators available for representing the desired query. Some of these operators are provided by Oracle's Spatial package itself and some others by HERMES extension. The wrapper makes simple use of these operators, without any applying further modifications on them.

Furthermore, DAL is responsible for converting primitive data into structures capable for being processed by the next layer. In this way, the loosely-coupling between these two layers is achieved. In other words, the current layer can produce the desired results in any way it *wants*. The only constraint is that is always has to end up with the same structures.

For the wrapper to be as flexible as possible, DAL does not make use (from the actual code perspective) of any specific data provider. In contrast, we have used generic code, which is able to adapt itself to any possible provider. This provider-agnostic pattern makes this layer more adaptive and interoperable than using any other design practice. The required data provider is declared via an external configuration file.

7.3.3 Business Logic Layer

The Business Logic Layer (BLL) is in charge of modeling the primitive data, received from DAL, to business objects, capable to pass through the required transformations, so as to become proper for the final results construction. Furthermore, in this layer the GML queries' transformation into SQL clauses is accomplished. After that, the transformed SQL query is able to be sent to DAL for its final execution (Figure 7-5).

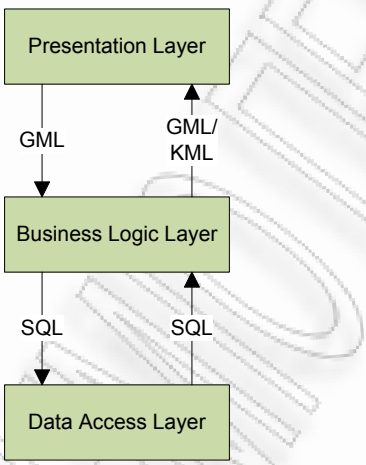


Figure 7-5: Business Logic Layer double role

For the overall system's needs, we have implanted two core business objects (in the nature of classes). If combined together, it is possible to achieve the required trajectories' representation for our moving objects. These are the following:

1. `Point` and
2. `Trajectory`

Each point has a series of characteristics (attributes) defining it, which are as follows:

1. Geographic coordinates and
2. Timestamp

Consider a single point object with longitude (X) 23.54 and latitude (Y) 37.43 marked at 5:00 pm on June 15, 2008. This point is represented as:

```
(23.54, 37.43) # 2008/06/15 17:00:00
```

In a similar manner, a set of such points is able to represent a moving object's trajectory with the following characteristics:

1. Moving object's ID,
2. Moving object trajectory's ID and
3. The points set, the trajectory is comprised of.

For example, consider the single point presented above as a starting point of a trajectory and a second similar point with longitude 23.67 and latitude 37.5 marked at 5:15 pm on June 15, 2008. These two points comprise a single trajectory object with ID 6 of a moving object with ID of 0420 represented as:

```
0420, 6, (23.54, 37.43) – (23.67, 37.5) # 2008/06/15 17:00:00 –
                          2008/06/15 17:15:00
```

From the above it is conceivable that a trajectory object can be comprised of several starting and stopping points.

Using the already implemented functions from BLL, it is possible to transform these objects into files, the contents of which meet the GML and KML specifications.

As mentioned above, the layer's main role is double (Figure 7-5); it formats the results and transforms GML queries into SQL clauses. For the *GML-to-KML* transformation's needs, we have implemented a set of additional business objects, in the nature of classes as well, making SQL queries' representation possible, as well as an intuitive type of query. The implemented classes are:

1. `QueryType`: The query's type literal identification (i.e. Trajectory, Spatial, Temporal etc.),
2. `TimePoint`: Represents a timestamp,
3. `TemporalWindow`: Represents a time window, comprised of two TimePoints
4. `SpatialWindow`: Represents a geographic rectangle, composed of two Points; these points represent the upper-right and lower-left corners respectively and
5. `Query`: Represents the clauses appearing in a SQL query such as type of query, object's ID, trajectory's ID, temporal window etc.

A combination of the query's type along with the additional clauses of it makes the application able to build the appropriate SQL query, via a set of already implemented functions. The final step involves the DAL, through which the newly built query is sent to HERMES.

7.3.4 Presentation Layer

As mentioned above, we are able to use different applications, platforms or even devices to access VisualHERMES wrapper. In this way a user is provided with an integrated working environment, through which she is able to send queries and receive results as well. For this reason, a web application based on Microsoft's ASP.NET (Microsoft ASP.NET, 2008) technology, has been implemented. This interface provides the user the ability to connect to the available services using a simple web browser (i.e. Internet Explorer, Mozilla Firefox etc.). No further plug-ins or custom software is required.

7.4 Application modules

Below, we present with the modules which compose each system's layer. Figure 7-6 depicts the business objects which *live* in our system.

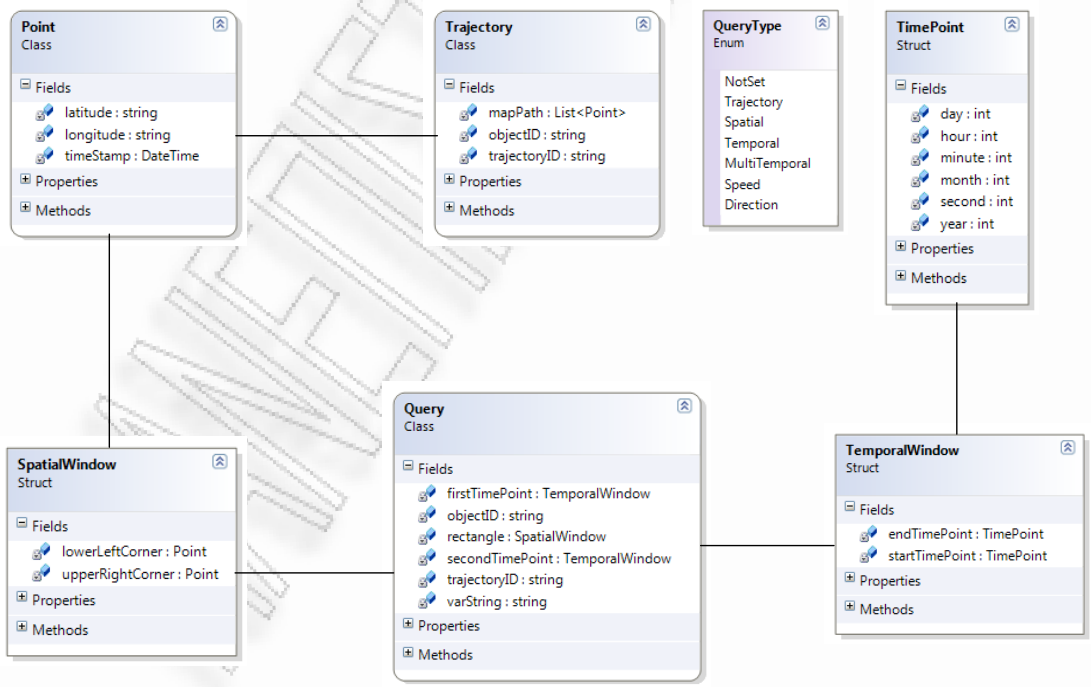


Figure 7-6: Business objects class diagram

As mentioned previously in this chapter, `Point` entity is the core component for representing our data. Each `Point` instance is comprised of a pair of *numbers*, representing the corresponding latitude and longitude of this point, along with a `timestamp`, representing the exact time point.

A list of several `Point` instances composes the `MapPath` element of `Trajectory` entity. Each `Trajectory` instance is accompanied with an `ObjectID` and a `TrajectoryID`; these elements' values are retrieved from the database and are used for providing extra information when a final object's trajectory is projected, via Google Maps/Earth.

Two `Point` instances are used to model the corners' coordinates of a spatial window. A `SpatialWindow` instance is then used as a parameter to a `Query` instance, representing the upper right and lower left corners of this rectangle.

The `TimePoint` entity is used for representing a specific timestamp. It is comprised of several elements, assisting the representation of `Year`, `Month`, `Day`, `Hour`, `Minute` and `Second` concepts. In turn, two `TimePoint` instance are used to model a `TemporalWindow`, in the nature of a time interval (start and stop). This `TemporalWindow` instance is then may be used by a `Query` instance as a parameter.

A `Query` entity is defined, representing the actual query, which is going to be sent to the database, after the necessary conversions are applied. The respective elements represent the actual query values are going to be used.

The type of the query is going to be sent is determined by a `QueryType` instance. It is mainly used by the BLL.

DAL's entities (classes) are characterized as *dummy*. That is, they do not provide any methods for changing an instance's state (fields' values). Any processing of the state is accomplished in BLL.

The Business Logic Layer (BLL), owns a series of entities, presented in Figure 7-7, accommodating business objects states' modifications. In other words, in this layer someone can get all the methods, through which an instance's state can be changed.

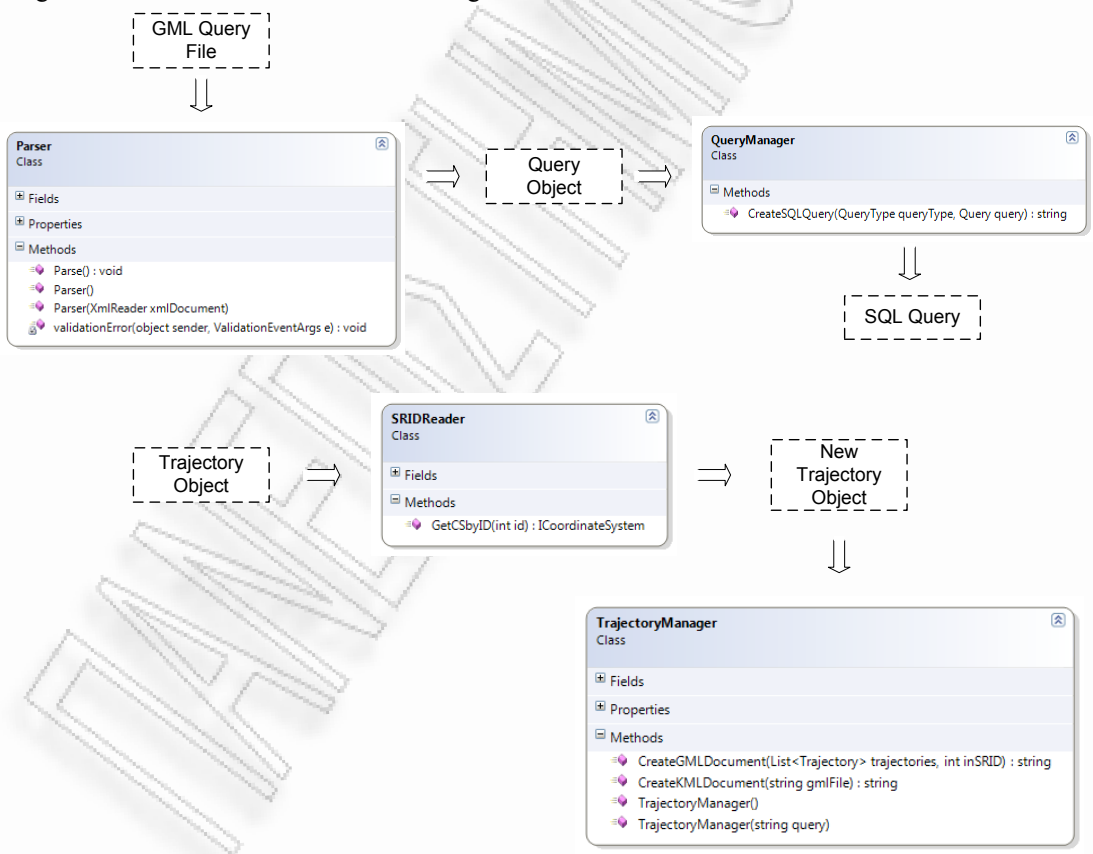


Figure 7-7: Business logic layer class diagram

The above is by no means a strict class diagram. Additional entities (surrounded by dashed lines) have been inserted, so as to provide an insight about which business objects are affected by BLL's methods.

The `Parser` entity is used to provide both validation for a GML query file, and values' extraction. This composes a `Query` instance, which is transformed into a SQL query, via the `QueryManager`

entity. When a `Trajectory` object is instantiated, its coordinate values have to be converted, consulting the `SRIDReader`'s functionality.

Finally, using `TrajectoryManager`'s methods, a `Trajectory` instance is able to be persistently stored into a GML/KML file.

The Data Access Layer (DAL), which is presented in Figure 7-8, provides all the functionality required for the wrapper to be able to communicate with the database. That is, a `DBManager` instance is able to handle the connection process to the database, a query's mission and any results' retrieval via its `Connection`, `Query` and `DataTables` elements respectively.

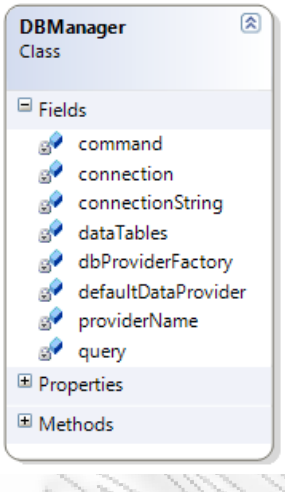


Figure 7-8: Data access layer class diagram

7.5 Data flow

Following is the corresponding data flow diagram. In this case, a user sends to VisualHERMES a GML query file and receives her results.

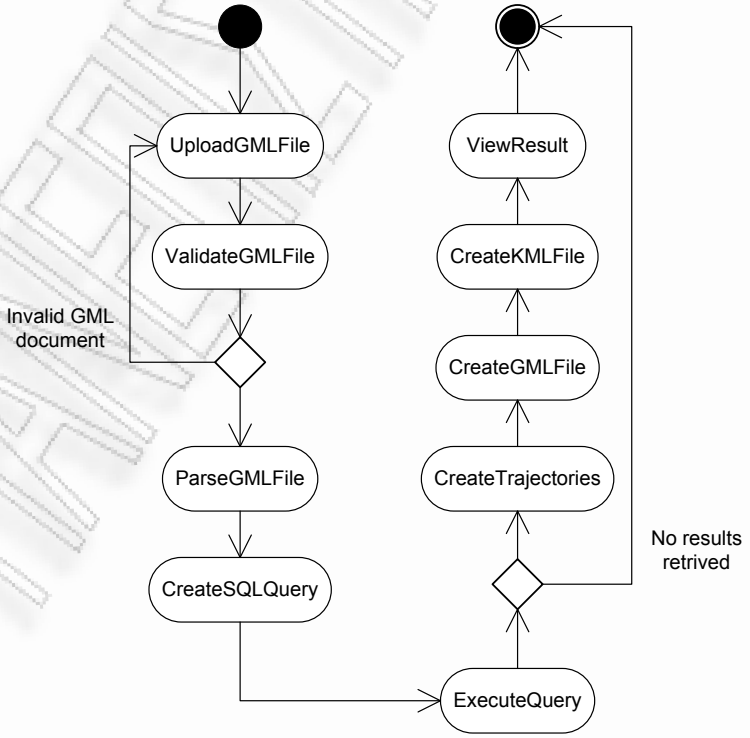


Figure 7-9: VisualHERMES data flow diagram

Figure 7-9 depicts a series of steps taking place, when a user poses a query to the system. Starting from the upper left corner of the figure, a user uploads a GML file, which represents a query. The system processes the uploaded file to validate it against the predefined schema (consult Appendix A) and if the file is not valid it returns to the first step, urging the user to correct the file. If the uploaded file is valid, the system parses it in order to extract the required values. In the next step the just extracted values are used to construct the appropriate SQL query, which is sent to HERMES system for execution. After execution is completed, results are returned to VisualHERMES, which is responsible to inform the user. If no results are returned, the overall process reaches the end. On the contrary, if VisualHERMES has results for presentation at its disposal, the process of creating the corresponding trajectories initiates; this process is the transformation of raw data into trajectory business objects. The next step is the construction of the final GML file. This file is then transformed into a KML file. This transformation is accomplished via an external formatting rules file (consult Appendix B). When the two final files' construction is completed, VisualHERMES informs end-user about the produced results. This is the final step of the overall procedure.

7.6 Development environment

VisualHERMES wrapper has been developed using Microsoft C# 2.0 programming language via Microsoft Visual Studio 2008 Integrated Development Environment, in its overall. As a consequence, Microsoft .NET Framework 2.0 must be installed, for the application to be able to run on a computer.

7.7 Access to data

Microsoft .NET Framework provides its own data access technology, named ADO.NET. ADO.NET is comprised of several core components (classes), accommodating .NET applications to connect to different data sources (usually relational and object-relational DBMSs), execute queries and process retrieved results.

The corner stone for the wrapper to be able to connect to HERMES is the use of and appropriate data provider, accomplishing all the intrinsic processes. Thus, VisualHERMES connects to the DBMS, sends queries and receives the corresponding results (Figure 7-10).

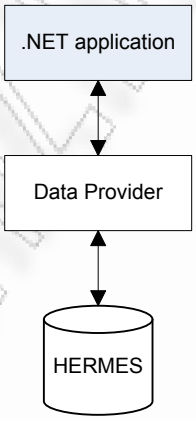


Figure 7-10: Data providers

.NET Framework provides several data providers, depending on the data source an application needs to connect to. In VisualHERMES case, the need is to connect to an Oracle DBMS, so the alternatives are the following:

1. .NET Framework Data Provider for OLE DB,
2. .NET Framework Data Provider for ODBC,
3. .NET Framework Data Provider for Oracle and
4. Oracle Data Provider for .NET Framework

From the providers above, the first two are generic approaches, covering a wide variety of DBMSs (and not only); thus the do not provide optimized functions for the connection process. .NET Framework's provider for Oracle is an optimized alternative for our wrapper's needs and finally, Oracle's

provider is the best approach we have at our disposal. VisualHERMES makes use of the forth option, Oracle's data provider.

Data handling occurs in a disconnected nature. That is, the application builds a SQL query, via either users' selections or a GML query file, sends it to the DBMS and after receiving the results, it terminates the connection. From now on, data are locally stored in out application's domain for further processing in the nature of a DataTable object, leaving the original data source disengaged (Figure 7-11).

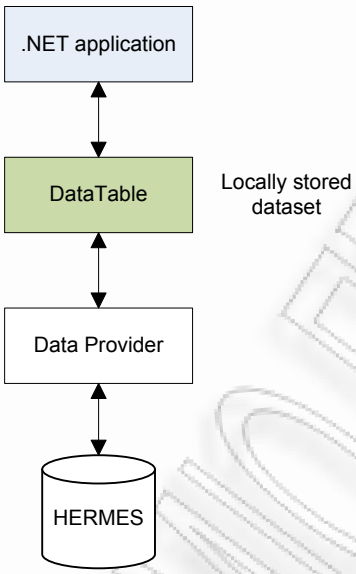


Figure 7-11: Disconnected DB access architecture

7.8 Implementation

To make VisualHERMES wrapper as user-friendly as possible, a web interface (application) has been implemented, through which a user can build the desired queries. The transformation process includes both the GML and KML files, which contain the formatted data. All a user must have is a web browsing application. The prototype has already been tested with Internet Explorer (Microsoft Corp.), Firefox (Mozilla Foundation) and Safari (Apple). Chances are that this application is compatible with almost any browser available in the market today, although it has not been tested. As a final note, the client's operating system or device does not affect user's experience (Figure 7-12).

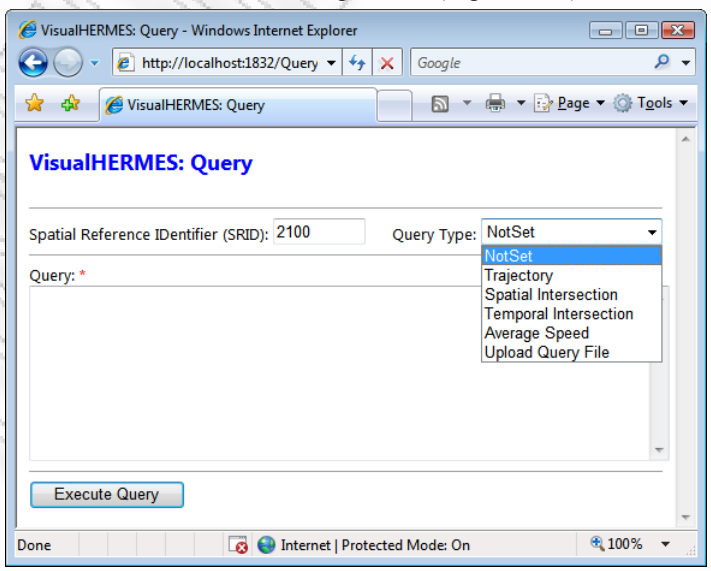


Figure 7-12: VisualHERMES query screen

7.8.1 System logon

The first time a user accesses wrapper's web interface, she is presented with the following logon screen (Figure 7-13):

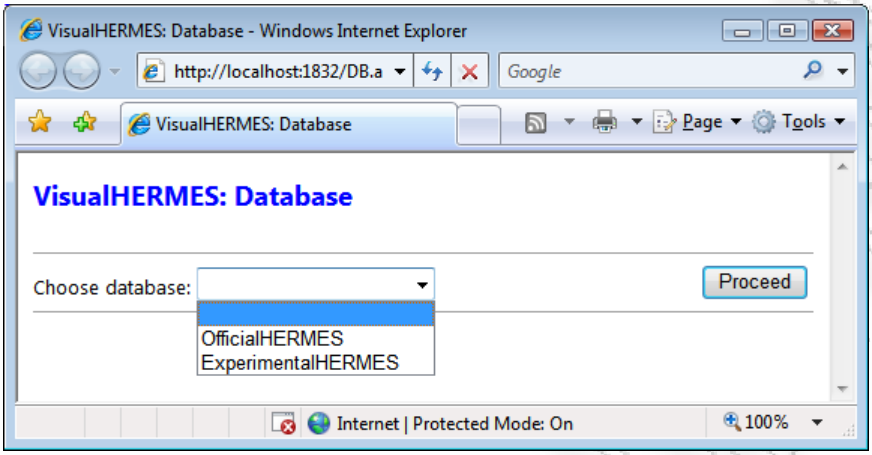


Figure 7-13: VisualHERMES logon screen

VisualHERMES is able to connect to several HERMES instances. Thus, via this option panel, a user is able to select the desired *database*.

From an administrative perspective, the wrapper retains a configuration file (`web.config`), which is placed in application's root directory. An administrator can edit this XML-based file so as to modify several aspects of the application, including the available databases. A typical file's contents can be as the following:

```
<?xml version="1.0"?>
<configuration>
 <!-- In this area we define the appropriate connection strings -->
  <connectionStrings>
   <add name="OfficialHERMES"
        connectionString="{HOST, PORT, SERVICE_NAME, USER_ID etc.}"
        providerName="Oracle.DataAccess.Client" />
   <add name="ExperimentalHERMES"
        connectionString="{HOST, PORT, SERVICE_NAME, USER_ID etc.}"
        providerName="System.Data.OracleClient" />
</connectionStrings>
...
</configuration>
```

The file above provides the administrator with a human-readable set of directives. Each <add> element represents an HERMES's instance with the corresponding connection information. Thus, an administrator is able to add as many records she wants and the wrapper's logon screen will fetch all of them on a user's first connection.

7.8.2 Query building

From the application's query screen, a user can choose the way she is going to build the query. The alternatives she has at her disposal are as follows:

1. Build the SQL query manually in the Query field,
2. Choose one of the predefined query types in the Query Type field or
3. Upload a GML query file, containing all the required information needed for query's construction.

Thus, users have the ability to build their queries using any technique there are more familiar with. For example, a user does not have to know anything about SQL or GML. She only has to provide the required values and get the results. A user who is familiar with GML syntax can construct her query file (using a text editor) without being aware of the various spatio-temporal operators HERMES provides.

Finally, a user who has been working with HERMES for some time can construct a query using straight SQL syntax. This method reveals a more flexible schema is which she is able to override the predefined query types and get results based on more sophisticated queries.

If the user selects the `NotSet` query type, she is presented with an empty text area. Therein, she is able to type any valid SQL query, which is then passed to HERMES via VisualHERMES wrapper. At this point, it has to be mentioned that there is a specific constraint; that is, the SQL query must request three specific fields from the database. These fields are the moving object's ID (`OBJECT_ID`), the moving object trajectory's ID (`TRAJ_ID`) and of course the actual trajectory field (`MPOINT`), with the remark of trajectory. Thus, a typical query that could be provided in this area is the following.

```
SELECT
      a.object_id,
      a.traj_id,
      a.mpoint.to_clob() AS trajectory
FROM
      mpoints a
WHERE
      a.object_id = {Object ID}
      AND
      a.traj_id = {Trajectory ID}
```

The use of *`to_clob()`* function, is discussed later in this chapter.

In the case, a user selects one of the predefined query types, she is provided with several fields that must be filled, so the system becomes able to get the required information for building the query. Each option leads to different fields, as described below.

The trajectory query type causes the system to return a specific trajectory, identified by an object ID and a trajectory ID. Thus, it is required from the user to specify the just mentioned values. The equivalent SQL query is as follows.

```
SELECT
      a.object_id,
      a.traj_id,
      a.mpoint.to_clob() AS trajectory
FROM
      mpoints a
WHERE
      a.object_id = {Object ID}
      AND
      a.traj_id = {Trajectory ID}
```

In the case of spatial intersection query type the returned trajectory is restricted inside a given geometry. The user is required to provide both object's and trajectory's IDs, as well as a pair of coordinates representing the upper right and the lower left corners of a geographic rectangle. Although HERMES is able to handle any polygonal region, VisualHERMES is restricted to rectangles. The required Oracle's Spatial operators are `SDO_GEOMETRY` and `F_INTERSECTION`. The equivalent SQL query is the following.

```
SELECT
    a.object_id,
    a.traj_id,
    a.mpoint.f_intersection
    (
        MDSYS.SDO_GEOMETRY
        (
            2003,
            NULL,
            NULL,
            MDSYS.SDO_ELEM_INFO_ARRAY
            (
                1,
                1003,
```

```
                3
            ),
            MDSYS.SDO_ORDINATE_ARRAY
            (
                {Upper Right Longitude},{Upper Right Latitude},
                {Lower Left Longitude},{Lower Left Latitude}
            )
        ), 0.001
    ).to_clob() AS trajectory
FROM
    mpoints a
WHERE
    a.object_id = {Object ID}
    AND
    a.traj_id = {Trajectory ID}
```

The `f_intersection` object method returns either a geometry object that is the topological intersection (AND operation) of the two associated moving types projected at a user-defined time point or a moving object whose mapping at each instant represents a geometry that is the outcome of this set operation. Invoking `f_intersection` method for the simplest moving object, as one would expect, the result of this operation is the projection of itself on the spatial domain at time instants that intersects with other moving types or static geometries and null at time instants where it is not on the boundary or the interior of `linestrings` and `polygons` or it coincides with none of the points in a collection of them. Figure 7-14 below depicts the instantiation of a moving object modeling the intersection of a `Moving_LineString` with a polygon, at three different time points t1, t2, and t3 (Marketos, et al., 2008).

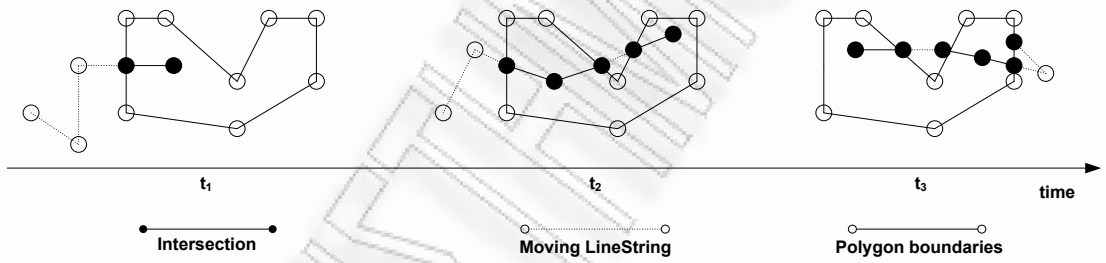


Figure 7-14: HERMES intersection operation

The temporal intersection query type, returns a trajectory restricted inside a given time period, which is specified by a `TAU_TLL`'s `D_Period_Sec` object type as a parameter to the `at_period` HERMES's operator. Beside object's and trajectory's IDs, the user has to provide values for two time points, representing the corresponding start and stop timestamps. This query type produces the following SQL code.

```
SELECT
      a.object_id,
      a.traj_id,
      a.mpoint.at_period
      (
            tau_tll.d_period _sec
            (
                  tau_tll.D_Timepoint_Sec
                  (
                        {Start Timestamp}
                  ),
                  tau_tll.D_ Timepoint_Sec
                  (
                        {Stop Timestamp}
                  )
            )
      ).to_clob() AS trajectory
```

```
FROM
      mpoints a
WHERE
      a.object_id={Object ID}
      AND
      a.traj_id={Trajectory ID}
```

The `at_period` object method is an operation that restricts the moving object to the temporal domain. In other words, by using this function the user can delimit the time period that is meaningful to ask the projection of the moving object to the spatial domain. More specifically, the time period passed as argument to the method is compared with all `D_Period_Sec` objects that characterize the unit moving objects. If the parameter period does not overlap with the compared period then the corresponding unit type is omitted. If it overlaps, then the time period that defines a unit-moving object becomes its *intersection* with the given period (Pelekis, et al., 2006).

The average speed query type is more general than the ones mentioned above, due to the fact that is does not require from the user to provide object's and trajectory's IDs. Thus, it returns all those trajectories, for which the corresponding moving objects has an average speed inside the range provided. So, a user must declare a lower and an upper speed threshold. HERMES's `f_avg_speed()` function is used from our wrapper behind the scenes and more specifically *avg_speed* measure. The equivalent query produced is the following:

```
SELECT
      a.object_id,
      a.traj_id,
      a.mpoint.to_clob() as trajectory
FROM
      mpoints a
WHERE
      a.mpoint.f_avg_speed()
      BETWEEN
            {Lower Bound Speed}
      AND
            {Upper Bound Speed}
```

The `f_avg_speed()` measure is calculated by dividing the auxiliary measure `sum_speed()` (i.e. The sum of the speeds of each trajectory portion *TP* inside a given base cell *bc*) with `count_trajectories` (Marketos, et al., 2008):

$$AVG\_SPEED(bc) = \frac{\text{SUM\_SPEED(bc)}}{\text{COUNT\_TRAJECTORIES(bc)}}$$

Where,

$$SUM\_SPEED(bc) = \sum_{\text{TPi} \subseteq \text{bc}} \frac{\text{len(TPi)}}{\text{lifespan(TPi)}}$$

If a user selects to upload her query via a GML file, she will be presented with a file path field, in which she must provide an appropriate file's path as it is represented in her local computer.

This file's contents must first meet XML's grammar specification. For an uploaded file to be able to be parsed by VisualHERMES there is a second validation level against a predefined schema. This schema defines the accepted tags a file can have. In other words, it acts as a dictionary for the query file's contents. The uploaded file can be parsed by the wrapper if and only if it has no spelling mistakes, meets the XML grammar rules and uses the accepted directives.

As an example, the following listing is a sample query file, which is equivalent to the trajectory query described previously.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is a Trajectory Query -->
<query queryType="Trajectory">
      <trajectory>
            <objectID>{Object ID}</objectID>
            <trajectoryID>{Trajectory ID}</trajectoryID>
```

```
        </trajectory>
</query>
```

In correspondence with the available SQL queries discussed above, the following are the equivalent GML queries, VisualHERMES is able to process.

A Spatial Intersection query is as follows.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is a Spatial Intersection Query -->
<query queryType="Spatial">
    <trajectory>
        <objectID>{Object ID}</objectID>
        <trajectoryID>{Trajectory ID}</trajectoryID>
    </trajectory>
    <spatial>
        <polygon>
            <exterior>
                <linearRing>
                    <posList>
                        {Upper Right Longitude} {Upper Right Latitude}
                        {Lower Left Longitude} {Lower Left Latitude}
                    </posList>
                </linearRing>
            </exterior>
        </polygon>
    </spatial>
</query>
```

A Temporal Intersection query is as follows.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is a Temporal Intersection Query -->
<query queryType="Temporal">

    <trajectory>
        <objectID>{Object ID}</objectID>
        <trajectoryID>{Trajectory ID}</trajectoryID>
    </trajectory>
    <temporal>
        <timePeriod>
            <begin>{Start Timestamp}</begin>
            <end>{Stop Timestamp}</end>
        </timePeriod>
    </temporal>

</query>
```

An Average Speed query is as follows.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is an Average Speed Query -->
<query queryType="Speed">
    <avgspeed>
        <lBoundSpeed>{Lower Bound Speed}</lBoundSpeed>
        <uBoundSpeed>{Upper Bound Speed}</uBoundSpeed>
    </avgspeed>
</query>
```

At this point, it has to be mentioned that none of the above uploaded query files is saved permanently on the server (possibly on a hard disk drive). Any file's process occurs on the fly, that is while the file's contents are on server's main memory. When the process is finished, the file is disposed.

No matter what query method is selected by a user, it is required from the latter to provide the right Spatial Reference IDentifier (SRID). This code identifies the coordinate system in which data belong.

This declaration is very important, due to that both GML and KML specifications require data being in the WGS84 (OGP, 2008) reference system (Latitude/Longitude). By specifying the original spatial reference system, VisualHERMES is able to achieve correct data conversion between these two reference systems and finally produce the appropriate files with the correct coordinates in them. All declarations and IDs required by the system are based on codification provided by the OGP Surveying and Positioning Committee, also known as European Petroleum Survey Group – EPSG (EPSG, 2008).

As mentioned above, an uploaded query file has to meet both XML's and a predefined GML-based schema's specifications. As far as the latter is concerned, below are the constraints introduced are as follows.

1. The query's type declaration is mandatory as an attribute, inside the file's first element (`NotSet`, `Trajectory`, `Spatial`, `Temporal`, `AverageSpeed`),
2. The moving object's ID must be defined inside `<objectID>` element,
3. The moving object trajectory's ID must be defined inside `<trajectoryID>` element,
4. If the query is of Spatial type, the spatial window must be provided inside the `<posList>` element in the following formation:

   `{Upper Right Longitude}<space>`
   `{Upper Right Latitude}<space>`
   `{Lower Left Longitude}<space>`
   `{Lower Left Latitude}`

   This is a core element of a complex type, named `<polygon>/<exterior>/<linearRing>/<posList>`,
5. If the query is of Temporal type, the temporal window must be defined inside the `<begin>` and `<end>` elements respectively, meeting the following notation:

   `{Year/Month/DayTHour:Minute.Second}`

   These are core elements belonging to a complex type `<timePeriod>/<begin>` and `<timePeriod>/<end>`,
6. If the query is of type Speed, the appropriate threshold must be defined inside the `<lBoundSpeed>` and `<uBoundSpeed>` elements respectively, which compose the complex type `<avgspeed>/<lBoundSpeed>` and `<avgspeed>/<uBoundSpeed>` and

As in any SQL query, a user is able to make use of any combination of the above elements. The constraints applicable in this case are as follows.

1. The `queryType` attribute has to be set as `NotSet` and
2. An additional complex type named `complex` has to bound the respective complex types.

As an example, a user can create the following GML query file.

```
<?xml version="1.0" encoding="utf-8" ?>
<query queryType="NotSet">
 <complex>
  <temporal>
   <timePeriod>
    <begin>{Start Timestamp}</begin>
    <end>{Stop Timestamp}</end>
   </timePeriod>
  </temporal>
  <avgspeed>
   <lBoundSpeed>{Lower Bound Speed}</lBoundSpeed>
   <uBoundSpeed>{Upper Bound Speed}</uBoundSpeed>
  </avgspeed>
</complex>
</query>
```

This produces the following equivalent SQL query, via the wrapper.

```
SELECT
      a.object_id,
      a.traj_id,
```

```
        a.mpoint.at_period
        (
              tau_tll.d_period_sec
              (
                    tau_tll.D_Timepoint_Sec
                    (
                          {Start Timestamp}
                    ),
                    tau_tll.D_Timepoint_Sec
                    (
                          {Stop Timestamp}
                    )
              )
        ).to_clob() AS trajectory
FROM
        mpoints a
WHERE
        a.mpoint.f_avg_speed()
        BETWEEN
              {Lower Bound Speed}
        AND
              {Upper Bound Speed}
```

The GML schema's for query files full specification listing is provided in Appendix A.

A file meeting these specifications is characterized as valid and is able to be processed by the system. VisualHERMES is able to build the appropriate SQL query using the already implemented parser, which is in charge of searching specific data formatting tags. In contrast, if the uploaded file is invalid, the user gets all the appropriate information about the errors occurred and she is able to re-upload a newer (and possibly revised) version of the query file.

7.8.3 Results construction

HERMES is able to respond to spatio-temporal queries, for moving objects. After a valid query's execution, the system is able to return the corresponding results to VisualHERMES and to be more precise, to the wrapper's DAL. Returned data are based on a predefined format, as follows.

$\{(X_1, Y_1) - (X_2, Y_2)\ \#\ TimeStamp_1 - TimeStamp_2 (X_3, Y_3) - (X_4, Y_4)\ \#\ TimeStamp_3 - TimeStamp_4 \ldots\}$

It has to be mentioned, that the data formation above is not best suited for the wrapper's needs; as a result, the latter has to re-format the results in a more convenient, tabular, structure. Thus, the first formatting process of the primitive data has to do with the flattening of them in discrete values, based on the following pattern:

X_1	Y_1	$TimeStamp_1$
X_2	Y_2	$TimeStamp_2$
X_3	Y_3	$TimeStamp_3$
X_4	Y_4	$TimeStamp_4$
…	…	…

This flattening procedure makes use of *regular expressions* in a programmatic level, giving the wrapper the ability to search and consequently match specific character patterns, separating the individual entities.

After this step, data are transferred to BLL, so more transformations to be applied on them. Thus, we end up having a set of points and timestamps at our disposal. This transformation is critical, so the system to be able to insert these data into GML- and KML-based files, via the appropriate intrinsic functions. At this point, any spatial reference system conversion is applied.

Based on the fact that our business objects, representing the corresponding data retrieved from HERMES, are ready, the final files can now be created. This process is composed of the following two discrete steps:

1. Create the GML file, where the wrapper traverses all the available trajectory objects and encapsulates them with the appropriate GML tags and
2. Create the KML file, where the wrapper, based on an external XSLT file, applies all the transformation rules, provided by the latter, to the original GML data.

Wrapper's code for the GML file's creation works independently, without any manual interference; that is, all the styling rules are hard-coded. Thus, a trajectory dataset, in the nature of business objects, follows the pattern below.

X	*Y*	*TimeStamp*
…	…	…
479493.6	4199234.9	2001/02/12 17:45:51
479219.9	4199087	2001/02/12 17:46:21
479219.9	4199087	2001/02/12 17:46:21
479013.4	4199078.6	2001/02/12 17:46:51
479013.4	4199078.6	2001/02/12 17:46:51
478861.3	4199033.7	2001/02/12 17:47:21
…	…	…

Table 7-1: Trajectories as business objects

These data are transformed into a structure similar to the following.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gml:featureCollection xmlns:gml="http://www.opengis.net/gml">
  ...
  <gml:featureMember>
    <gml:name>
        Trajectory ID: {Trajectory ID} - Object ID: {Object  ID}
    </gml:name>
    <gml:description>
        Trajectory Segment: {Segment ID}
    </gml:description>
    <gml:TimePeriod>
      <gml:begin>{Start Timestamp}</gml:begin>
      <gml:end>{Stop Timestamp}</gml:end>
    </gml:TimePeriod>
    <gml:LineString>
      <gml:posList>
        {Start Point Latitude} {Start Point Longitude}
        {Stop Point Latitude} {Stop Point Longitude}
      </gml:posList>
    </gml:LineString>
  </gml:featureMember>
  ...
</gml:featureCollection>
```

From the above listing becomes clear that all database's results are bounded by a `<gml:featureCollection>` element and each record is in turn bounded by a `<gml:featureMember>` element. Several other elements can be found in such a file, representing different concepts. These are the following:

- `<gml:name>` – Represents the objects' name, including objects' and trajectories' IDs,
- `<gml:description>` – Represents a text description of objects. In this case each trajectory's segment ID is provided,

- `<gml:TimePeriod>` – This element corresponds to the pair of timestamps required to represent a start (<gml:begin>) and a stop (<gml:end>) time point of an object's movement and
- `<gml:LineString>` – A *LineString* element is composed of a `<posList>` element, which represents the respective coordinates of an intelligible trajectory's start and stop geographic points.

For the final KML file to be created, another external this time, XML-based file, with the prominent XSLT name, providing formatting rules, consults the wrapper. Such files are based on corresponding patterns (templates). In other words, in these files someone finds declarations specifying both the original and transformed data formats. For the implemented XSLT file's full listing, consult Appendix B.

Based on the implemented file's rules, a new KML-based file is created, the contents of which can be shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <Folder>
      <name>VisualHermes</name>
      <Style id="blueLine">
        <LineStyle>
          <color>ffff0000</color>
          <width>3</width>
        </LineStyle>
      </Style>
        ...
        <Placemark>
          <name>
              Trajectory ID: {Trajectory ID} - Object ID: {Object ID}
          </name>
          <description>Trajectory Segment: {Segment ID}</description>
          <TimeSpan>
            <begin>{Start Timestamp}</begin>
            <end>{Stop Timestamp}</end>
          </TimeSpan>
          <styleUrl>#blueLine</styleUrl>
          <LineString>
            <coordinates>
                {Start Point Latitude},{Start Point Longitude},
                {Start Point Altitude}
                {Stop Point Latitude},{Stop Point Longitude},
                {Stop Point Altitude}
            </coordinates>
          </LineString>
        </Placemark>
        ...
      </Folder>
  </Document>
</kml>
```

As the above file implies, we have an XML-based file, which is composed of a root element <kml>. Except the root element, several different elements are presented, each of which represents the following:

- <name> - The file's name as this is presented using a visualization engine (Google Maps/Earth),
- <Style> - The styling rules of the presented object. On a map, a trajectory object can be presented using a line object (<LineStyle>). Based on the styling rules above, the line will have a blue color (<color>) and a width of *3* pixels (<width>),
- <Placemark> - Whilst a trajectory object is represented by a <gml:featureMember> element according to GML specification, KML makes use of the corresponding <Placemark> element,
- <name> - Represents the objects' name, including objects' and trajectories' IDs,
- <description> - Represents a text description of objects. In this case each trajectory's segment ID is provided,
- <TimeSpan> - A TimeSpan element is the GML's <gml:timePeriod> equivalent; the same goes for the respective <begin> and <end> elements,
- <StyleUrl> - In a KML file, many styling rules can be defined. Thus, each of the represented objects can be styled using a different rule. In this element is defined the rule's ID.
- <LineString> - As far as <LineString> and <coordinates> elements are concerned, they represent the <gml:LineString> and <gml:posList> elements of a GML file.

Although it is not clear from the listing above, we have to mention that a modification occurs, as far as timestamps are concerned. To be more precise, the timestamp format changes when it comes to meet KML specification. Each timestamp in a KML file complies with the following pattern.

```
{Year}-{Month}-{Date}T{Hour}:{Minute}:{Second}Z
```

The files produced by the process described above, are the final result files, VisualHERMES sends to end users. One important issue is that of duplicates in file names. To outflank this issue, a custom naming algorithm has been implemented. The basic files' name is *Output*. Using the algorithm above, two concatenations are applied.

1. A pseudo-random string, representing a statically unique 128-bits integer (Globally Unique Identifier - GUID), is supplied as a suffix to the original name and
2. A timestamp with seconds' precision is supplied as a suffix too, to the just *renamed* file name.

Thus, a typical GML file's name is based on the following naming pattern.

```
Output-{GUID}-{Date}-{Time}.gml
```

According to the above, the KML file has a name as below.

```
Output-{GUID}-{Date}-{Time}.kml
```

Finally, the produced by VisualHERMES files, are presented to the end user, as two hyperlinks. The first of them represents a GML file the user can download to her local computer. As far as the second (KML) file is concerned, the user is able to get the visualized results, via the same web browser's window, using the provided Maps services by Google. Through Google Maps API, a second web page has been implemented. Its purpose is to project KML results, VisualHERMES created, on real earth's maps.

At this point, the end user is provided with several tools. Using them she is able to *move* selected map regions, *zoom in* or *zoom out* etc. Furthermore, each trajectory's segment carries additional information like the segment (movement) ID, its overall ID as provided by HERMES and the moving object's ID. For more information about this data, consult the *Case Study* chapter of this thesis.

7.9 Compression features

One of the largest challenges when dealing with XML-based data is the quick increase of files' sizes. This is due to XML's nature itself. Tagging each piece of information with several *keywords* can lead to really large-sized data files, especially if the original dataset is big by its own; this is an often case when dealing

with trajectory data. As a proof of concept for the claim above, we present the following table so as to understand the increase ratios.

Rows	*Raw (Bytes)*	*GML (Bytes)*	*KML (Bytes)*
7.550	669.057	3.464.816	3.021.036
5.771	513.087	2.651.869	2.312.224
901	79.816	413.139	360.293
404	35.044	184.806	161.283
333	29.023	152.180	132.846
44	3.779	19.856	17.514

Table 7-2: Dataset sizes comparison

The first technique a developer has at her disposal is *compression*. Based on this approach, sizes tend to be reduced. Especially, in the case of text files (XML is ultimately a text-based grammar), the reduction ratios that can be achieved are really promising. To get an insight about how compression helps in our application domain, consult the following chart.

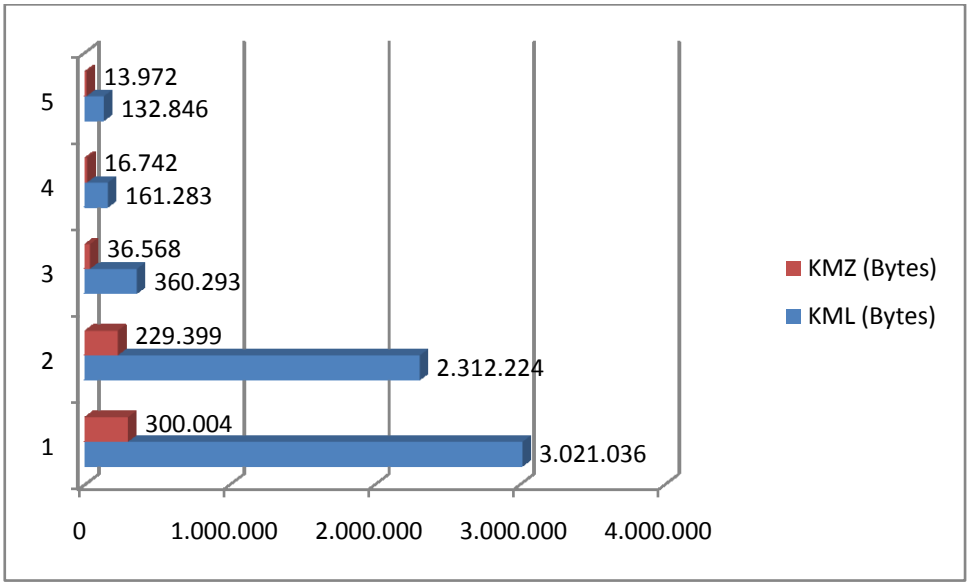


Figure 7-15: Dataset compression ratios

Introduction of compression techniques has drawbacks as well. First and most critical is that the system becomes more complex. Furthermore, produced files tend to be more difficult while handling them. For example, if a GML file gets compressed, it will require a corresponding decompression mechanism, to become *readable* again. This is due to the fact that GML is a generic specification and many different applications use such files for processing the underlying data, without being aware of algorithm used to compress the original data file. As a result, handling compressed GML data files is not always trivial, making them unsuitable for generic and interoperable solutions.

The same is not true as far as KML data files are concerned. This happens because KML is used only for presenting data and this projection is accomplished by specific visualization engines. Each of them is aware of the compression mechanism a KML file can use (the *Deflate algorithm* (Deutsch, 1996)), so it is possible for a visualization engine to get a compressed KML file (known as KMZ files), decompress it on the fly and present the appropriate data. After all, almost all of the known projection engines are aware of KMZ files nowadays.

As far as VisualHERMES is concerned, and from the .NET Framework perspective, it is quite difficult to implement such functionality. .NET Framework has a built-in mechanism supporting the deflate compression algorithm, but when it comes to be applied on physical files, rather than in-memory streams, the appropriate headers are not added into the final result file. This leads to a file, which does not make itself understood to the outer world (namely, applications, that handle KML files).

Although, this rather odd behavior has been fixed in later versions of .NET Framework, in our case (VisualHERMES runs on .NET Framework 2.0), we had to make use of a third-party compression library,

named *DotNet Zip Library 1.3* (DotNet Zip, 2007). DotNet Zip Library is an open-source, fully managed code, written in C# DLL library, that provides support for reading and writing Zip archive files and streams, using either Deflate or GZip algorithms. All tests have been made, gave promising results, but due to the increase of overall system's complexity, this feature is disabled in the current release (although it has been fully implemented and tested).

7.10 Problems in implementation

During the development process of VisualHERMES, we faced two main problems. The first was about the spatial reference systems conversion. The other one had to do with HERMES *Abstract Data Types – ADT* (Oracle Corp., 2003) and the way .NET Framework data providers handle them.

As far as the conversion algorithm is concerned, the problem has its roots in that HERMES uses Oracle's built-in spatial reference system, which is the Cartesian system. This SRS-agnostic handling of coordinates was a burden for VisualHERMES, due to GML and KML specifications, which demands from the coordinates to be in the WGS84 (Latitude/Longitude) spatial reference system (OGP, 2008). So what the wrapper is supposed to do if a dataset is already in the previously mentioned SRS and another one is in, for example Greek Geodetic Reference System - GGRS87 (OGP, 2008)? How is it going to handle such different data? The approach chosen was the use of a third-party library, named Proj.Net (Guidi, 2007). Proj.Net is an open-source, DLL library, written in C# that performs point-to-point coordinate conversions between geodetic coordinate systems. The spatial reference model used adheres to the Simple Features specification (OGC, 2008). Using its functionality a .NET Framework application can accomplish all the required conversions between different geodetic reference systems via EPSG's codification.

HERMES uses a set of ADTs to present various custom moving objects' information. None of the available data providers can handle such types, because it is not aware of the intrinsic functionality a type can have.

For VisualHERMES to be able to handle the data retrieved using HERMES, we has to convert the primitive data into a well known data type, the data provider if aware of, such as `varchar2`. As a matter of fact, HERMES already supports such kind of functionality, via the corresponding `to_string()` function, but using a `varchar2` as the return data type introduces a drawback. `Varchar2` data types can handle a stream of up to 4000 characters. Any stream containing more characters causes HERMES to return an error.

To overcome this issue, a new function has been implemented. This new function uses a `CLOB` (a variation of the generic `BLOB` type, targeting in character streams) variable as the return type, capable for handling enormous character streams (approx. 2×10^9 characters). By integrating `to_clob()` function with HERMES's `moving_point` core ADT, we are now able to handle large amounts of data via our wrapper. Furthermore, due to the fact that this conversion occurs after the dataset is retrieved from the actual database, there are no drawbacks with SQL queries, some of them do not operate correctly on `CLOB` data types. The full `to_clob()`'s function source code (in PL/SQL) is provided in Appendix C.

7.11 Conclusion

Data can be transformed in any desired format using various techniques and/or technologies. Using a plain text transformation algorithm we managed to convert raw data into GML structures. GML data can be transformed into KML files based on XSLT styling rules. Any of these well-known specifications, or even custom ones, can be validated via XSD files. All the above operations can be accomplished using an integrated environment, acting as a wrapper for a DBMS.

8 Case Study

8.1 Introduction

As a proof of evidence for the prototype system described in the previous chapter, herein we going to discuss the overall system's operation using real life trajectory data. We are going to demonstrate all the available ways of building a query, sending it to HERMES and retrieving the corresponding results both in GML and KML format. Visualized results are also presented. Each section follows a specific pattern: Objective, building a query via VisualHERMES's web interface, SQL equivalent query, GML equivalent query, getting results in GML and finally getting and visualizing KML results.

8.2 Dataset characteristics

The original dataset comes from (R-tree Portal, 2005) and it represents a fleet of school buses moving in Athens metropolitan area between years 2000 and 2001. As a consequence, all the available coordinates' information is in the Greek Geodetic Reference System 87. This reference system has an ID of `2100` according to EPSG's codification, in contrast to WGS84's reference system, which has a corresponding ID of `4326`. So VisualHERMES has to get a value of `2100` for the Spatial Reference IDentification – SRID (OGP, 2008). The destination reference system has to always be `4326`, so this declaration is not required from the end user.

As a final note, the *NotSet* query type, option is not discussed independently. This is because all the equivalent SQL queries are fully described as part of the other query types.

8.3 Trajectory query type

The objective is to fetch the trajectory with ID `6`, which belongs to a moving object with ID `0420`. The web interface through which the query is constructed is illustrated in Figure 8-1.

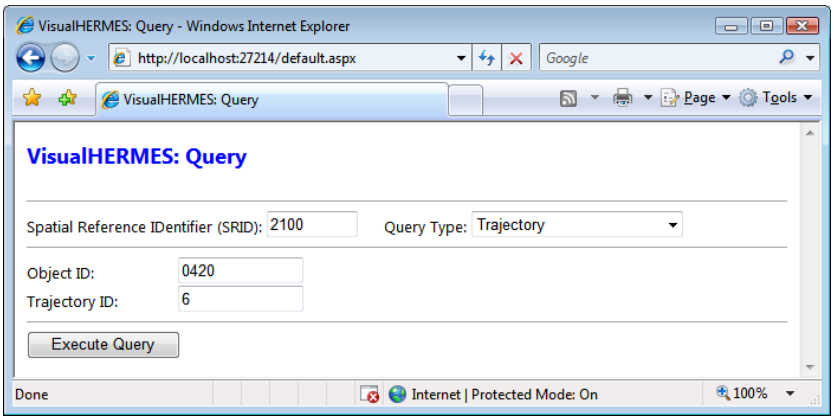


Figure 8-1: Trajectory web interface

The SQL and GML equivalents are illustrated in Figure 8-4 and Figure 8-7 respectively.

```
SELECT
      a.object_id,
      a.traj_id,
      a.mpoint.to_clob() as trajectory
FROM
      mpoints a
WHERE
      a.object_id = 0420
      AND
      a.traj_id = 6
```

Figure 8-2: Trajectory SQL query

```
<?xml version="1.0" encoding="utf-8" ?>
<query queryType="Trajectory">
      <trajectory>
            <objectID>0420</objectID>
            <trajectoryID>6</trajectoryID>
      </trajectory>
</query>
```

Figure 8-3: Trajectory GML query

Figure 8-4 illustrates the results page, which links to two files results in GML and KML formats respectively.

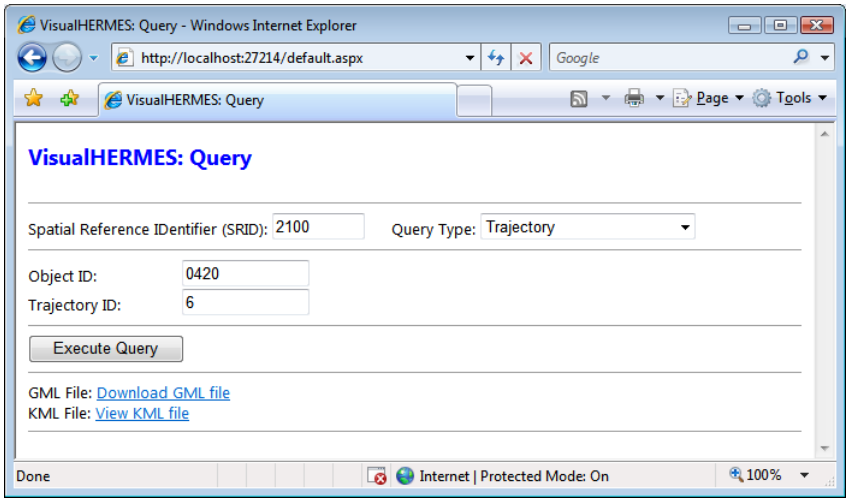


Figure 8-4: Trajectory results page

The results for the produced GML and KML files are presented in Figure 8-5 and Figure 8-6 respectively.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gml:featureCollection xmlns:gml="http://www.opengis.net/gml">
  <gml:featureMember>
    <gml:name>Trajectory ID: 6 - Object ID: 0420</gml:name>
    <gml:description>Trajectory Segment: 1</gml:description>
    <gml:TimePeriod>
      <gml:begin>2001/02/12T17:45.21</gml:begin>
      <gml:end>2001/02/12T17:45.51</gml:end>
    </gml:TimePeriod>
    <gml:LineString>
      <gml:posList>
          37.94513 23.77097 37.94306 23.76831
      </gml:posList>
    </gml:LineString>
  </gml:featureMember>
  <gml:featureMember>
  ...
  </gml:featureMember>
</gml:featureCollection>
```

Figure 8-5: Trajectory GML results

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
    <Document>
        <Folder>
            <name>VisualHermes</name>
            <Style id="blueLine">
                <LineStyle>
                    <color>ffff0000</color>
                    <width>3</width>
                </LineStyle>
            </Style>
            <Placemark>
                <name>Trajectory ID: 5 - Object ID: 0420</name>
                <description>Trajectory Segment: 1</description>
                <TimeSpan>
                    <begin>2001-02-09T16:48:53Z</begin>
                    <end>2001-02-09T16:49:23Z</end>
                </TimeSpan>
                <styleUrl>#blueLine</styleUrl>
                <LineString>
                    <altitudeMode>relative</altitudeMode>
                    <coordinates>
                        23.85472,38.00457,0
                        23.85752,38.00272,0
                    </coordinates>
                </LineString>
            </Placemark>
            <Placemark>
                ...
            </Placemark>
        </Folder>
    </Document>
</kml>
```

Figure 8-6: Trajectory KML results

KML results file can be visualized through Google Maps, as Figure 8-7 depicts.

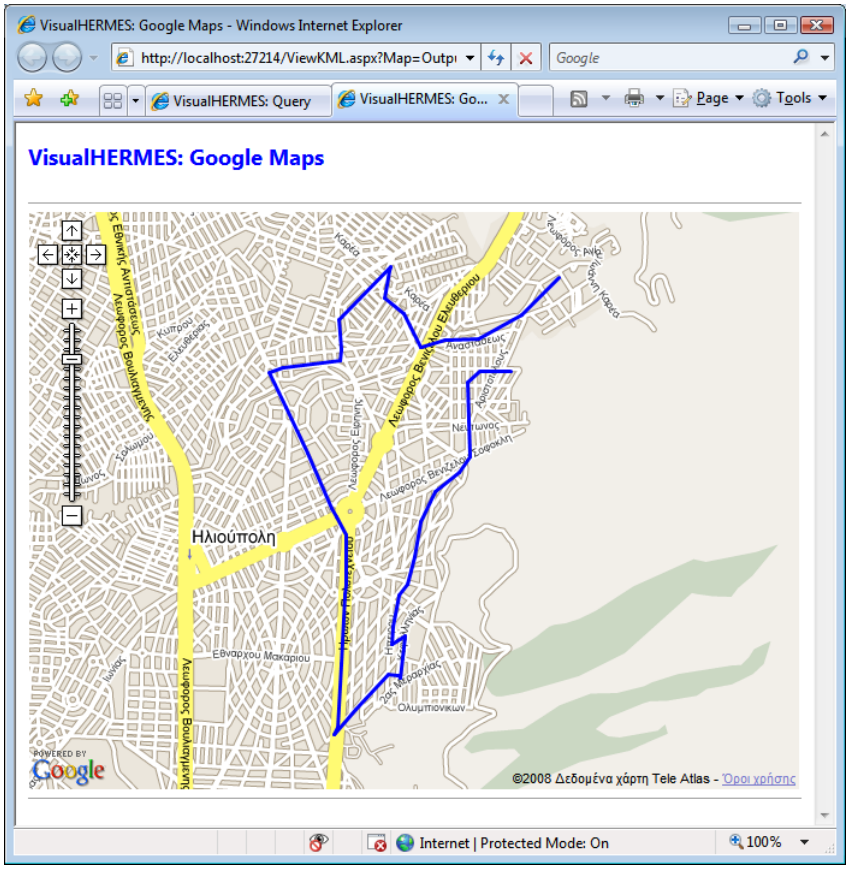


Figure 8-7: Trajectory visualized KML data

8.4 Spatial Intersection query type

In this scenario we want to fetch object's `0420` trajectory with ID `6` and restrict it in a geographic rectangle with the following characteristics.

- Upper right corner's coordinates.
  - Longitude (X): `479610.8`
  - Latitude (Y): `4199349.5`
- Lower left corner's coordinates.
  - Longitude (X): `479356.75`
  - Latitude (Y): `4199160.95`

Figure 8-8 illustrates the web interface to build our query.

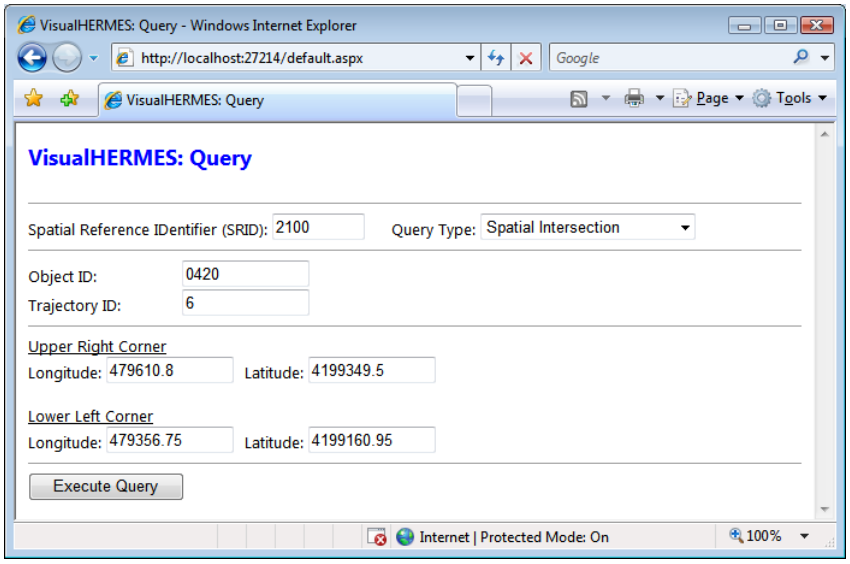


Figure 8-8: Spatial Intersection web interface

The query above is translated into a SQL query sent to HERMES. The actual query is depicted in Figure 8-9.

```
SELECT
    a.object_id,
    a.traj_id,
    a.mpoint.f_intersection
    (
        MDSYS.SDO_GEOMETRY
        (
            2003,
            NULL,
            NULL,
            MDSYS.SDO_ELEM_INFO_ARRAY
            (
                1,
                1003,
                3
            ),
            MDSYS.SDO_ORDINATE_ARRAY
            (
                479610.8,4199349.5, 479356.75,4199160.95
            )
        ), 0.001
    ).to_clob() AS trajectory
FROM
    mpoints a
WHERE
    a.object_id = 0420
      AND
       a.traj_id = 6
```

Figure 8-9: Spatial Intersection SQL query

In a similar manner, the GML equivalent, a user can upload is presented in Figure 8-10.

```
<?xml version="1.0" encoding="utf-8" ?>
<query queryType="Spatial">
      <trajectory>
        <objectID>0420</objectID>
        <trajectoryID>6</trajectoryID>
      </trajectory>
      <spatial>
          <polygon>
              <exterior>
                  <linearRing>
                      <posList>
                          479610.8 4199349.5 479356.75 4199160.95
                      </posList>
                  </linearRing>
              </exterior>
          </polygon>
      </spatial>
</query>
```

Figure 8-10: Spatial Intersection GML query

Figure 8-11 depicts the corresponding results page. Through this page a user is presented with two hyperlinks pointing to the two result files (GML and KML).

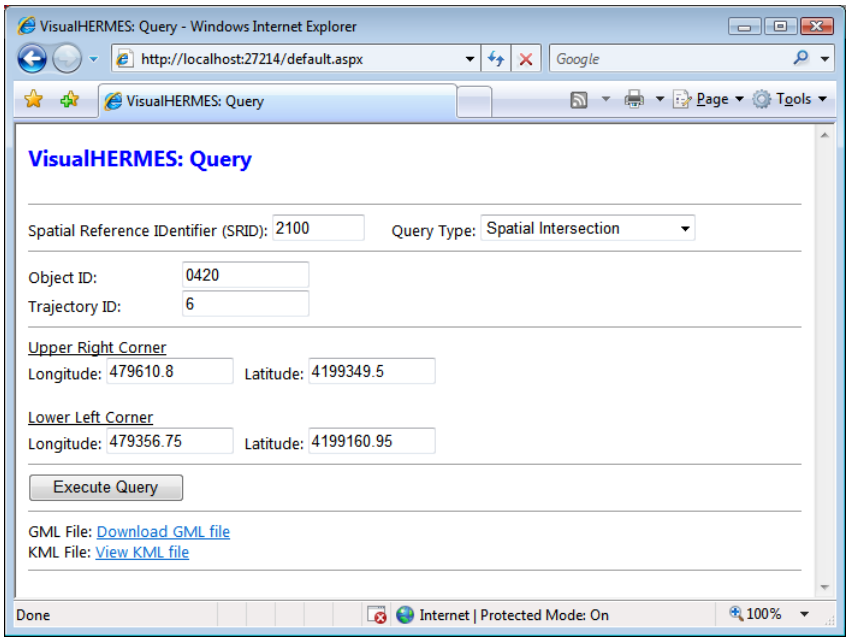


Figure 8-11: Spatial Intersection results page

Each of the two files produced for the Spatial Intersection query type is illustrated in Figure 8-12 for the GML file and in Figure 8-13 for the KML one.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gml:featureCollection xmlns:gml="http://www.opengis.net/gml">
    <gml:featureMember>
        <gml:name>Trajectory ID: 6 - Object ID: 0420</gml:name>
        <gml:description>Trajectory Segment: 1</gml:description>
        <gml:TimePeriod>
            <gml:begin>2001/02/12T17:45.36</gml:begin>
            <gml:end>2001/02/12T17:45.51</gml:end>
        </gml:TimePeriod>
        <gml:LineString>
            <gml:posList>
                37.94409 23.76964 37.94306 23.76831
            </gml:posList>
        </gml:LineString>
    </gml:featureMember>
    <gml:featureMember>
        ...
    </gml:featureMember>
</gml:featureCollection>
```

Figure 8-12: Spatial Intersection GML results

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
    <Document>
        <Folder>
            <name>VisualHermes</name>
            <Style id="blueLine">
                <LineStyle>
                    <color>ffff0000</color>
                    <width>3</width>
                </LineStyle>
            </Style>
            <Placemark>
                <name>Trajectory ID: 6 - Object ID: 0420</name>
                <description>Trajectory Segment: 1</description>
                <TimeSpan>
                    <begin>2001-02-12T17:45:36Z</begin>
                    <end>2001-02-12T17:45:51Z</end>
                </TimeSpan>
                <styleUrl>#blueLine</styleUrl>
                <LineString>
                    <altitudeMode>relative</altitudeMode>
                    <coordinates>
                        23.76964,37.94409,0
                        23.76831,37.94306,0
                    </coordinates>
                </LineString>
            </Placemark>
            <Placemark>
                ...
            </Placemark>
        </Folder>
    </Document>
</kml>
```

Figure 8-13: Spatial Intersection KML results

Google Maps provides the ability to visualize our KML results file. The visualized results are projected in Figure 8-14.

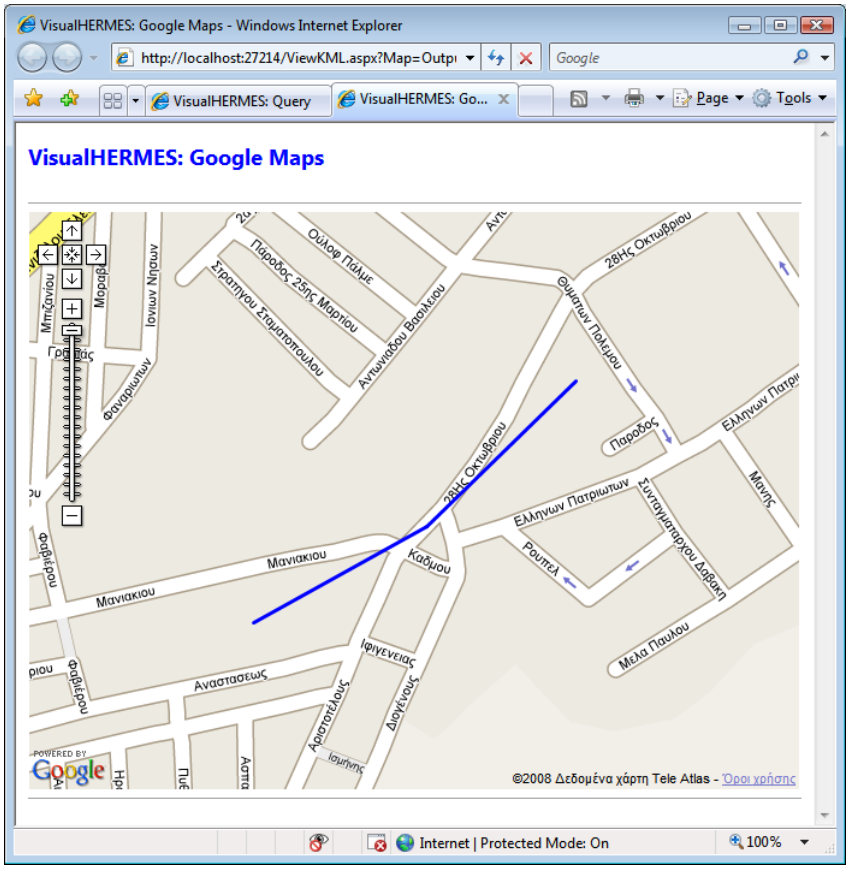


Figure 8-14: Spatial Intersection visualized KML data

8.5 Temporal Intersection query type

In this case this objective is to fetch the object's `0420` trajectory `5` and restrict it into a time interval with the following characteristics.

- Begin date and time: `February 9, 2001 at 4:48:53pm` and
- End date and time: `February 9, 2001 at 4:53:53pm`

The query builder provided by the web interface is illustrated in Figure 8-15, while its SQL and GML equivalents are presented in Figure 8-16 and Figure 8-17 respectively.

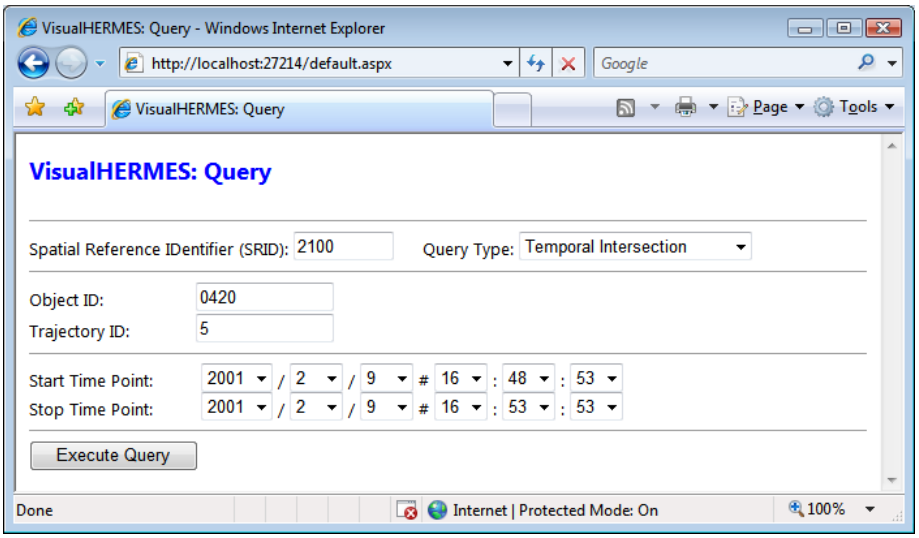


Figure 8-15: Temporal Intersection web interface

```
SELECT
    a.object_id,
    a.traj_id,
    a.mpoint.at_period
    (
        tau_tll.d_period_sec
        (
            tau_tll.D_Timepoint_Sec
            (
                2001,2,9,16,48,53
            ),
            tau_tll.D_Timepoint_Sec
            (
                2001,2,9,16,53,53
            )
        )
    ).to_clob() AS trajectory
FROM
    mpoints a
WHERE
    a.object_id = 0420
      AND
      a.traj_id = 5
```

Figure 8-16: Temporal Intersection SQL query

```
<?xml version="1.0" encoding="utf-8" ?>
<query queryType="Temporal">
      <trajectory>
            <objectID>0420</objectID>
            <trajectoryID>5</trajectoryID>
      </trajectory>
      <temporal>
            <timePeriod>
                  <begin>2001/02/09T16:48.53</begin>
                  <end>2001/02/09T16:53.53</end>
            </timePeriod>
      </temporal>
</query>
```

Figure 8-17: Temporal Intersection GML query

The corresponding links pointing to the produced files are presented though the results page in Figure 8-18.

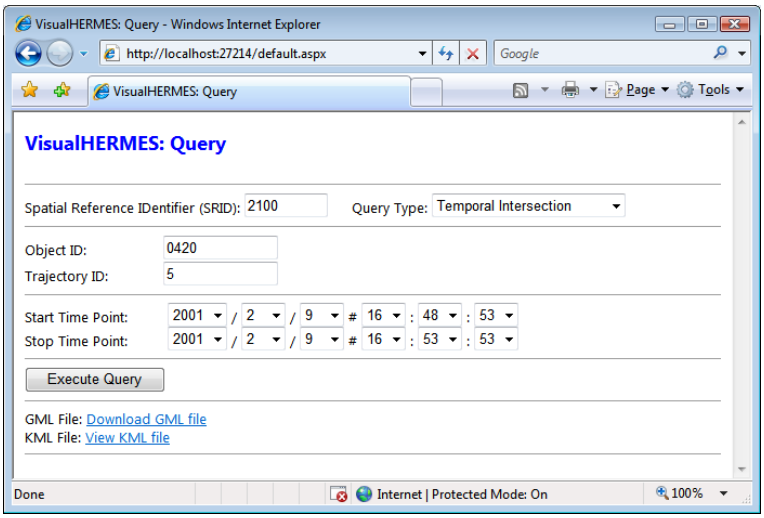


Figure 8-18: Temporal Intersection results page

The actual files constructed are presented in Figure 8-19 (GML results file) and in Figure 8-20 (KML results file).

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gml:featureCollection xmlns:gml="http://www.opengis.net/gml">
    <gml:featureMember>
        <gml:name>Trajectory ID: 5 - Object ID: 0420</gml:name>
        <gml:description>Trajectory Segment: 1</gml:description>
        <gml:TimePeriod>
            <gml:begin>2001/02/09T16:48.53</gml:begin>
            <gml:end>2001/02/09T16:49.23</gml:end>
        </gml:TimePeriod>
        <gml:LineString>
            <gml:posList>
                38.00457 23.85472 38.00272 23.85752
            </gml:posList>
        </gml:LineString>
    </gml:featureMember>
    <gml:featureMember>
        ...
    </gml:featureMember>
</gml:featureCollection>
```

Figure 8-19: Temporal Intersection GML results

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
    <Document>
        <Folder>
            <name>VisualHermes</name>
            <Style id="blueLine">
                <LineStyle>
                    <color>ffff0000</color>
                    <width>3</width>
                </LineStyle>
            </Style>
            <Placemark>
                <name>Trajectory ID: 5 - Object ID: 0420</name>
                <description>Trajectory Segment: 1</description>
                <TimeSpan>
                    <begin>2001-02-09T16:48:53Z</begin>
                    <end>2001-02-09T16:49:23Z</end>
                </TimeSpan>
                <styleUrl>#blueLine</styleUrl>
                <LineString>
                    <altitudeMode>relative</altitudeMode>
                    <coordinates>
                        23.85472,38.00457,0
                        23.85752,38.00272,0
                    </coordinates>
                </LineString>
            </Placemark>
            <Placemark>
                ...
            </Placemark>
        </Folder>
    </Document>
</kml>
```

Figure 8-20: Temporal Intersection KML results

The visualized version of the produced KML file using Google Maps is presented in Figure 8-21.

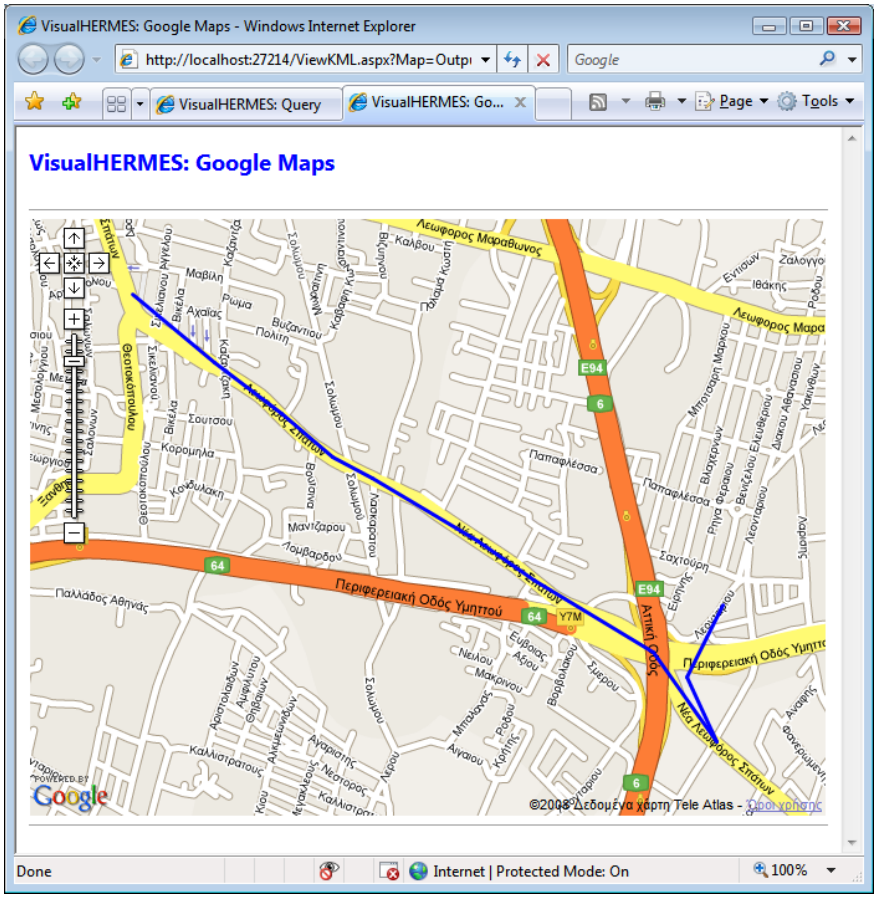


Figure 8-21: Temporal Intersection visualized KML data

8.6 Average Speed query type

In this scenario the goal is to fetch all available moving objects' trajectories having an average speed between `30` and `60 Km/h`.

The web interface accommodating our query building is depicted in Figure 8-22.

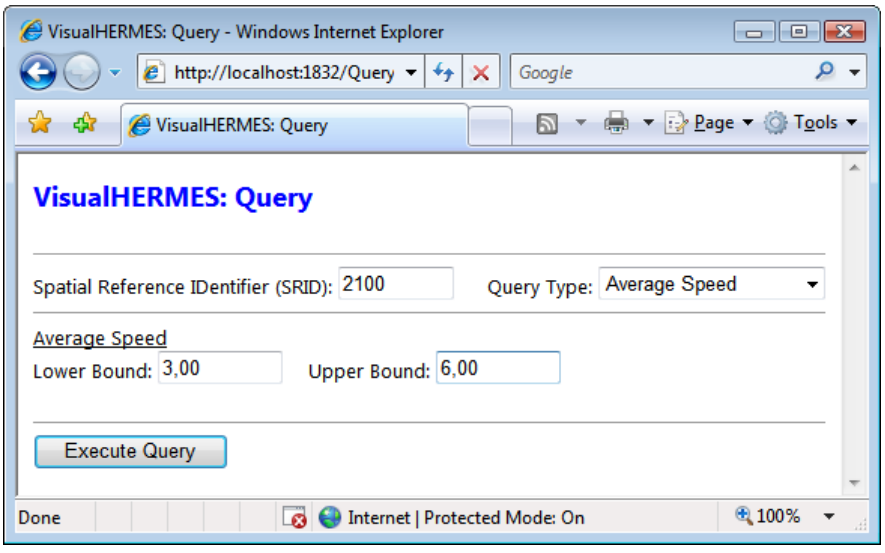


Figure 8-22: Average Speed web interface

The SQL and GML equivalents are presented in Figure 8-23 and Figure 8-24 respectively.

```
SELECT
    a.object_id,
    a.traj_id,
    a.mpoint.to_clob() AS trajectory
FROM
    mpoints a
WHERE
    CAST(a.mpoint.f_avg_speed() AS number(*,2))
    BETWEEN
        3.00
    AND
        6.00
```

Figure 8-23: Average Speed SQL query

```
<?xml version="1.0" encoding="utf-8" ?>
<query queryType="Averagespeed">
      <avgpeed>
            <lBoundSpeed>3.00</lBoundSpeed>
            <uBoundSpeed>6.00</uBoundSpeed>
      </avgspeed>
</query>
```

Figure 8-24: Average Speed GML query

The corresponding links are provided to the user via the results page, which is presented in Figure 8-25.

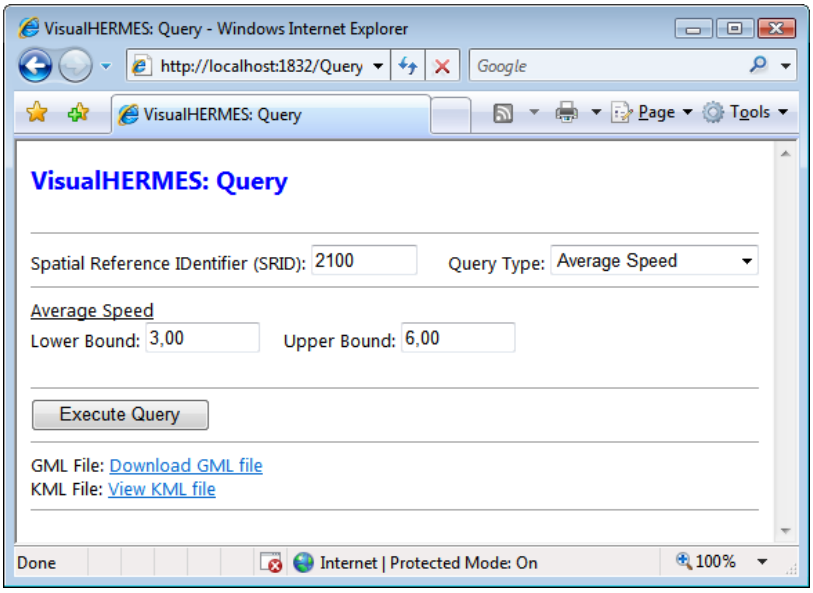


Figure 8-25: Average Speed results page

An excerpt of the results files produced by VisualHERMES is illustrated in Figure 8-26 for the GML file and in Figure 8-27 for the corresponding KML file.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gml:featureCollection xmlns:gml="http://www.opengis.net/gml">
  <gml:featureMember>
    <gml:name>Trajectory ID: 6 - Object ID: 0420</gml:name>
    <gml:description>Trajectory Segment: 1</gml:description>
    <gml:TimePeriod>
      <gml:begin>2001/02/12T17:45.21</gml:begin>
      <gml:end>2001/02/12T17:45.51</gml:end>
    </gml:TimePeriod>
    <gml:LineString>
      <gml:posList>
          37.94513 23.77097 37.94306 23.76831
      </gml:posList>
    </gml:LineString>
  </gml:featureMember>
  <gml:featureMember>
  ...
  </gml:featureMember>
</gml:featureCollection>
```

Figure 8-26: Average Speed GML results

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
    <Document>
        <Folder>
            <name>VisualHermes</name>
            <Style id="blueLine">
                <LineStyle>
                    <color>ffff0000</color>
                    <width>3</width>
                </LineStyle>
            </Style>
            <Placemark>
                <name>Trajectory ID: 5 - Object ID: 0420</name>
                <description>Trajectory Segment: 1</description>
                <TimeSpan>
                    <begin>2001-02-09T16:48:53Z</begin>
                    <end>2001-02-09T16:49:23Z</end>
                </TimeSpan>
                <styleUrl>#blueLine</styleUrl>
                <LineString>
                    <altitudeMode>relative</altitudeMode>
                    <coordinates>
                        23.85472,38.00457,0
                        23.85752,38.00272,0
                    </coordinates>
                </LineString>
            </Placemark>
            <Placemark>
                ...
            </Placemark>
        </Folder>
    </Document>
</kml>
```

Figure 8-27: Average Speed KML results

The KML results file can be visualized though Google Maps as illustrated in Figure 8-28.

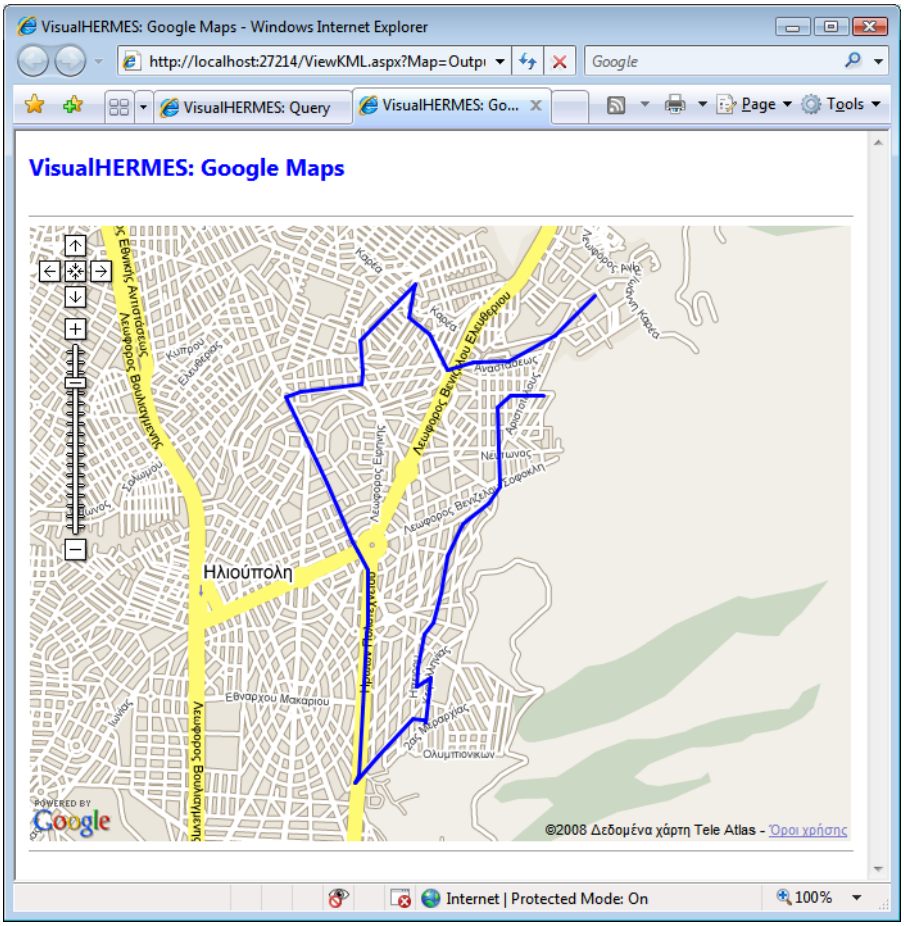


Figure 8-28: Average Speed visualized KML data

8.7 NotSet query type

As mentioned above, this type of query did not described independently. The only notice we have to make is that any of the previously mentioned SQL queries, as any valid SQL query as well, can be typed into the corresponding *Query* web interface's field and VisualHERMES, is able to process and execute it, retrieving the appropriate results as shown in Figure 8-29.

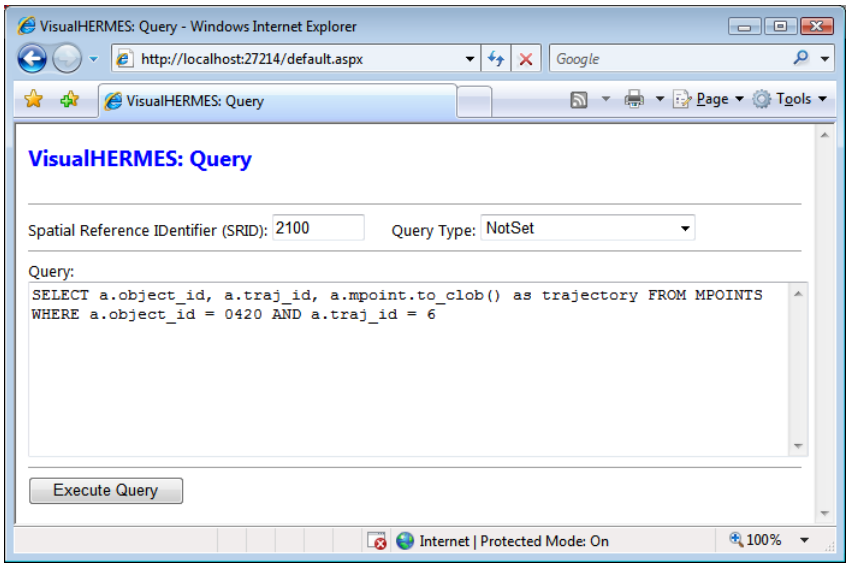


Figure 8-29: Manual build of SQL query

8.8 Upload GML query file

According to our schema specification, a user is able to construct more complex queries than the ones discussed previously. Thus, she is able to build queries specifying temporal, speed and spatial operators, all in one file.

In this scenario our intention is to fetch all the available trajectories between February 12, 2001 at 5:45:12pm and February 12, 2001 at 5:45:51pm, with average speed between 5 and 6 Km/h, building and uploading a more *complex* GML query file, as presented in Figure 8-30.

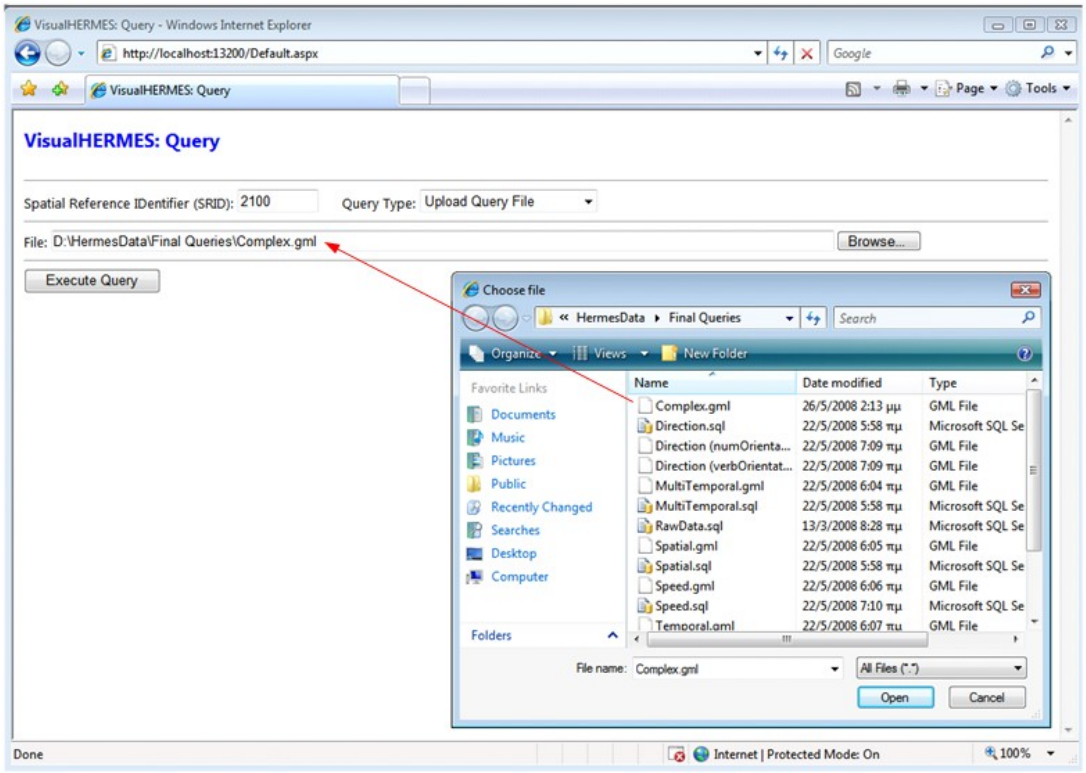


Figure 8-30: Uploading a GML query file

The process of uploading a query file is rather intuitive. All a user has to do, is via the corresponding *Browse* button, to select the appropriate file, or type the full file's path into the corresponding *File* field. VisualHERMES will validate and parse the uploaded document, as to extract the required information.

The SQL and GML equivalents for the above query are illustrated in Figure 8-31 and Figure 8-32 respectively.

```
SELECT
    a.object_id,
    a.traj_id,
    a.mpoint.at_period
    (
        tau_tll.d_period_sec
        (
            tau_tll.D_Timepoint_Sec
            (
                2001,02,12,17,45,21
            ),
            tau_tll.D_Timepoint_Sec
            (
                2001,02,12,17,45,51
            )
        )
    ).to_clob() AS trajectory
FROM
    mpoints a
WHERE
    a.mpoint.f_avg_speed()
    BETWEEN
        5.00
    AND
        6.00
```

Figure 8-31: Complex SQL query

```
<?xml version="1.0" encoding="utf-8" ?>
<query queryType="NotSet">
      <complex>
            <temporal>
                  <timePeriod>
                        <begin>2001,02,12,17,45,21</begin>
                        <end>2001,02,12,17,45,51</end>
                  </timePeriod>
            </temporal>
            <avgspeed>
                  <lBoundSpeed>5.00</lBoundSpeed>
                  <uBoundSpeed>6.00</uBoundSpeed>
            </avgspeed>
      </complex>
</query>
```

Figure 8-32: Complex GML query

The user is presented with two separate links pointing to the just created results files, as Figure 8-33 illustrates.

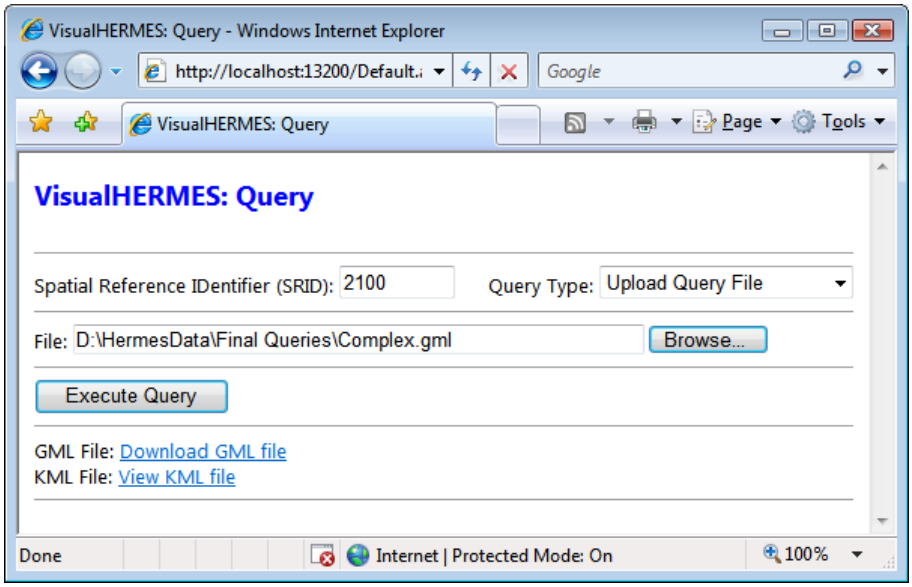


Figure 8-33: GML query file results page

Each of the two files produced for the Spatial Intersection query type is illustrated in Figure 8-34 for the GML file and in Figure 8-35 for the KML one.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gml:featureCollection xmlns:gml="http://www.opengis.net/gml">
 <gml:featureMember>
  <gml:name>Trajectory ID: 6 - Object ID: 0420</gml:name>
  <gml:description>Trajectory Segment: 1</gml:description>
  <gml:TimePeriod>
   <gml:begin>2001/02/12T17:45.21</gml:begin>
   <gml:end>2001/02/12T17:45.51</gml:end>
  </gml:TimePeriod>
  <gml:LineString>
   <gml:posList>37.94513 23.77097 37.94306 23.76831</gml:posList>
  </gml:LineString>
 </gml:featureMember>
</gml:featureCollection>
```

Figure 8-34: Complex query GML results

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
    <Document>
        <Folder>
            <name>VisualHermes</name>
            <Style id="blueLine">
                <LineStyle>
                    <color>ffff0000</color>
                    <width>3</width>
                </LineStyle>
            </Style>
            <Placemark>
                <name>Trajectory ID: 6 - Object ID: 0420</name>
                <description>Trajectory Segment: 1</description>
                <TimeSpan>
                    <begin>2001-02-12T17:45:21Z</begin>
                    <end>2001-02-12T17:45:51Z</end>
                </TimeSpan>
                <styleUrl>#blueLine</styleUrl>
                <LineString>
                    <altitudeMode>relative</altitudeMode>
                    <coordinates>
                        23.77097,37.94513,0
                        23.76831,37.94306,0
                    </coordinates>
                </LineString>
            </Placemark>
        </Folder>
    </Document>
</kml>
```

Figure 8-35: Complex query KML results

The user is able to visualize the KML results file via Google Maps, as depicted in Figure 8-36.

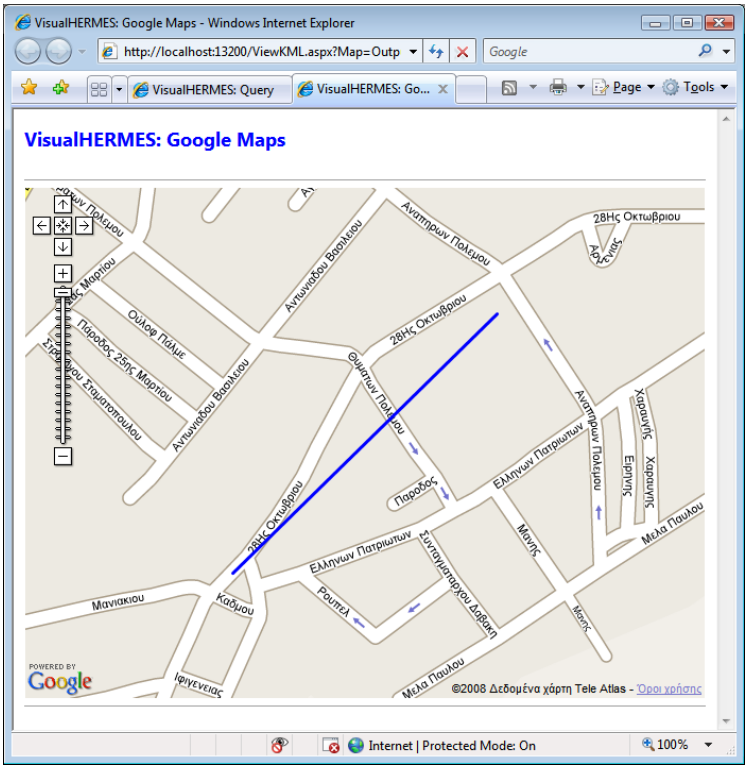


Figure 8-36: GML query file visualized KML data

8.9 Conclusion

This was a brief demonstration, in the nature of a case study, for VisualHERMES prototype. The original dataset belongs to R-tree Portal (R-tree Portal, 2005) representing school buses trajectories in Athens. All the possible querying and resulting methods were discussed, but VisualHERMES is not a closed project. Many other queries can be modeled in its web interface, setting a larger variety of possible uses to HERMES.

9 Conclusions

GML is an open standard, based on XML technology (Bray, et al., 2006), which describes the spatial and temporal data transfer and storage in the Internet environment. Latest advances for extending the standard's model in version 3 (OGC, 2004) bearing in topologies representation, three-dimensional geometries, surfaces and moving objects, add new dynamics to the acceptance and adoption of GML by the community. The success of such acceptance, therefore, has evolved GML in a first class solution to the problem of exchanging data between heterogeneous user groups.

XML data may be visualized using various methods. To visualize spatial and temporal information, which are based on XML, in a web browser, requires transforming them into a graphical form, it can interpret. There are various options, which provide solutions in this specific application domain, each of which bears its own unique features. In the current research, we selected the KML formatting standard (Google, 2008), because of the proximity to that of XML and because there are already implemented engines for results visualization, without requiring the installation of additional software (plug-ins). One such solution is the Maps service provided by Google Inc. (Google Maps, 2008).

XSLT (Clark, 1999) is a recommendation of W3C, for transforming an XML document to another. Given that GML is a representation based on XML, the XSLT is able to convert GML documents into KML. The respective conversion rules are defined within XSLT stylesheets. With the use of stylesheets, we can obtain additional information from the original GML document, which may be introduced in the final KML document.

Technically, it is possible to develop software for conversion of GML data in any form of graphic representation, but the conversion from GML in KML, using XSLT, brings some advantages. It makes use of XML, a technology on which many functions of the current Internet are based and is also supported by the largest software vendors. The production process of various representations is relatively simple because XML allows the separation of content data from presentation data. In addition, where the GML data make use of a predefined schema, any set of data can be represented on the same stylesheet.

HERMES (Pelekis, et al., 2006) is an extension to the Spatial package of Oracle's 10g object-relational database management system. Through this expansion, we are able to manage spatio-temporal data (trajectories) for moving objects, which alter their position and/or size, periodically or continuously. In addition, it provides the necessary infrastructure for supporting the queries' mission for the moving objects managed, using spatial and temporal operators. Due to the fact that HERMES system, acts as a data source for moving objects' trajectories, in third applications, it seemed appropriate to implement a wrapper, which would be able to format such information in terms of GML, before their mission to the end user, with the ultimate aim of extending system's interoperability. Furthermore, a user can receive a visualized presentation of the results wanted, on electronic maps, using Google Maps services, via a thin client application, such as a web browser. Therefore, data are formatted in accordance with the KML standard.

For this thesis's needs, VisualHERMES prototype has been designed and implemented. Through this system, a user is able to connect to a web site, from where is able to send queries to HERMES, either by choosing from a set of predefined formulas, or by sending a GML query file and take the results of that query, based on GML standard. Finally, she is able to get a visualized (KML-based) version of the results, projected on an electronic map using Google Maps service. All the above functionality has been integrated under an intuitive web application.

9.1 Open issues

Although, VisualHERMES acts as a wrapper between HERMES and end users, providing only selection queries, it is not limited to this functionality. Through the appropriate extensions and modifications in application's source code, the prototype will be able to accommodate several query types, such as insertion queries. Thus, it will be possible for a GML file to be inserted, as far as their underlying values are concerned, into HERMES, in a more bulk fashion. It has to be noted that during insertion queries wrapper's code has to provide all the transactional mechanisms, thus providing a more effective environment.

An interesting variation of VisualHERMES would be the development of the wrapper for PDAs. These devices provide the important advantage of increased portability giving thus access to the service

of VisualHERMES, literally from any place. Furthermore, the majority of such devices are equipped with GPS receivers. Given that there will be, sometime in the future, the ability to feed HERMES directly with data from a GPS, a user will be able to send spatio-temporal data using her device and receive visualized information via VisualHERMES.

It has to be noted that in many cases the visualized results are not accurate as projected on road maps; see for example Figure 8-14, Figure 8-21 and Figure 8-36. This awkward behavior is either to errors in transmitting the exact location of a moving object from the GPS, or imperfections in the road network, as it appears on the map. The solution to this problem is the on demand use of the so-called Map Matching algorithms (Brakatsoulas, et al., 2005). These algorithms are able to adjust a trajectory, by changing, the original data, so that the final result of a visualized trajectory coincides in the current road network that appears on the map.

Bibliography

Abdelguerfi, M., Givaudan, J., Shaw, K., & Ladner, R. (2002). The 2-3TR-tree, a trajectory-oriented index structure for fully evolving valid-time spatio-temporal dataasets. *Proceedings of the 10th International Symposium on Advances in Geographic Information Systems (ACMGIS '02)* (pp. 29-34). McLean, Virginia, USA: ACM Press.

Adler, S., Berglund, A., Caruso, J., Deach, S., Graham, T., Grosso, P., et al. (2001). *Extensible Stylesheet Language (XSL) Version 1.0.* World Wide Web Consortium, Recommendation REC-xsl-20011015.

Agarwal, P. K., Guibas, L. J., Edelsbrunner, H., Erickson, J., Isard, M., Harpeled, S., et al. (2002). Algorithmic issues in modeling motion. *ACM Computing Surveys , 34* (4), 550-572.

Altova. (2005). *XMLSpy - XML Editor*. Retrieved from http://www.altova.com/products/xmlspy/xml_editor.html

Bidochko, A., & Firman, O. (2005). *Map Builder::Rapid mashup development tool for Google and Yahoo! Maps*. Retrieved from http://www.mapbuilder.net/

Bishr, Y. (1997). *Semantic Aspect if Interoperable GIS (PhD Thesis).* Wageningen Agricultural University and ITC.

Brakatsoulas, S., Pfoser, D., Randall, S., & Wenk, C. (2005). On map-matching vehicle tracking data. *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05)* (pp. 853 - 864). Trondheim, Norway: ACM Press.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2006). *Extensible Markup Language (XML) 1.0 (Fourth Edition).* World Wide Web Consortium, Recommendation REC-xml20060816.

Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., Yergeau, F., & Cowan, J. (2006). *Extensible Markup Language (XML) 1.1 (Second Edition).* World Wide Web Consortium, Recommendation REC-xml11-20060816.

Carlisle, D., Ion, P., Miner, R., & Poppelier, N. (2003). *Mathematical Markup Language (MathML) 2.0 (Second Edition).* World Wide Web Consortium, Recommendation REC-MathML2-20031021.

Clark, J. (1999). *XSL Transformations (XSLT) 1.0.* World Wide Web Consortium, Recommendation REC-xslt-19991116.

Clark, J., & DeRose, S. (1999). *XML Path Language (XPath) 1.0.* World Wide Web Consortium, Recommendation REC-xpath-19991116.

Cooper, M. (2003, July). Antennas get smart. *(283(7))* , 48-55. Scientific American.

Cox, S., Cuthbert, A., Lake, R., & Martell, R. (2004). *Geographic Markup Language (GML) 3.1.1.* OGC Recommendation Paper.

Damiani, M. L., de Macebo, J. A., Parent, C., Porto, F., Spaccapietra, S., & Vangenot, C. (2007). A conceptual view on trajectories. *Proceedings of the 26th International Conference on Conceptual Modeling.* Aucklan, New Zeland: Springer.

de By, R. A., & Knippers, R. A. (2001). *Principles of Geographic Information Systems (2nd Edition).* Enschede, The Netherlands: The International Institute for Aerospace Survey and Earth Sciences.

Deng, M., & Zhang, F. (2002). Spatial temporal queries and triggers for managing moving objects. *Proceedings of the 6th East-European Conference on Advances in Databases and Information Systems (ADBIS '02)* (pp. 88-97). Bratislava, Slovakia: Springer-Verlag.

DeRose, S., Maler, E., & Orchard, D. (2001). *XML Linking Language (XLink) Version 1.0.* World Wide Web Consortium, Recommendation REC-xlink-20010627.

Deutsch, P. (1996, May). *RFC 1951 - DEFLATE Compressed Data Format Specification.* Retrieved from http://www.faqs.org/rfcs/rfc1951.html

DotNet Zip. (2007, October). *DotNet Zip Library*. Retrieved from http://www.codeplex.com/DotNetZip

Engelschall, R. S., & Barbier, D. (2001). *Website META Language (WML) http://thewml.org/.*

EPSG. (2008). *OGP Surveying & Positioning Committee*. Retrieved from http://www.epsg.org/

Erwig, M., Guting, R. H., Schneider, M., & Vazirgiannis, M. (1999). Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *Geoinformatica , 3* (3), 269-296.

Erwig, M., Schneider, M., & Guting, R. H. (1997). *Temporal and spatio-temporal data models and their expressive power.* Chorochronos Research Project.

Evjenet, B. (2007). *Professional XML.* Indianapolis, Indiana, USA: Wrox Press.

Google Earth. (2008). *Google Earth*. Retrieved from http://earth.google.com/

Google. (2008). *Keyhole Markup Language 2.2*. Retrieved from http://code.google.com/apis/kml/

Google Maps API. (2004). *Google Maps API Concepts - Google Maps API - Google Code*. Retrieved from http://code.google.com/apis/maps/documentation/index.html

Google Maps. (2008). *Google Maps.* Retrieved from http://maps.google.com/

Guidi, D. (2007). *Proj.Net*. Retrieved from http://www.codeplex.com/ProjNET

Guting, R. H. (1994). An introduction to spatial database systems. *The VLDB Journal , 3* (4), 357-399.

Guting, R. H., Bohlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., et al. (2000). A foundation of representing and querying moving objects. *ACM Transactions on Database Systems , 25* (1), 1-42.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *Proceedings of ACM SIGMOD Conference on Management of Data.*

Harri, S., Mena, E., & Illaramendi, A. (2002). Monitoring Continuous Location Queries Using Mobile Agents. *Proceedings of the 6th East- European Conference on Advances in Database and Information Systems (ADBIS).* Bratislava, Slovakia.

Hilton, B. N. (2007). *Emerging Spatial Information Systems and Applications.* USA: IDEA Group Publishing.

Houldling, S. W. (2001). XML - An opportunity for <meaningful> data standards in the geosciences. *Computers and Geosciences , 27*, 839-849.

IBM. (2008). *IBM - Data server - Informix - online processing*. Retrieved from http://www-306.ibm.com/software/data/informix/

Ingres. (2008). *Enterprise Open Source Database Ingres*. Retrieved from http://www.ingres.com/

International Organization for Standarization. (1988). *SGML Document Interchange Format (SDIF) ISO 9069.* Information Processing: SGML Support Facilities.

International Organization for Standarization. (1986). *Standard Generalized Markup Language (SGML) ISO 8879.* Text and Office Systems, Information Processing.

ITC Educational Textbook Series. (2001). *Principles of Geographic Information Systems: An Introductory Textbook* (Second Edition ed.). (R. A. By, Ed.) ITC.

Jensen, C. S. (2002). Research challenges in location-enabled M-services. *Proceedings of the 3rd International Conference on Mobile Data Management (MDM '02)* (pp. 3-7). IEEE Computer Society.

Jensen, C. S., Friis-Christensen, A., Pedersen, T. B., Pfoser, D., Saltenis, S., & Tryfona, N. (2001). Location-based services - a database perspective. *Proceedings of the 8th Scandinavian Research Conference in Geographical Information Science (ScanGIS '01)*, (pp. 22-25). As, Norway.

Joshi, B. (2007). *Pro .NET 2.0 XML.* USA: Apress.

Kay, M. H. (2002). *Instant SAXON*. Retrieved from http://saxon.sourceforge.net/saxon6.5.2/instant.html

Kothuri, R., Godfrind, A., & Beinat, E. (2007). *Pro Oracle Spatial for Oracle Database 11g.* New York, USA: Apress.

Lake, R. (2001). *GML - Enable the GeoSpatial Web.*

Le Hegaret, P., & Le Hors, A. (2004). *Document Object Model Level 3.* World Wide Web Consortium.

Lilley, C. (2002). *Graphics Activity Statement.* World Wide Web Consortium, Activity.

Lu, C. T., Santos Jr., R. F., Spirada, L. N., & Kou, Y. (2007). Advances in GML for Geospatial Applications. *Geoinformatica , 11*, 131-157.

Marketos, M., Frentzos, E., Ntoutsi, I., Pelekis, N., Raffaeta, A., & Theodoridis, Y. (2008). Building real-world trajectory warehouses. *Proceedings of the 7th International ACM SIGMOD Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '08).* Vancouver, Canada.

Mathews, B., Lee, D., Dister, B., Bowler, J., Cooperstein, H., Jindal, A., et al. (1998). *VML - the Vector Markup Language.* Retrieved from http://www.w3.org/TR/1998/NOTE-VML-19980513

Microsoft ASP.NET. (2008). *Microsoft ASP.NET*. Retrieved from http://www.asp.net/

Microsoft Corporation. (2008). *.NET Framework Developer Center*. Retrieved from http://msdn.microsoft.com/en-us/netframework/default.aspx

Microsoft Virtual Earth. (2008). *Microsoft Virtual Earth: The Integrated Mapping, Imaging, Search and Location Platform.* Retrieved from http://www.microsoft.com/virtualearth/

Mokbel, M., Ghanem, T., & Aref, W. (2003). "Spatio-Temporal Access Methods", Special Issue on Infrastructure for Research in Spatio-Temporal Query Processing. *IEEE Data Engineering Bulletin , 26* (2), 40-49.

Murray-Rust, P., Rzepa, H. S., & Leach, C. (1995). *Chemical Markup Language (CML).* Imperial College London.

OGC. (2008). *OGC 05-033r9: Simple Features Schema.* Retrieved from http://schemas.opengis.net/gml/3.1.1/base/feature.xsd

OGC. (2004). *OpenGIS Geography Markup Language*. Retrieved from http://www.opengeospatial.org/standards/gml

OGC. (2008). *OpenGIS Standards and Specifications*. Retrieved from http://www.opengeospatial.org/standards

OGP. (2008). *International Association of Oil & Gas Producers*. Retrieved from http://www.epsg.org/Geodetic.html

Oracle Corp. (2003). *Oracle © Data Cartridge Developer's Guide 10g Release 1 (10.1).* Retrieved from http://www.oracle.com/pls/db10g/portal.portal_demo3?selected=7

Oracle Corp. (2003). *Oracle © Spatial User's Guide and Reference 10g Release 1 (10.1).* Retrieved from http://www.oracle.com/pls/db10g/portal.portal_demo3?selected=7

Oracle Corp. (2008). *Oracle Spatial, Locator, and Location-Based Services*. Retrieved from http://www.oracle.com/technology/products/spatial/index.html

Oracle Corp. (2008). *Oracle® Database Documentation Library 10g Release 1 (10.1)*. Retrieved from http://otn.oracle.com/pls/db10g/

Pelekis, N., Theodoridis, Y., Vosinakhs, S., & Panayiotopoulos, T. (2006). HERMES - A Framework for Location-Based Data Management. *Proceedings of the 10th International Conference on Extending Database Technology (EDBT '06).* Munich, Germany.

Pelekis, N., Theodoulidis, B., Kopanakis, I., & Theodoridis, Y. (2004). Literature Review of Spatio-Temporal Database Models. *The Knowledge Engineering Journal , 19* (3), 235-274.

Pemberton, S. (2002). *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition).* World Wide Web Consortium, Recommendation REC-xhtml11-20020801.

Pfoser, D., & Jensen, C. S. (1999). Capturing the uncertainty of moving-object representations. *Proceedings of the 6th International Symposium on Advances in Spatial Databases (SSD '99)* (pp. 111-132). Hong Kong, China: Springer-Verlag.

Pfoser, D., & Jensen, C. S. (2003). Indexing of network constrained moving objects. *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems Database (ACMGIS '03)* (pp. 25-32). New Orleans, Louisiana USA: ACM Press.

Pfoser, D., Jensen, C., & Theodoridis, Y. (2000). Novel approaches in query processing for moving objects. *Proceedings of Very Large Database (VLDB)*, (pp. 395-406).

PostGIS. (2008). *PostGIS: Home*. Retrieved from http://www.postgis.org/

Procopiuc, C. M., Agarwal, P. K., & Har-Peled, S. (2002). STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects. *Proceedings of the Workshop on Alg. Eng. and Experimentation (ALENEX)*, (pp. 178-193).

Raggett, D., Le Hors, A., & Jacobs, I. (1999). *Document Type Definition.* World Wide Web Consortium, Recommendation REC-html401-19991224.

Raggett, D., Le Hors, A., & Jacobs, I. (1999). *HTML 4.01 Specification.* World Wide Web Consortium, Recommendation REC-html401-19991224.

Robinson, A. H., & Morrison, J. L. (1995). *Elements of Cartography (6th Edition).* USA: John Wiley & Sons Inc.

R-tree Portal. (2005). *Spatio-temporal (trajectory) datasets.* Retrieved from http://www.rtreeportal.org/datasets/trajectories/buses.zip

Saltenis, S., & Jansen, C. S. (2002). Indexing of Moving Objects for Location-Based Services. *Proceedings of the International Conference on Data Engineering (ICDE).*

Saltenis, S., Jensen, C. S., Leutenegger, S. T., & Lopez, M. A. (2000). Indexing the positions of continuously moving objects. *Proceedings of 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)* (pp. 331-342). Dallas, Texas, USA: ACM Press.

Sun Microsystems. (1994). *Developer Resources for Java Technology*. Retrieved 2008, from http://java.sun.com/

Sun Microsystems. (1994). *Sun Microsystems*. Retrieved 2008

Thompson, H., Beech, D., Maloney, M., & Mendelsohn, N. (2004). *XML Schema Part 1: Structures (Second Edition).* World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028.

Tryfona, N., & Hadzilacos, T. (1997). *Evaluation of database modeling methods for geographical information systems.* Chorochronos Research Project.

Tryfona, N., & Hadzilacos, T. (1997). *Logical data modeling of spatio-temporal applications: Definitions and a model.* Chorochronos Research Project.

Vazirgiannis, M., & Wolfson, O. (2001). A spatiotemporal model and language for moving objects on road networks. *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD '01)* (pp. 20-35). Redondo Beach, California, USA: Springer-Verlag.

W3Schools. (2008). *DTD Tutorial*. Retrieved from http://www.w3schools.com/dtd/default.asp

W3Schools. (2008). *Schema Tutorial*. Retrieved from http://www.w3schools.com/schema/default.asp

W3Schools. (2008). *XML Tutorial*. Retrieved from http://www.w3schools.com/xml/default.asp

Wolf, M., & Wicksteed, C. (1997). *Date and Time Formats.* World Wide Web Consortium, Note NOTE-datetime-19980827 .

Wolfson, O. (2002). Moving objects information management: The database challenge. *Proceedings of the 5th Workshop on Next Generation Information Technologies and Systems (NGITS '02)* (pp. 75-89). Caesarea, Israel: Springer-Verlag.

Wolfson, O., Sistla, A. P., Xu, B., Zhou, J., Chamberlain, S., Yesha, Y., et al. (1999). Tracking moving objects using database technology in DOMINO. In R. Y. Pinter, & S. Tsur (Ed.), *Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems (NGITS '99). Volume 1649 of Lecture Notes in Computer Science (LNCS)*, pp. 112-119. Zikhron-Yaakov, Israel: Springer-Verlag.

Wolfson, O., Sistla, P., Chamberlain, S., & Yesha, Y. (1999). Updating and querying databases that track mobile units. *Distributed and Parallel Databases , 7* (3), 257-387.

Wolfson, O., Sistla, P., Xu, B., Zhou, J., & Chamberlain, S. (1999). DOMINO: Databased fOr MovINg Objects tracking. *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)* (pp. 547-549). Philadelphia, Pennsylvania, USA: ACM Press.

Wolfson, O., Xu, B., Chamberlain, S., & Jiang, L. (1998). Movings objects databases: Issues and solutions. *Proceedings of the 10th International Conference on Scientific and Statistical Database Management (SSDBM '98)* (pp. 111-122). Capri, Italy: IEEE Computer Society.

Zhu, H., Su, J., & Ibarra, O. H. (2002). Trajectory queries and octagons in moving object databases. *Proceedings of 2002 ACM CIKM International Conference on Information and Knowledge Management (CIKM '02)*, (pp. 413-421). McLean, Virginia, USA.

Appendix

A VisualHERMES GML Schema Definition File

GML query files that are being uploaded into VisualHERMES, must meet the following query definition schema, in order to be processed by the wrapper.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="QuerySchema" elementFormDefault="qualified"
xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:simpleType name="PosList">
            <xs:restriction base="xs:string">
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="ObjectID">
            <xs:restriction base="xs:string">
                  <xs:maxLength value="20" />
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="TrajectoryID">
            <xs:restriction base="xs:positiveInteger" />
      </xs:simpleType>
      <xs:simpleType name="UBoundSpeed">
            <xs:restriction base="xs:double">
                  <xs:minInclusive value="0.0" />
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="LBoundSpeed">
            <xs:restriction base="xs:double">
                  <xs:minInclusive value="0.0" />
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="Begin">
            <xs:restriction base="xs:string">
                  <xs:pattern
value="\d{4}/\d{2}/\d{2}T\d{2}:\d{2}.\d{2}" />
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="End">
            <xs:restriction base="xs:string">
                  <xs:pattern
value="\d{4}/\d{2}/\d{2}T\d{2}:\d{2}.\d{2}" />
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="QueryType">
            <xs:restriction base="xs:string">
                  <xs:enumeration value="NotSet" />
                  <xs:enumeration value="Trajectory" />
                  <xs:enumeration value="Spatial" />
                  <xs:enumeration value="Temporal" />
                  <xs:enumeration value="AverageSpeed" />
            </xs:restriction>
      </xs:simpleType>
      <xs:complexType name="LinearRing">
            <xs:sequence>
                  <xs:element name="posList" type="PosList" />
            </xs:sequence>
```

```
		</xs:complexType>
		<xs:complexType name="Exterior">
			<xs:sequence>
				<xs:element name="linearRing" type="LinearRing"
/>
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="Polygon">
			<xs:sequence>
				<xs:element name="exterior" type="Exterior" />
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="TimePeriod">
			<xs:sequence>
				<xs:element name="begin" type="Begin" />
				<xs:element name="end" type="End" />
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="Query">
			<xs:sequence>
				<xs:sequence>
					<xs:element name="trajectory"
type="Trajectory" minOccurs="0" />
					<xs:element name="spatial" type="Spatial"
minOccurs="0" />
					<xs:element name="temporal"
type="Temporal" minOccurs="0" />
					<xs:element name="avgspeed"
type="AverageSpeed" minOccurs="0" />
					<xs:element name="complex" minOccurs="0"
/>
				</xs:sequence>
			</xs:sequence>
			<xs:attribute name="queryType" type="QueryType"
use="required" />
		</xs:complexType>
		<xs:complexType name="Trajectory">
			<xs:sequence>
				<xs:element name="objectID" type="ObjectID" />
				<xs:element name="trajectoryID"
type="TrajectoryID" />
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="Spatial">
			<xs:sequence>
				<xs:element name="polygon" type="Polygon" />
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="Temporal">
			<xs:sequence>
				<xs:element name="timePeriod" type="TimePeriod"
/>
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="AverageSpeed">
			<xs:sequence>
				<xs:element name="lBoundSpeed"
type="LBoundSpeed" />
				<xs:element name="uBoundSpeed"
```

```
type="UBoundSpeed" />
            </xs:sequence>
      </xs:complexType>
      <xs:element name="query" type="Query">
      </xs:element>
</xs:schema>
```

B VisualHERMES eXtensible Stylesheet Language Transformations File

Each GML results file produced by VisualHERMES has to be transformed according to KML specification, in order to be visualized by Google Earth, Google Maps etc. The transformation rules are defined in the following file:

```
<?xml
	version="1.0"
	encoding="utf-8"
?>

<xsl:stylesheet
	version="1.0"
	xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
	xmlns:gml="http://www.opengis.net/gml"
	xmlns:msxsl="urn:schemas-microsoft-com:xslt"
	xmlns:ext="http://goutsidis.gr/extension"
	exclude-result-prefixes="gml msxsl ext"
>

<xsl:output
	method="xml"
	version="1.0"
	encoding="utf-8"
	indent="yes"
	media-type="application/vnd.google-earth.kml+xml"
/>

<msxsl:script language="C#" implements-prefix="ext">
	<![CDATA[
	public string formatPosList(string value)
	{
		string[] res = value.Split(' ');
		return res[1] + "," + res[0] + ",0 " + res[3] + "," +
res[2] + ",0";
	}

	public string formatTimeStamp(string value)
	{
		string res = value.Replace("/", "-");
		return res.Replace(".", ":") + "Z";
	}
	]]>
</msxsl:script>

	<xsl:template match="/gml:featureCollection">
		<xsl:element name="kml"
namespace="http://earth.google.com/kml/2.2">
			<xsl:element name="Document">
				<xsl:element name="Folder">
					<xsl:element name="name">

	<xsl:text>VisualHermes</xsl:text>
					</xsl:element>
					<xsl:element name="Style">
						<xsl:attribute name="id">
```

```
            <xsl:text>blueLine</xsl:text>
                                          </xsl:attribute>
                                          <xsl:element name="LineStyle">
                                                <xsl:element
name="color">

            <xsl:text>ffff0000</xsl:text>
                                                </xsl:element>
                                                <xsl:element
name="width">

            <xsl:text>3</xsl:text>
                                                </xsl:element>
                                          </xsl:element>
                                    </xsl:element>
                                    <xsl:apply-templates />
                              </xsl:element>
                        </xsl:element>
                  </xsl:element>
            </xsl:template>

            <xsl:template match="gml:featureMember">
                  <xsl:element name="Placemark">
                        <xsl:apply-templates select="gml:name" />
                        <xsl:apply-templates select="gml:description" />
                        <xsl:apply-templates select="gml:TimePeriod" />
                        <xsl:apply-templates select="gml:Point" />
                        <xsl:apply-templates select="gml:LineString" />
                        <xsl:apply-templates select="gml:Polygon" />
                  </xsl:element>
            </xsl:template>

            <xsl:template match="gml:name">
                  <xsl:element name="name">
                        <xsl:value-of select="." />
                  </xsl:element>
            </xsl:template>

            <xsl:template match="gml:description">
                  <xsl:element name="description">
                        <xsl:value-of select="." />
                  </xsl:element>
            </xsl:template>

            <xsl:template match="gml:TimePeriod">
                  <xsl:element name="TimeSpan">
                        <xsl:element name="begin">
                              <xsl:value-of
select="ext:formatTimeStamp(gml:begin)" />
                        </xsl:element>
                        <xsl:element name="end">
                              <xsl:value-of
select="ext:formatTimeStamp(gml:end)" />
                        </xsl:element>
                  </xsl:element>
            </xsl:template>

            <xsl:template match="gml:Point">
                  <xsl:element name="Point">
```

```
                        <xsl:element name="coordinates">
                              <xsl:value-of select="." />
                        </xsl:element>
                  </xsl:element>
            </xsl:template>

            <xsl:template match="gml:LineString">
                  <xsl:element name="styleUrl">
                        <xsl:text>#blueLine</xsl:text>
                  </xsl:element>
                  <xsl:element name="LineString">
                        <xsl:element name="altitudeMode">
                              <xsl:text>relative</xsl:text>
                        </xsl:element>
                        <xsl:element name="coordinates">
                              <xsl:value-of
      select="ext:formatPosList(gml:posList)" />
                        </xsl:element>
                  </xsl:element>
            </xsl:template>

            <xsl:template match="gml:Polygon">
                  <xsl:element name="Polygon">
                        <xsl:element name="tessellate">
                              <xsl:text>1</xsl:text>
                        </xsl:element>
                        <xsl:element name="outerBoundaryIs">
                              <xsl:element name="LinearRing">
                                    <xsl:element name="coordinates">
                                          <xsl:value-of
      select="ext:formatPosList(gml:exterior/gml:LinearRing/gml:coordina
      tes)" />
                                    </xsl:element>
                              </xsl:element>
                        </xsl:element>
                        <xsl:element name="innerBoundaryIs">
                              <xsl:element name="LinearRing">
                                    <xsl:element name="coordinates">
                                          <xsl:value-of
      select="ext:formatPosList(gml:interior/gml:LinearRing/gml:coordina
      tes)" />
                                    </xsl:element>
                              </xsl:element>
                        </xsl:element>
                  </xsl:element>
            </xsl:template>

      </xsl:stylesheet>
```

C HERMES TO_CLOB() Function

In order for HERMES to be able to handle large trajectory datasets that can be sent to VisualHERMES wrapper, a new function has to be implemented and added.

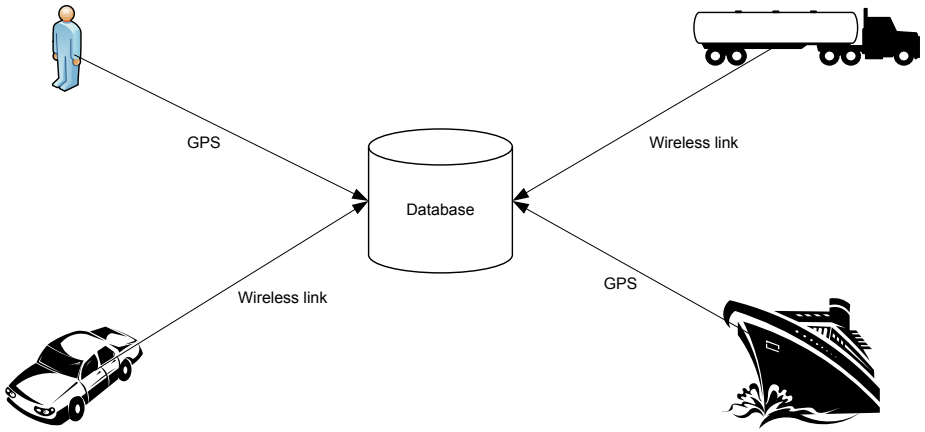
```
MEMBER FUNCTION to_clob
RETURN CLOB IS

      i   PLS_INTEGER;
      str CLOB;

BEGIN
      FOR i IN u_tab.FIRST .. u_tab.LAST LOOP

            str := str ||
                  '(' ||
                  SUBSTR(TO_CHAR (u_tab(i).m.xi), 0, 10) ||
                  ',  ' ||
                  SUBSTR(TO_CHAR (u_tab(i).m.yi), 0, 10) ||
                   ') - (' ||
                   SUBSTR(TO_CHAR (u_tab(i).m.xe), 0, 10) ||
                   ',  ' ||
                   SUBSTR(TO_CHAR (u_tab(i).m.ye), 0, 10) ||
                   ') # ' ||
                   TO_CHAR(u_tab(i).p.b.M_Y) ||
                   '/' ||
                   TO_CHAR(u_tab(i).p.b.M_M) ||
                   '/' ||
                   TO_CHAR(u_tab(i).p.b.M_D) ||
                   ' - ' ||
                   TO_CHAR(u_tab(i).p.b.M_H) ||
                   ':' ||
                   TO_CHAR(u_tab(i).p.b.M_MIN) ||
                   ':' ||
                   TO_CHAR(u_tab(i).p.b.M_SEC) ||
                   TO_CHAR(u_tab(i).p.e.M_Y) ||
                   '/' ||
                   TO_CHAR(u_tab(i).p.e.M_M) ||
                   '/' ||
                   TO_CHAR(u_tab(i).p.e.M_D) ||
                   ' - ' ||
                   TO_CHAR(u_tab(i).p.e.M_H) ||
                   ':' ||
                   TO_CHAR(u_tab(i).p.e.M_MIN) ||
                   ':' ||
                   TO_CHAR(u_tab(i).p.e.M_SEC);

      END LOOP;

      return str;
END;
```

D Εκτενής Περίληψη στην Ελληνική

Εισαγωγή

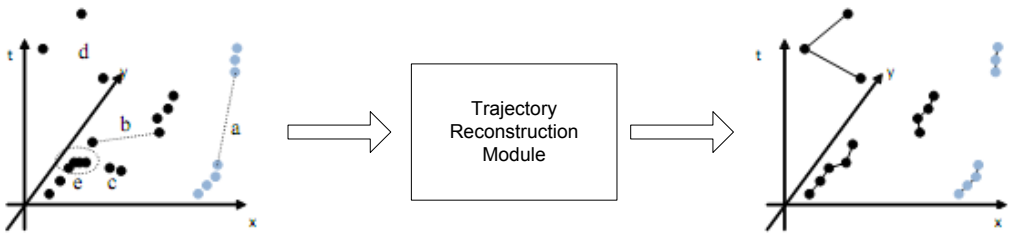
Τρέχουσα κατάσταση

Στις ημέρες μας οι κινητοί χρήστες είναι περισσότεροι από κάθε άλλη φορά. Επιπρόσθετα, οι εξελίξεις στα σύγχρονα συστήματα διαχείρισης βάσεων δεδομένων (Database Management Systems – DBMS) έχουν καταστήσει εφικτή την αποθήκευση κάθε είδους πληροφορίας, που αφορά σε διάφορες πηγές κινούμενων αντικειμένων (Moving Objects – MO). Για παράδειγμα, με τη βοήθεια των (Global Positioning Systems) GPS συστημάτων πλοήγησης και των κινητών-ασύρματων επικοινωνιών είναι δυνατή η αποθήκευση πληροφοριών που σχετίζονται με τη γεωγραφική θέση ενός κινούμενου αντικειμένου σε συνάρτηση με τον χρόνο (Εικόνα 1).



Εικόνα 1: Κινούμενα αντικείμενα

Σε ένα δεύτερο επίπεδο, βρίσκεται ο συνδυασμός αυτού του είδους της πληροφορίας από την σκοπιά του κινούμενου αντικειμένου, δημιουργώντας τις επονομαζόμενες τροχιές κινούμενων αντικειμένων (Moving Object Trajectories – MOT). Βέβαια, η διαδικασία παραγωγής τροχιών από ένα σύνολο σημείων σε συνάρτηση με τον χρόνο (Trajectory Reconstruction), απαιτεί την ύπαρξη κατάλληλου λογισμικού, το οποίο βρίσκεται στην κορυφή του συστήματος διαχείρισης βάσεων δεδομένων και δύναται να μοντελοποιήσει τις απαιτούμενες δομές δεδομένων για την υποστήριξη χωρικών και χρονικών χαρακτηριστικών (Εικόνα 2).

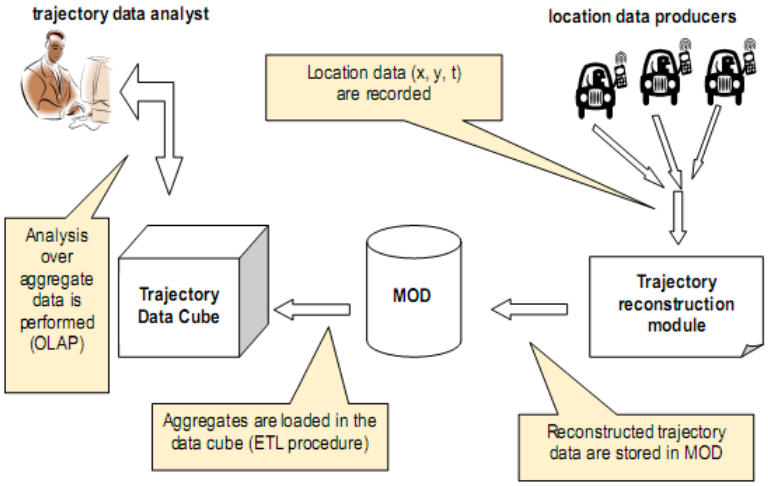


Εικόνα 2: Απλές τοποθεσίες και τροχιές που δημιουργούνται

Το επόμενο στάδιο είναι η τροφοδοσία της βάσης δεδομένων κινούμενων αντικειμένων (Moving Object Database – MOD) με όλες τις απαραίτητες πληροφορίες, ώστε να καταστούν εφικτές οι διάφορες λειτουργίες εξόρυξης δεδομένων (Data Mining – DM) και λαμβάνοντας υπόψη το εκάστοτε γεωγραφικό πλαίσιο, όπως επίσης και τα διάφορα γεωγραφικά επίπεδα, θα επιτρέψουν την εξαγωγή χρήσιμων συμπερασμάτων και τελικά γνώσης, για το συγκεκριμένο πεδίο έρευνας (Εικόνα 3).

Όπως γίνεται βέβαια αντιληπτό, η εξόρυξη γνώσης, αποτελεί έναν μόνο τομέα, ενός ευρύτερου πεδίου έρευνας που σχετίζεται με τις χωρο-χρονικές πληροφορίες κινούμενων αντικειμένων. Άλλοι

τομείς στο εν λόγω πεδίο, μπορεί να περιλαμβάνουν την εξαγωγή προτύπων, την οπτικοποίηση της πληροφορίας κ.ά.



Εικόνα 3: Αρχιτεκτονική Βάσεων Δεδομένων Κινούμενων Αντικειμένων (Marketos, et al., 2008)

Ένα κρίσιμο σημείο για τη λειτουργία ενός συστήματος, το οποίο αφενός μεν διαχειρίζεται τροχιές κινούμενων αντικειμένων, αφετέρου δε αποτελεί την πηγή παροχής τέτοιων πληροφοριών σε τρίτα μέρη, είναι η μεταφορά των δεδομένων. Πιο συγκεκριμένα , η ανάγκη της δια-λειτουργικότητας σε επίπεδο διακινούμενων δεδομένων από και προς ένα χωρικό σύστημα διαχείρισης βάσεων δεδομένων (Spatial Database Management Systems – SDBMS), κρίνεται επιτακτική. Ως δια-λειτουργικότητα στην προκειμένη περίπτωση νοείται η δυνατότητα ενός τέτοιου συστήματος να αποστέλλει δεδομένα που υπόκεινται σε μια αυτο-περιγραφόμενη σήμανση, η οποία με τη σειρά της είναι αναγνωρίσιμη από κάθε ενδιαφερόμενο μέρος.

Μια τέτοια προσέγγιση προϋποθέτει τη μεσολάβηση, κατάλληλου λογισμικού, το οποίο θα έχει ως στόχο τη μετατροπή των εξερχόμενων, από το σύστημα διαχείρισης βάσεων δεδομένων, δεδομένων σε μια κοινώς αποδεκτή μορφή όπως αυτή του eXtensible Markup Language – XML (Bray, et al., 2006) και (Bray, et al., 2006). Επιπρόσθετα, τα εισερχόμενα δεδομένα καλούνται να συμμορφώνονται βάσει ενός προκαθορισμένου σχήματος, έτσι ώστε να επιτυγχάνεται η αποδοχή τους από το σύστημα διαχείρισης βάσεων δεδομένων για την περαιτέρω επεξεργασία τους. Ένα ανταγωνιστικό πλεονέκτημα που ανακύπτει από την χρήση του XML προτύπου, είναι το γεγονός ότι επειδή πρόκειται για χωρικά δεδομένα, μπορεί να γίνει χρήση του άμεσα σχετιζόμενου με αυτά προτύπου περιγραφής Geography Markup Language – GML (OGC, 2004). Με τις πιο πρόσφατες μάλιστα προσθήκες στην έκδοση 3.1.1 του GML (Cox, et al., 2004), είναι εφικτή η μοντελοποίηση τόσο χωρικών όσο και χρονικών χαρακτηριστικών, καθώς επίσης και ο χειρισμός κινούμενων αντικειμένων, μέσω των αντίστοιχων επεκτάσεων.

Η εισαγωγή μηχανισμών για τη μορφοποίηση των δεδομένων (wrappers) βέβαια, δεν περιορίζεται στην χρήση των XML και GML προτύπων. Είναι δυνατή η μοντελοποίηση των δεδομένων είτε βάσει άλλων γνωστών και κοινά αποδεκτών προτύπων, όπως αυτό του Keyhole Markup Language – KML (Google Inc., 2008), το οποίο φέρει αρκετές ομοιότητες με αυτό του GML, αλλά και η υλοποίηση προσαρμοσμένων (custom) προτύπων αναπαράστασης, σύμφωνα με τις εκάστοτε ανάγκες.

Η χρήση κοινά αποδεκτών προτύπων που βασίζονται στο XML, όπως αυτά των GML ή KML, εκτός των γνωστών πλεονεκτημάτων όπως αυτο-περιγραφή, επεκτασιμότητα, εύκολη επεξεργασία και διακίνηση, ανεξαρτησία πλατφόρμας κ.ά., έχει και ορισμένα μειονεκτήματα, τα οποία έγκεινται κυρίως στη φύση των χωρο-χρονικών δεδομένων και στον τρόπο με τον οποίο αυτά μορφοποιούνται. Αναφορικά με την XML μορφοποίηση των δεδομένων, πρέπει να επισημανθεί ότι τα GML έγγραφα τείνουν να είναι μεγάλα σε μέγεθος, κυρίως λόγω της εκτενούς ιεραρχίας που υπάρχει στο GML 3. Το γεγονός αυτό καθιστά τα εν λόγω δεδομένα πιο δύσκολα και πολύπλοκα στην επεξεργασία τους, ιδιαίτερα όταν πρόκειται για την περιγραφή μεγάλου όγκου δεδομένων. Μια λύση στο ζήτημα αυτό, θα μπορούσε να δώσει κάποια μέθοδος συμπίεσης (π.χ. αλγόριθμος Deflate (Deutsch, 1996)), η οποία όμως έχει ως άμεσο αντίκτυπο την αύξηση της πολυπλοκότητας του συστήματος και του χρόνου επεξεργασίας.

Συμπερασματικά, γίνεται κατανοητό, ότι η χρήση XML wrappers/parsers, οι οποίοι υπεισέρχονται στη διαδικασία ανταλλαγής δεδομένων από και προς ένα σύστημα διαχείρισης βάσεων δεδομένων,

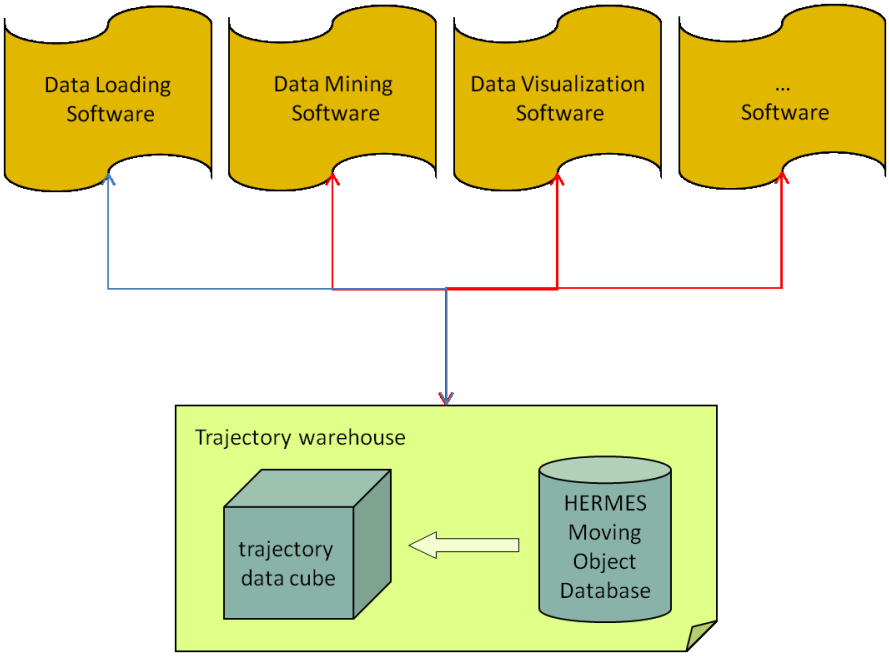
αποτελεί μια αρκετά καλή λύση στο ζήτημα στη δια-λειτουργικότητας, με απώτερο στόχο τη δημιουργία ενός *χαλαρά συνδεδεμένου* (*loosely-coupled*) συστήματος για το συγκεκριμένο σκοπό.

Αντικείμενο εργασίας

Το HERMES (Pelekis, et al., 2006) αποτελεί ένα ευέλικτο υπολογιστικό πλαίσιο (framework), ικανό να υποστηρίξει τον σχεδιασμό και την ανάπτυξη χωρο-χρονικών βάσεων δεδομένων. Επιπλέον, παρέχει όλη την απαιτούμενη υποδομή για την αποστολή ερωτημάτων σε μια βάση δεδομένων με κινούμενα αντικείμενα (MOD), των οποίων η τοποθεσία, το σχήμα και το μέγεθος μεταβάλλονται μέσα στον χρόνο.

Το σύστημα HERMES, έχει αναπτυχθεί ως επέκταση, η οποία παρέχει χωρο-χρονικές δυνατότητες στο αντικειμενο-σχεσιακό σύστημα βάσεων δεδομένων της Oracle στην έκδοση 10g (Oracle Corp., 2008). Επιπλέον, έχει σχεδιαστεί κατά τέτοιο τρόπο, ώστε να είναι δυνατή η χρήση του είτε ως αμιγώς χωρικού είτε ως χρονικού συστήματος, αλλά η κύρια συνεισφορά του έγκειται στην υποστήριξη της διαχείρισης συνεχώς κινούμενων αντικειμένων.

Στην τρέχουσα έκδοσή του, το HERMES, λειτουργεί ως αποθετήριο (repository) δεδομένων τροχιών, μέσω του οποίου ο χρήστης καταλήγει να έχει στη διάθεσή του ένα σύνολο τροχιών για το κινούμενο αντικείμενο που τον ενδιαφέρει σε μια αντικειμενο-σχεσιακή μορφή. Επιπρόσθετα, ο χρήστης μπορεί να εκτελέσει μια σειρά ερωτημάτων στα δεδομένα των τροχιών (χρησιμοποιώντας χωρο-χρονικούς τελεστές ή/και όχι) και να λάβει τα αντίστοιχα αποτελέσματα. Με τον ρόλο του, ως αποθετήριο τροχιών κινούμενων αντικειμένων, το HERMES, χρησιμοποιείται παράλληλα και ως πηγή παροχής δεδομένων σε μια σειρά επιμέρους εφαρμογών, οι οποίες επιτελούν συγκεκριμένους σκοπούς, όπως για παράδειγμα λογισμικό φόρτωσης δεδομένων, εξόρυξης γνώσης, οπτικοποίησης πληροφορίας κ.ά. (Εικόνα 4). Με την παρούσα υλοποίηση βέβαια, δημιουργείται άμεσα η ανάγκη της μορφοποίησης των δεδομένων που ανταλλάσσονται μεταξύ του αποθετηρίου και των διάφορων εφαρμογών, βάσει ενός κοινά αποδεκτού προτύπου, με απώτερο στόχο τη δημιουργία ενός πιο ευέλικτου και επεκτάσιμου συστήματος.



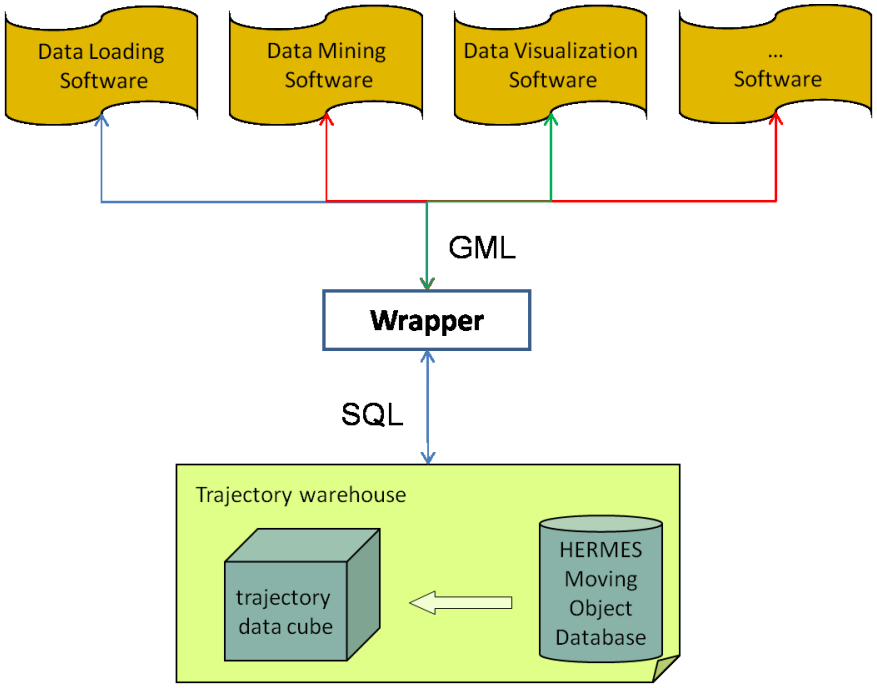
Εικόνα 4: Το σύστημα HERMES

Με κύριο στόχο την ανάπτυξη ενός πιο ανεξάρτητου και ευέλικτου συστήματος, κρίνεται αναγκαία η υλοποίηση ενός ανώτερου επιπέδου επεξεργασίας για τη σήμανση τόσο των εισερχόμενων δεδομένων, όσο και των εξερχόμενων αποτελεσμάτων, με την χρήση του GML προτύπου. Με βάση την τελευταία έκδοση του εν λόγω προτύπου (3.1.1), είναι δυνατή η αναπαράσταση και κινούμενων αντικειμένων. Η πρόταση στηρίζεται στην ιδέα της υλοποίησης ενός επιπλέον μηχανισμού, ο οποίος θα παρεμβάλλεται μεταξύ του HERMES και της εκάστοτε εφαρμογής (Εικόνα 5) και θα είναι σε θέση να:

1. Παρέχει ένα προκαθορισμένο σύνολο GML προτύπων, τα οποία θα μπορούν να χρησιμοποιηθούν από τον χρήστη για την αποστολή χωρο-χρονικών ερωτημάτων προς το

σύστημα διαχείρισης βάσεων δεδομένων και τα οποία θα μετατρέπονται, εν συνεχεία, σε SQL ερωτήματα, έτσι ώστε να είναι δυνατή η περαιτέρω επεξεργασία τους και

2. Λάβει τα SQL αποτελέσματα και πριν αυτά αποσταλούν πίσω στον χρήστη, να μετατρέπονται σε δομές δεδομένων βάσει του GML προτύπου, δίνοντας με τον τρόπο αυτό, στον τελικό χρήστη ένα αυτο-περιγραφόμενο σύνολο αποτελεσμάτων σε σχέση με το ερώτημα που εκτέλεσε.



Εικόνα 5: Επέκταση με την εισαγωγή wrapper

Με δεδομένο ότι το GML πρότυπο είναι σε θέση να περιγράψει χωρικές και μη έννοιες, καθίσταται ιδανική λύση τόσο για τη δημιουργία τροχιών κινούμενων αντικειμένων (με την χρήση *LineStrings* στην πλέον βασική τους μορφή), όσο και για τη συνοδεία των δεδομένων αυτών από άλλες, μη χωρικές, πληροφορίες, οι οποίες όμως είναι αναγκαίες για την περιγραφή του εκάστοτε αντικειμένου. Επιπλέον, το γεγονός ότι το GML πρότυπο βασίζεται στην XML φιλοσοφία μορφοποίησης δεδομένων, καθιστά δυνατή τη μετατροπή των GML δεδομένων σε οποιαδήποτε XML μορφή οπτικοποίησης, όπως Vector Markup Language – VML (Mathews, et al., 1998), KML κ.ά., εργασία που μπορεί να επωμιστεί είτε ο διακομιστής (server) είτε ο πελάτης (client). Με τον τρόπο αυτό, επιτυγχάνεται η άμεση οπτικοποίηση των αποτελεσμάτων, που επιστρέφονται από το HERMES, επί χάρτου, με χρήση μηχανών τρίτων κατασκευαστών, όπως Microsoft Virtual Earth (Microsoft Virtual Earth, 2008), Google Maps (Google Maps, 2008), Google Earth (Google Earth, 2008) κ.λπ.

Βάσεις Δεδομένων Κινούμενων Αντικειμένων

Η διαχείριση δεδομένων που παράγονται από κινούμενα αντικείμενα απασχολεί την ερευνητική κοινότητα των Χωρικών Βάσεων Δεδομένων (ΧΒΔ). Εφαρμογές εντοπισμού (positioning) εμφανίζονται τα τελευταία χρόνια στην αγορά. Η κινητή τηλεφωνία και τα συστήματα παρακολούθησης εμπορικών στόλων ήδη χρησιμοποιούν τεχνολογία παρακολούθησης κινούμενων αντικειμένων. Πλέον, οι συσκευές εντοπισμού έχουν πολύ μικρό μέγεθος και μπορούν να εγκατασταθούν σε διαφορών ειδών κινητό εξοπλισμό (portable equipment). Όλες αυτές οι συσκευές παράγουν πολύ μεγάλο όγκο δεδομένων, αποτελούμενο από χρονο-σημασμένα στίγματα (time-stamped positions). Η διαδικασία αυτή εγείρει θέματα μετάδοσης (transmission), αποθήκευσης (storage), υπολογισμού (computation) και οπτικοποίησης (visualization).

Χωρο-χρονικά δεδομένα

Ως χωρο-χρονικά δεδομένα ορίζουμε μια κλάση δεδομένων που παράγεται από τη συστηματική περιγραφή των χωρο-χρονικών φαινομένων. Χωρο-χρονικό φαινόμενο είναι ένα φαινόμενο κατά το οποίο η θέση ή/και η έκταση ενός χωρικού αντικειμένου, που μπορεί να είναι σημείο ή πολύγωνο, μεταβάλλονται με τον χρόνο.

Η υπάρχουσα τεχνολογία παρέχει τη δυνατότητα τέτοιων λειτουργιών. Ο εντοπισμός της θέσης ενός αντικειμένου με μεγάλη ακρίβεια είναι εφικτός με χρήση τεχνολογιών GPS (Global Positioning Systems). Η διοχέτευση των παραγόμενων δεδομένων σε κόμβους επεξεργασίας είναι εφικτός χάρη στην ανάπτυξη των ασύρματων τηλεπικοινωνιών.

Η προαναφερθείσα γεωμετρική μεταβολή στην κατάσταση των αντικειμένων μπορεί να συμβαίνει είτε με τρόπο διακριτό είτε με τρόπο συνεχή. Στην παρούσα εργασία, το ενδιαφέρον επικεντρώνεται στη συνεχή κίνηση σημειακών αντικειμένων. Η κίνηση αυτή παρακολουθείται με διακριτό τρόπο. Θεωρούμε, δηλ., ότι με κάποιο ρυθμό, όχι απαραίτητα σταθερό, ένα κινούμενο αντικείμενο (εξοπλισμένο με κατάλληλη τεχνολογία όπως GPS/General Packet Radio Service – GPRS) αποστέλλει στίγμα ανά τακτά χρονικά διαστήματα.

Κινούμενα αντικείμενα

Ο κόσμος μας είναι συνωστισμένος από κινούμενες μονάδες. Η κινητικότητα (mobility) των μονάδων δημιουργεί κίνηση (traffic). Η κίνηση αυτή δημιουργεί μοτίβα (patterns). Η ανάλυση και η κατανόηση των μοτίβων κίνησης μπορεί να οδηγήσει στην εξαγωγή σημαντικών συμπερασμάτων. Ομάδες κινούμενων αντικειμένων μπορεί να είναι ένας στόλος από φορτηγά που εκτελούν μεταφορές στην επικράτεια μιας χώρας, πεζοί σε εμπορικά κέντρα, σταθμούς τρένων ή αεροδρόμια, καρότσια σε super markets, ζώα σε βιοτόπους κ.λπ. Όλα αυτά τα κινούμενα αντικείμενα πρέπει, φυσικά, να είναι εξοπλισμένα με κάποια συσκευή εντοπισμού.

Η κίνηση των αντικειμένων ενδέχεται να υπόκειται σε περιορισμούς, στενά συνδεδεμένους με τη φύση των αντικειμένων που παρατηρούνται και του χώρου, στον οποίο κινούνται. Για παράδειγμα, η κίνηση ενός πεζού εμποδίζεται από κάποιο φυσικό εμπόδιο. Αντίθετα, η κίνηση ενός πλοίου περιορίζεται σε θαλάσσιες εκτάσεις χωρίς όμως πολλούς περιορισμούς. Η κίνηση ενός πτηνού υπόκειται σε ακόμη λιγότερους περιορισμούς κοκ. Η κίνηση σε δίκτυο έχει μεγάλο ενδιαφέρον. Για παράδειγμα, η κίνηση οχημάτων στην πόλη γίνεται στο οδικό της δίκτυο.

Τροχιές κινούμενων αντικειμένων

Η κίνηση ενός αντικειμένου στο επίπεδο μπορεί να αναπαρασταθεί με την τροχιά του σε τρισδιάστατο σύστημα αξόνων (Εικόνα 6), το οποίο συντίθεται από δύο χωρικές (x, y) και μία χρονική (t) συντεταγμένη.

Μία τροχιά μπορεί να προσεγγιστεί από μία ακολουθία από πλειάδες της μορφής:

```
<object_id, timestamp, x_coordinate, y_coordinate>
```

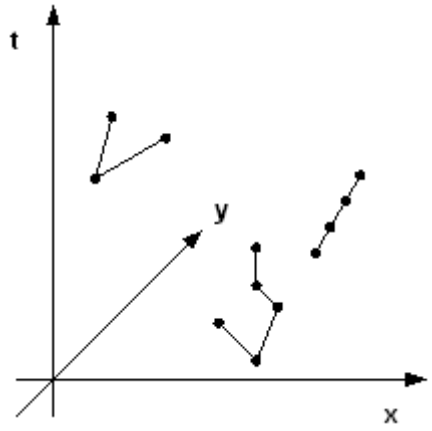
Όπου:

- object_id: η μοναδική ταυτότητα του κινούμενου αντικειμένου
- timestamp: το χρονόσημο, που περιγράφει την χρονική στιγμή, στην οποία αναφέρεται η πλειάδα
- x_coordinate: η τετμημένη και
- y_coordinate: η τεταγμένη

Οι πλειάδες αυτές είναι τα στίγματα.

Αφού οι συνεχής καταγραφής της τροχιάς είναι πρακτικά αδύνατη, η εύρεση της θέσης αντικειμένου για κάποια ενδιάμεση χρονική στιγμή μπορεί να προκύψει με παρεμβολή. Η γραμμική παρεμβολή κρίνεται επαρκής για τις περισσότερες εφαρμογές. Εναλλακτικά, πιο πολύπλοκες τεχνικές, όπως τα πολυώνυμα splines, μπορούν να χρησιμοποιηθούν για την καλύτερη εκτίμηση της θέσης του αντικειμένου σε χρονικές στιγμές για τις οποίες δεν υπάρχει πληροφορία.

Θεωρώντας γραμμική παρεμβολή, τα γνωστά σημεία θεωρούνται άκρα ευθυγράμμων τμημάτων και η τροχιά προσεγγίζεται από μία τεθλασμένη πολύ-γραμμή.



Εικόνα 6: Αναπαράσταση τροχιάς κινούμενου αντικειμένου (x, y, t)

Ερωτήματα σε κινούμενα αντικείμενα

Τα συστήματα διαχείρισης δεδομένων που παράγονται από κινούμενα αντικείμενα οφείλουν να αποκρίνονται σε διάφορα ερωτήματα. Τα ερωτήματα θέσης (location-based) αφορούν την κύρια πληροφορία που καταγράφει ένα σύστημα. Διακρίνονται στα:

- Ερωτήματα χωρο-χρονικού παραθύρου: Εφαρμόζεται ένα χωρικό και ένα χρονικό παράθυρο συγκεκριμένου εύρους και επιλέγονται τμήματα τροχιών που το τέμνουν.
- Ερωτήματα εγγύτερου γείτονα: Εύρεση k-πλησιέστερων αντικειμένων σε κάποιο άλλο αντικείμενο. Οι γείτονες είναι αυτοί που βρέθηκαν κοντά με βάση κάποια μετρική της απόστασης (Ευκλείδεια, Manhattan κ.α.) σε αντίστοιχα χρονόσημα.
- Ερωτήματα κλιμακούμενης απόστασης: Όμοια με τα ερωτήματα εγγύτερου γείτονα. Η διαφορά τους έγκειται στο γεγονός ότι δεν αναζητούνται k αντικείμενα, αλλά ο κατάλογος των λοιπών αντικειμένων με διάταξη βάσει της απόστασης.
- Ερωτήματα χρονικού τεμαχίου: αναζήτηση θέσεων αντικειμένου για συγκεκριμένο χρονικό παράθυρο.

Δεικτοδότηση κινούμενων αντικειμένων

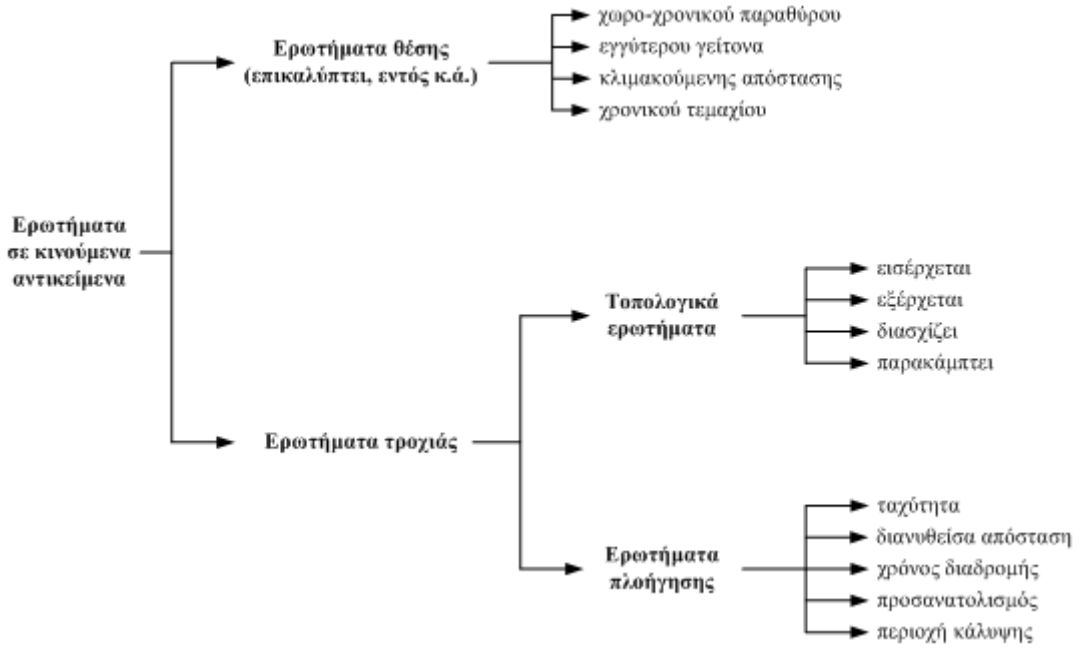
Η συγχρονισμένη εποπτεία (online) μεγάλου αριθμού κινούμενων αντικειμένων θέτει δύσκολα προβλήματα σχετικά με την αποτελεσματική προσπέλαση της πληροφορίας. Λόγω του μεγάλου όγκου της πληροφορίας, η χρήση δίσκου κρίνεται αναπόφευκτη. Συνεπώς, η επινόηση κατάλληλων δομών δεδομένων δεικτών ή ευρετηρίων (indexes) είναι αναγκαία. Οι τεχνικές που έχουν προταθεί ακολουθούν δύο κυρίως τάσεις. Η πρώτη είναι αυτή που συμπεριφέρεται στην χρονική διάσταση ως μία ακόμη διάσταση (time-oblivious). Η δεύτερη, αναφέρεται ως κινητές δομές δεδομένων (kinetic data structures) και επιχειρεί την κατασκευή ενός δυναμικού δείκτη στην κύρια μνήμη για την παρακολούθηση της κίνησης σημειακών αντικειμένων.

Η δεικτοδότηση κινούμενων αντικειμένων βασίζεται κυρίως σε υπάρχουσες τεχνικές για χωρικά αντικείμενα, με κυρίαρχο το R-tree (Guttman, 1984). Παραλλαγές, όπως το Spatio-Temporal R-tree - STR-tree (Pfoser, et al., 2000), το Trajectory-Bundle tree – TB-tree (Pfoser, et al., 2000), το Time-Parameterized tree – TPR-tree (Saltenis, et al., 2000), το R^{EXP}-tree (Saltenis, et al., 2002) και το STAR-tree (Procopiuc, et al., 2002), προσπαθούν να καλύψουν την ανεπάρκεια του R-tree στη δεικτοδότηση κινούμενων αντικειμένων.

Αρχιτεκτονική και λειτουργικότητα ΒΔ Κινούμενων Αντικειμένων

Μια ΒΔ Κινούμενων Αντικειμένων αποτελείται από στατική χωρική (γεωγραφική) και χρονική πληροφορία, μέρος της οποίας ενημερώνεται σε πραγματικό χρόνο. Η στατική πληροφορία περιλαμβάνει

χάρτες, πληροφορίες σχετικά με τα κινούμενα αντικείμενα και σχέδια κίνησης (για παράδειγμα, το όχημα ν ξεκινά από τη διεύθυνση a και κάνει παραδόσεις στις διευθύνσεις b, c και d).



Εικόνα 7: Κατηγορίες ερωτημάτων σε κινούμενα αντικείμενα

Οι ενημερώσεις πραγματικού χρόνου περιλαμβάνουν την τρέχουσα θέση και άλλη πληροφορία προερχόμενη από αισθητήρες. Αυτό είναι ένα ιδεατό μοντέλο. Στην πράξη, η πληροφορία θέσης μπορεί να μην είναι ολοκληρωμένη, δηλαδή μπορεί να είναι διαθέσιμες μόνο μερικές τροχιές. Επίσης, η ΒΔ μπορεί να είναι κατανεμημένη αντί κεντροποιημένη.

Τέλος, η κατανομή μπορεί να είναι ανάμεσα στα ίδια τα κινούμενα αντικείμενα εκτός από την ειδική περίπτωση, όπου το κάθε κινούμενο αντικείμενο αποθηκεύει τη δική του πληροφορία θέσης.

Μια άλλη γενίκευση αποτελεί η πληροφορία αισθητήρων, η οποία σχετίζεται με την πληροφορία χρόνου και θέσης. Μερικά τέτοια παραδείγματα είναι το επίπεδο των καυσίμων, οι εικόνες περιβάλλοντος, η δεσμευμένη πληροφορία από τους αισθητήρες (για παράδειγμα, η διαθεσιμότητα μίας θέσης στο γειτονικό παρκινγκ) καθώς και ένα ατύχημα σε απόσταση, το οποίο δηλώνεται από έναν αισθητήρα αερόσακων.

Μια ΒΔ Κινούμενων Αντικειμένων αποθηκεύει και διαχειρίζεται τη θέση καθώς και άλλη δυναμική πληροφορία σχετική με τα κινούμενα αντικείμενα. Από επάνω προς τα κάτω, μπορεί να περιγραφεί σαν μία αρχιτεκτονική τριών επιπέδων:

1. Μια περιβολή λογισμικού (software envelop), η οποία διαχειρίζεται τα δυναμικά στοιχεία του συστήματος.
2. Ένα Σύστημα Γεωγραφικής Πληροφορίας (GIS), το οποίο παρέχει διάφορες λειτουργίες για τη διαχείριση της γεωγραφικής πληροφορίας.
3. Ένα Σύστημα Διαχείρισης ΒΔ, το οποίο αποθηκεύει και διαχειρίζεται την πληροφορία.

Επομένως, έχουμε χρησιμοποιήσει ένα DBMS και ένα GIS για να κατασκευάσουμε μία πλατφόρμα για την ανάπτυξη εφαρμογών υπηρεσιών που είναι βασισμένες στη θέση. Η πληροφορία θέσεων και άλλες τιμές δυναμικών χαρακτηριστικών ρέουν από τα κινούμενα αντικείμενα μέχρι εκεί όπου βρίσκεται η ΒΔ Κινούμενων Αντικειμένων (γενικά, σε proxies και σε άλλα στοιχεία του ασύρματου δικτύου).

Στην προηγούμενη περιγραφή υποθέσαμε μία κεντρική αρχιτεκτονική. Ωστόσο, δεν είναι ρεαλιστικό να σκεφθούμε έναν κεντρικό proxy, όπου η ενδιαφέρουσα πληροφορία στέλνεται από όλα τα κινούμενα αντικείμενα και ο οποίος θα μπορούσε να κατακλυσθεί από την πληροφορία αυτή. Με τον τρόπο, με τον οποίο τα ασύρματα δίκτυα αποτελούνται από ένα δίκτυο στοιχείων (proxies), με τον ίδιο τρόπο και τα στοιχεία αυτά παρέχουν συνδεσιμότητα στα κινούμενα αντικείμενα που βρίσκονται στην περιοχή τους. Για το λόγο αυτό, θα μπορούσαμε να θεωρήσουμε, ότι η προσέγγιση αυτή είναι κατανεμημένη και ικανή να διαχειρισθεί ερωτήματα θέσεων, τα οποία περιλαμβάνουν κινούμενα αντικείμενα κατανεμημένα γεωγραφικά.

Επομένως, τα ερωτήματα θα πρέπει να απαντηθούν με έναν αυξητικό τρόπο (ανά περιοχή). Στη συνέχεια, θα πρέπει τα διαφορετικά αποτελέσματα να συνδυασθούν αναδρομικά, ώστε να ληφθεί ένα τελικό αποτέλεσμα που θα παρουσιασθεί στο χρήστη. Η παλιά τεχνολογία (Mokbel, et al., 2003) μπορεί να είναι μία καλή επιλογή για να υλοποιήσουμε μία παρόμοια προσέγγιση.

Οι κινητοί πράκτορες (mobile agents) θα μεταφέρουν την πληροφορία (επιμέρους ερωτήματα και τις απαντήσεις τους) στο σωστό μέρος, θα χειρίζονται τα προβλήματα διασύνδεσης και τις καθυστερήσεις του δικτύου και για αυτό θα προσαρμόζονται καλά στη δυναμική φύση των ασύρματων δικτύων. Μία ερευνητική εργασία που χρησιμοποιεί την τεχνολογία αυτή αποτελεί η αναφορά (Harri, et al., 2002). Η κατανεμημένη προσέγγιση που περιγράφεται εκεί κλιμακώνεται καλά, κάτι το οποίο είναι ένα πολύ σημαντικό χαρακτηριστικό στα δυναμικά περιβάλλοντα, όπως τα ασύρματα δίκτυα.

Εφαρμογές Βάσεων Δεδομένων Κινούμενων Αντικειμένων

Κάποιες από τις εφαρμογές που μπορούν να αναπτυχθούν στηριζόμενες στις τεχνολογίες των βάσεων δεδομένων κινούμενων αντικειμένων είναι οι εξής:

Ανακάλυψη γεωγραφικών πηγών: Ένας χρήστης κινητού τηλεφώνου παρέχει τη μελλοντική τροχιά του σε έναν πάροχο υπηρεσιών και περιμένει την απάντηση σε ερωτήματα/ενεργοποιήσεις όπως: *Ενημέρωσέ με όταν βρεθώ 2 χιλιόμετρα μακριά από ένα ξενοδοχείο που έχει διαθέσιμα δωμάτια με κόστος λιγότερο από Χ ευρώ για κάθε διανυκτέρευση*. Ο πάροχος υπηρεσιών χρησιμοποιεί μία ΒΔ κινούμενων αντικειμένων για να αποθηκεύσει την πληροφορία θέσης των πελατών του και απαντά στα ερωτήματα/ενεργοποιήσεις τους.

Μεταφορές: Τα ταξί, οι ταχυμεταφορές, οι αποκρίσεις σε επείγοντα περιστατικά, τα μέσα μεταφοράς, ο έλεγχος κυκλοφορίας, η διαχείριση αλυσίδας προμηθειών και η διαμετακομιστική αποτελούν μερικές μόνο εφαρμογές. Σε όλες αυτές τις εφαρμογές, η ΒΔ κινούμενων αντικειμένων αποθηκεύει τις τροχιές των κινούμενων αντικειμένων και απαντά σε ερωτήματα όπως: *Ποιο ταξί αναμένεται να βρεθεί εγγύτερα σε μία συγκεκριμένη διεύθυνση σε 30 λεπτά από τώρα*. *Πότε θα φθάσει το λεωφορείο σε μία συγκεκριμένη στάση*. *Πόσες φορές κατά τη διάρκεια του προηγούμενου μήνα, το λεωφορείο 25 άργησε περισσότερο από 10 λεπτά*.

Εμπόριο και πώληση θέσης: Στις εφαρμογές αυτές, κουπόνια και άλλη πληροφορία πωλήσεων, βασισμένη στη θέση, προωθείται σε μία κινητή συσκευή που προβάλει υποθετικά τις επιλογές του χρήστη.

Έλεγχος εναέριας κυκλοφορίας: Σήμερα οι τακτικές εμπορικές πτήσεις ακολουθούν συγκεκριμένους αεροδιαδρόμους, αλλά σε περίπτωση μίας ελεύθερης πτήσης, μια τυπική ενεργοποίηση προς τον πύργο εναέριου ελέγχου θα ήταν: *Βρες όλα τα ζεύγη αεροπλάνων που βρίσκονται σε πορεία σύγκρουσης ή αναμένεται να βρίσκονται μεταξύ τους σε απόσταση μικρότερη από 1 χιλιόμετρο*.

Δυναμική κατανομή εύρους ζώνης στο δίκτυα κελιών: Οι πάροχοι υπηρεσιών κελιών μπορεί να εντοπίσουν τους πελάτες τους και να αλλάξουν δυναμικά την κατανομή του εύρους ζώνης σε διαφορετικά κελιά, για να ικανοποιήσουν την διαφορετική πυκνότητα των πελατών τους.

Oracle Spatial

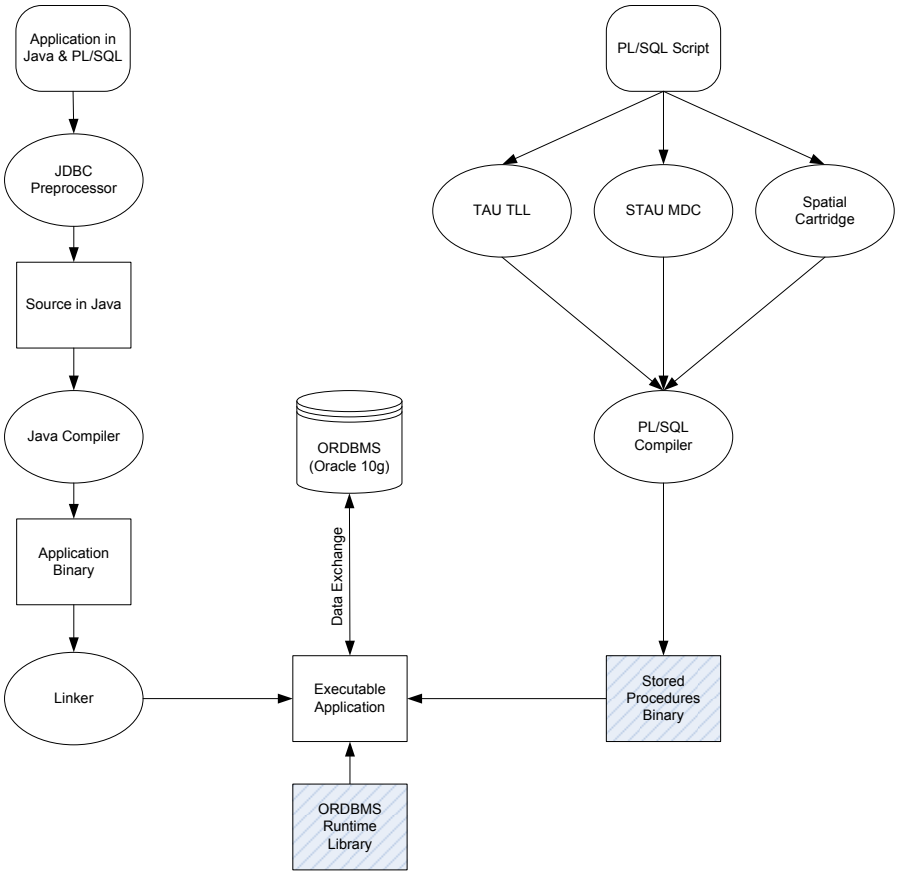
Η Oracle Database αποτελεί ένα από τα διαδεδομένα Συστήματα Διαχείρισης Βάσεων Δεδομένων (DBMS). Στην παρούσα εφαρμογή χρησιμοποιήθηκε η έκδοση Oracle Spatial 10g, η οποία υποστηρίζει χωρικές βάσεις δεδομένων, δηλαδή βάσεις δεδομένων στις οποίες είναι δυνατή επιπλέον η αποθήκευση, εισαγωγή, ενημέρωση και αποδοτικός χειρισμός (με τα χωρικά ευρετήρια) χωρικών δεδομένων και αποδοτική διεξαγωγή χωρικών επερωτήσεων σε αυτά. Παράλληλα είναι δυνατός και ο ταυτόχρονος χειρισμός μη χωρικών δεδομένων.

HERMES-Moving Data Cartridge

Το σύστημα HERMES έχει υλοποιηθεί ως επέκταση συστήματος, που παρέχει χωρο-χρονικές λειτουργίες στο Αντικειμενο-Σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (Object-Relational Database Management System – ORDBMS) της Oracle 10g. Είναι σχεδιασμένο κατά τέτοιο τρόπο ώστε να μπορεί να χρησιμοποιηθεί ως ένα αμιγώς χωρικό είτε χρονικό σύστημα, αλλά η κύρια λειτουργικότητά του έγκειται στην υποστήριξη της μοντελοποίησης (modeling) και απόκρισης σε

ερωτήματα (querying) που αφορούν συνεχώς κινούμενα αντικείμενα. Μία τέτοια συλλογή τύπων δεδομένων και των αντίστοιχων μεθόδων τους ορίζονται, υλοποιούνται και παρέχονται ως επέκταση (data cartridge) της Oracle. Το HERMES Moving Data Cartridge (HERMES-MDC) αποτελεί τον ακρογωνιαίο λίθο στην όλη αρχιτεκτονική του συστήματος HERMES. Παρέχει όλες τις απαραίτητες λειτουργίες για τη δημιουργία κινούμενων και μεταβαλλόμενων σε μέγεθος ή/και στο πεδίο του χρόνου, γεωμετριών. Καθένα από τα κινούμενα αυτά αντικείμενα είναι εφοδιασμένο με μια ομάδα μεθόδων, οι οποίες παρέχουν στον χρήστη του συστήματος όλα τα απαραίτητα εργαλεία για την αποστολή και λήψη ερωτημάτων και την ανάλυση χωρο-χρονικών δεδομένων. Με την ενσωμάτωση της λειτουργικότητας του HERMES-MDC στην γλώσσα χειρισμού δεδομένων (Data Manipulation Language – DML) της Oracle, επιτυγχάνεται η παροχή μιας εκφραστικής (expressive) και εύκολης στην χρήση γλώσσας ερωτημάτων για κινούμενα αντικείμενα.

Για την υλοποίηση ενός τέτοιου πλαισίου (framework) υπό τη μορφή επέκτασης, το σύστημα διαθέτει εκτός από ένα σύνολο βασικών τύπων δεδομένων και στατικούς χωρικούς τύπους δεδομένων που αποτελούν μέρος του Oracle Spatial. Επιπρόσθετα, διατίθενται και χρονικοί τύποι δεδομένων ως τμήμα της αντίστοιχης χρονικής επέκτασης, που ονομάζεται TAU-TLL DC (TAU-Temporal Literal Library Data Cartridge) (Oracle Corp., 2003). Με βάση αυτούς τους χωρικούς και χρονικούς τύπους δεδομένων το HERMES-MDC ορίζει ένα σύνολο τύπων δεδομένων που αναπαριστούν κινούμενα αντικείμενα. Η αρχιτεκτονική του συστήματος αποτυπώνεται παρακάτω (Εικόνα 8):



Εικόνα 8: Η αρχιτεκτονική του συστήματος HERMES (Pelekis, et al., 2006)

Η eXtensible Markup Language (XML)

Η eXtensible Markup Language (XML), αποτελεί ένα πλαίσιο για τον ορισμό γλωσσών σήμανσης (markup languages). Σε αντίθεση με την HyperText Markup Language – HTML (Raggett, et al., 1999), δεν υπάρχει μια καθορισμένη συλλογή από ετικέτες σήμανσης (markup tags) στην XML. Αντίθετα, η XML μας επιτρέπει να ορίσουμε τις δικές μας ετικέτες., σχεδιασμένες με τρόπο τέτοιο ώστε να καλύπτουν το είδος των πληροφοριών το οποίο θέλουμε να αναπαραστήσουμε. Κάθε γλώσσα XML είναι επικεντρωμένη σε ένα συγκεκριμένο πεδίο εφαρμογών, αλλά όλες μοιράζονται κοινά χαρακτηριστικά·

έχουν όλες την ίδια βασική σύνταξη και επωφελούνται την ύπαρξη ενός κοινού συνόλου εργαλείων για την επεξεργασία αρχείων.

Ο όρος eXtensible Markup Language δεν αποδίδει με σαφή τρόπο τη σημασία του όρου. Η XML δεν είναι μια μοναδική γλώσσα σήμανσης, η οποία μπορεί να επεκταθεί και για άλλες χρήσεις, αλλά κυρίως αποτελεί μια κοινή βάση πάνω στην οποία μπορούν να αναπτυχθούν markup γλώσσες. Η XML δεν αποτελεί επέκταση της HTML, ούτε αποτελεί έναν αντικαταστάτη αυτής και η οποία θα έπρεπε να αποτελεί ένα είδος γλώσσας XML. Λόγο όμως ορισμένων συντακτικών διαφορών, η HTML δεν ανήκει στο πλαίσιο της XML.

Στα αρχεία XML, τα δεδομένα περιλαμβάνονται ως σειρά κειμένου, τα οποία περικλείονται από σημάνσεις κειμένου (text markup), οι οποίες περιγράφουν τα δεδομένα αυτά. Η βασική μονάδα δεδομένων στην XML είναι το `element`. Οι προδιαγραφές της XML καθορίζουν την ακριβή σύνταξη των σημάνσεων (markup)· πως τα `elements` διαχωρίζονται από ετικέτες (`tags`), πως είναι μια ετικέτα (`tag`), ποια είναι τα αποδεκτά ονόματα για τα `elements`, που τοποθετούνται τα γνωρίσματα (`attributes`) κ.α. (Joshi, 2007).

Η XML είναι μια *meta-markup* γλώσσα. Αυτό σημαίνει ότι δε διαθέτει ένα προκαθορισμένο αριθμό ετικετών (`tags`) και αντικειμένων (`elements`), τα οποία πρέπει να λειτουργούν για τον καθένα σε κάθε τομέα ενδιαφέροντος ανά πάσα στιγμή. Οποιαδήποτε προσπάθεια δημιουργίας ενός τέτοιου προκαθορισμένου συνόλου ετικετών είναι καταδικασμένη σε αποτυχία. Αντιθέτως, επιτρέπει τον καθορισμό από τους προγραμματιστές αυτών των ετικετών οι οποίες τούς είναι απαραίτητες. Το γράμμα X στο ακρωνύμιο XML αντιστοιχεί στη λέξη eXtensible (επεκτάσιμη) που δηλώνει ότι η γλώσσα μπορεί να επεκταθεί και να προσαρμοστεί έτσι ώστε να καλύψει διαφορετικές ανάγκες.

Μολονότι η XML είναι αρκετά ευέλικτη στα αντικείμενα που επιτρέπει να οριστούν, είναι αρκετά αυστηρή σε πολλά άλλα θέματα. Παρέχει ένα *συντακτικό* στα XML αρχεία, το οποίο καθορίζει πού μπορούν να τοποθετηθούν οι ετικέτες (`tags`), πώς πρέπει να εμφανίζονται, ποια ονόματα αντικειμένων επιτρέπονται, πώς τα γνωρίσματα προσκολλώνται στις ετικέτες κ.α. Αυτό το συντακτικό είναι αρκετά συγκεκριμένο ώστε να επιτρέπει τη δημιουργία λογισμικού ανάλυσης XML (parsers), οι οποίοι έχουν την δυνατότητα να διαβάζουν οποιοδήποτε XML αρχείο. Τα αρχεία, τα οποία ικανοποιούν το συντακτικό αυτό ονομάζονται *σωστά μορφοποιημένα* (*well-formed*). Αρχεία τα οποία δεν είναι *well-formed*, δεν επιτρέπονται, όπως ένα πρόγραμμα σε γλώσσα Java που περιέχει κάποιο συντακτικό λάθος.

Σε ένα XML αρχείο το markup περιγράφει τη δομή του. Παρέχει τη δυνατότητα να προσδιοριστεί ποια elements σχετίζονται μεταξύ τους . Σε ένα καλά σχεδιασμένο XML αρχείο, περιγράφει επίσης τη σημασιολογία του. Για παράδειγμα, το markup μπορεί να υποδεικνύει ότι κάποιο element είναι μια ημερομηνία ή ένα πρόσωπο κ.α. Από την άλλη πλευρά δεν παρουσιάζεται καμία πληροφορία για το πως εμφανίζεται στο αρχείο και αυτό γιατί η XML είναι εννοιολογική γλώσσα που παρουσιάζει τη δομή και όχι την εμφάνιση του εγγράφου.

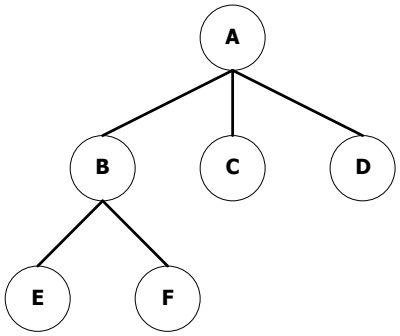
Το markup που επιτρέπεται να έχει μια συγκεκριμένη XML εφαρμογή μπορεί να τεκμηριωθεί σε ένα σχήμα (schema). Διάφορα αρχεία μπορούν να συγκριθούν με το schema και από αυτά, όποια ταιριάζουν με αυτό θεωρούνται *έγκυρα* (*valid*), ενώ όσα δεν ταιριάζουν με το σχήμα θεωρούνται *μη-έγκυρα* (*invalid*). Η εγκυρότητα ενός XML αρχείου εξαρτάται από το schema, δηλ. το εάν ένα αρχείο είναι έγκυρο ή όχι εξαρτάται από το schema με το οποίο συγκρίνεται. Δεν είναι απαραίτητο όλα τα αρχεία να είναι έγκυρα. Για πολλούς λόγους αρκεί να είναι σωστά μορφοποιημένο (well-formed).

Υπάρχουν πολλές γλώσσες προσδιορισμού σχήματος (schema languages), με διαφορετικό επίπεδο έκφρασης. Η περισσότερο διαδεδομένες και με μεγαλύτερη υποστήριξη γλώσσες, είναι η Document Type Definition (DTD) και η XML Schema. Αυτές καταγράφουν όλα τα έγκυρα markup και καθορίζουν σε ποιο σημείο και με ποιο τρόπο μπορούν να περιληφθούν τα διάφορα elements σε ένα έγγραφο (Raggett, et al., 1999) και (Thompson, et al., 2004).

XML δένδρα

Ένα XML αρχείο είναι μια ιεραρχική δομή η οποία ονομάζεται XML Tree, το οποίο αποτελείται από κόμβους διαφόρων τύπων, οι οποίοι έχουν τοποθετηθεί σε μια δενδροειδή μορφή. Δεν υπάρχει συναίνεση στην ορολογία που χρησιμοποιείται για την περιγραφή αυτών των δένδρων και για το λόγο αυτό στη συνέχεια της περιγραφής των δένδρων θα χρησιμοποιηθεί το XPath μοντέλο δεδομένων (Clark, et al., 1999).

Η Εικόνα 9 παρουσιάζει ένα παράδειγμα ενός δένδρου. Οι κόμβοι (nodes) έχουν σχεδιαστεί ως κύκλοι. Ο κόμβος ο οποίος βρίσκεται πιο ψηλά από όλους ονομάζεται ρίζα (root) του δένδρου. Οι ακμές δείχνουν την σχέση γονέα-παιδιού μεταξύ των κόμβων, για παράδειγμα ο κόμβος B είναι παιδί του κόμβου A και ο A είναι γονέας του κόμβου B. Το περιεχόμενο (content) κάθε κόμβου είναι η ακολουθία των κόμβων παιδιών του. Για τον κόμβο A, το περιεχόμενο του είναι η ακολουθία κόμβων (B, C, D). Οι κόμβοι σε ένα δένδρο είναι δυνατόν να έχουν διαφορετικό αριθμό παιδιών. Οι κόμβοι δίχως παιδιά ονομάζονται φύλλα (leaves), στο παράδειγμά μας οι κόμβοι E, F, C και D είναι κόμβοι φύλλα.



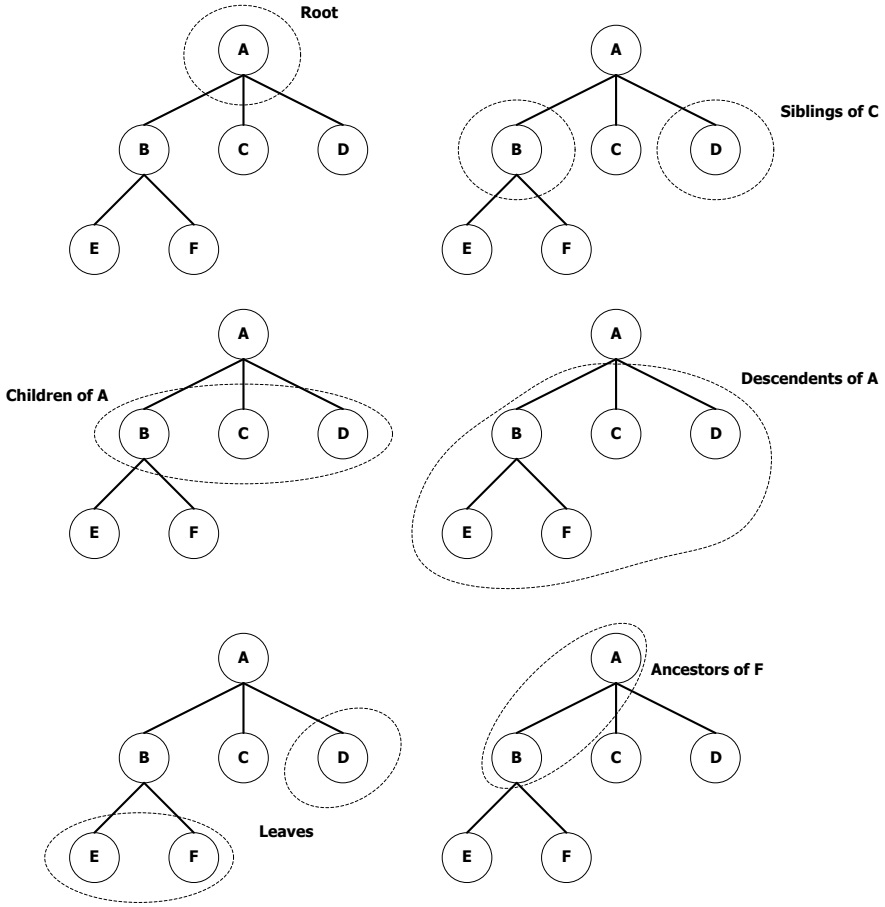
Εικόνα 9: XML δένδρο

Ένα XML δέντρο είναι διατεταγμένο (ordered), γεγονός που σημαίνει ότι η διάταξη των παιδιών ενός κόμβου είναι σημαντική. Οι αδελφοί κόμβοι (siblings) ενός κόμβου είναι τα υπόλοιπα παιδιά του κόμβου γονέα. Οι πρόγονοι (ancestors) ενός κόμβου αποτελούνται από τον γονέα αυτού, τον γονέα του γονέα κοκ, περιλαμβανομένου της ρίζας. Για παράδειγμα οι πρόγονοι του κόμβου F είναι οι κόμβοι B και A. Οι επίγονοι κόμβοι (decedents) ενός κόμβου είναι το σύνολο που αποτελείται από τα παιδιά του, τα παιδιά των παιδιών του κοκ. Για παράδειγμα οι επίγονοι του κόμβου ρίζα είναι το σύνολο όλων των κόμβων του δέντρο, εκτός από τον ίδιο τον κόμβο ρίζα. Οι παραπάνω έννοιες παρουσιάζονται στην Εικόνα 10.

Σε ένα XML Tree οι κόμβοι μπορούν να είναι ένα από τα παρακάτω είδη (Evjenet, 2007):

- Text nodes: Ένας κόμβος κειμένου (text node) αντιστοιχεί σε ένα τμήμα της συνολικής πραγματικής πληροφορίας, η οποία αναπαρίσταται από το XML αρχείο. Κάθε text node περιέχει μια μη κενή στοιχειο-σειρά, η οποία περιέχει την πληροφορία αυτή. Οι text nodes δεν έχουν κόμβους παιδιά, γεγονός που σημαίνει ότι είναι φύλλα στο XML δέντρο. Επίσης, δύο text nodes δεν μπορούν να είναι αδελφοί κόμβοι, εκτός εάν κάποιος άλλος κόμβος παρεμβάλλεται μεταξύ τους.
- Element nodes: Ένας κόμβος αντικειμένου (element node) ορίζει μια λογική ομαδοποίηση της πληροφορίας η οποία αναπαρίσταται από τους απογόνους. Κάθε element node έχει ένα όνομα (name), μια λέξη δηλαδή η οποία περιγράφει την ομαδοποίηση που πραγματοποιείται. Συνήθως χρησιμοποιείται ο όρος element για χάριν συντόμευσης. Αντίστοιχες συντομεύσεις χρησιμοποιούνται και για τους άλλους κόμβους.
- Attribute nodes: Ένας κόμβος ιδιότητας (attribute node) σχετίζεται με έναν element node, δηλ. ο κόμβος γονέας του είναι πάντοτε ένα αντικείμενο (element). Οι ιδιότητες τυπικά ενεργούν ως επεξήγηση του ονόματος του αντικειμένου, περιγράφοντας με μεγαλύτερη πληρότητα, περισσότερες ιδιότητες της ομαδοποίησης, την οποία καθορίζει ο κόμβος αντικειμένου. Μία ιδιότητα (attribute) είναι ένα ζεύγος τιμών ονόματος – τιμής, όπου το όνομα είναι μια λέξη, η οποία περιγράφει την ιδιότητα και η τιμή είναι μια στοιχειο-σειρά η οποία περιγράφει την τιμή της ιδιότητας. Κάθε αντικείμενο μπορεί να έχει το πολύ μια ιδιότητα με συγκεκριμένο όνομα, αλλά συνολικά πολλές με διαφορετικό όνομα.
- Comment nodes: Ένας κόμβος σχολίου (comment node) είναι ένας ειδικός κόμβος φύλλο (leaf node), ο οποίος έχει ως ετικέτα κάποιο κείμενο. Μπορεί να παρομοιαστεί ως τα σχόλια στον κώδικα, ο οποίος είναι γραμμένος σε μια γλώσσα προγραμματισμού.
- Processing instruction nodes: Κάθε κόμβος επεξήγησης εκτέλεσης (processing instruction node) περιλαμβάνει ένα στόχο (target) και μια τιμή (value). Χρησιμοποιείται για την μεταβίβαση πληροφορίας σε διάφορα εργαλεία επεξεργασίας αρχείων XML. Ο στόχος (target) είναι μια λέξη, η οποία καθορίζει το εργαλείο επεξεργασίας στο οποίο απευθύνεται η επεξήγηση εκτέλεσης (processing instruction). Όλα τα άλλα εργαλεία πρέπει να την αγνοήσουν. Η τιμή (value) περιλαμβάνει την σχετική πληροφορία που απευθύνεται στο εργαλείο.

- Root nodes: Κάθε XML δέντρο ξεκινά από ένα και μοναδικό κόμβο ρίζα (root node). Οι απόγονοι (children) του root node αποτελούνται από οποιοδήποτε αριθμό comment και processing instruction nodes μαζί με ένα ακριβώς element node, ο οποίος ονομάζεται root element. Ένα συνηθισμένο λάθος που γίνεται είναι η σύγχυση μεταξύ του root node και του root element.



Εικόνα 10: Ορολογία των XML δένδρων

Η αποτύπωση των αρχείων XML

Ένα αρχείο XML γράφεται ως αρχείο Unicode με ετικέτες markup και με άλλες πληροφορίες, οι οποίες αναπαριστούν τα αντικείμενα, τις ιδιότητες και τους άλλους κόμβους. Οι κόμβοι κειμένου (text nodes) γράφονται σύμφωνα με το κείμενο που αναπαριστούν. Αυτό το κείμενο ονομάζεται character data. Οι κόμβοι αντικειμένου (element nodes) δηλώνονται από ετικέτες markup. Για παράδειγμα:

```
<related ref="42">...</related>
```

Η ετικέτα `<related ref=”42”>` είναι μια ετικέτα αρχής σε ένα κόμβο αντικειμένου, ενώ η ετικέτα `</related>` είναι η αντίστοιχη ετικέτα τέλους. Το κείμενο στο ενδιάμεσο των ετικετών αποτελεί το περιεχόμενο του αντικειμένου. Οι ιδιότητες (attributes) γράφονται μέσα στην ετικέτα αρχής του αντικειμένου. Στην περίπτωση του παραδείγματος η μοναδική ιδιότητα ονομάζεται `ref` με τιμή `42`. Μέσα στην ετικέτα αρχής, η διάταξη των ιδιοτήτων δεν έχει χρηστική σημασία. Οι τιμές των ιδιοτήτων περικλείονται από ' ή ". Αντικείμενα χωρίς περιεχόμενα ονομάζονται κενά (empty) και αυτά μπορούν να αναπαρασταθούν με σύντομο τρόπο ως `<.....>`.

Ένα XML αρχείο κατά τον τρόπο αναπαράστασης του πρέπει να είναι και σωστά μορφοποιημένο (well-formed). Αυτό ουσιαστικά σημαίνει ότι πρέπει να ορίζει μια δενδρική δομή, η οποία να είναι αντίστοιχη με το εννοιολογικό μοντέλο. Για να είναι ένα XML αρχείο σωστά μορφοποιημένο οι ετικέτες αρχής και τέλους πρέπει να ταιριάζουν και να υπάρχει σωστή ενσωμάτωση των διαφόρων αντικειμένων. Για παράδειγμα τα παρακάτω δύο τμήματα κώδικα XML δεν είναι σωστά μορφοποιημένα:

```
<something>...</somethingelse> (1)
<something>...<somethingelse>...</something>...</somethingelse> (2)
```

Στο πρώτο τμήμα (1), οι ετικέτες αρχής και τέλους δεν ταιριάζουν, ενώ στο δεύτερο (2) οι ετικέτες δεν ενσωματώνονται σωστά.

Η XML κάνει διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων (case sensitive). Για παράδειγμα το τμήμα κώδικα:

```
<related ref="42">...</RelATED>
```

δεν είναι σωστά μορφοποιημένο από τη στιγμή που οι ετικέτες αρχής και τέλους έχουν διαφορετικό είδος χαρακτήρων. Κάθε σωστά μορφοποιημένο αρχείο XML πρέπει να έχει μόνο ένα κόμβο ρίζα.

Ένα αρχείο XML συνήθως ξεκινά με μια XML δήλωση (XML declaration):

```
<?xml version="1.0" encoding="UTF-8"?>
```

η οποία ακολουθείται από τον κόμβο ρίζα και τους υπόλοιπους κόμβους του αρχείου. Το τμήμα `version` δηλώνει την έκδοση της XML που χρησιμοποιείται και στην προκειμένη περίπτωση είναι `1.0`. Το τμήμα `encoding` δηλώνει με ποια κωδικοποίηση έχει γραφεί το αρχείο.

Τα σχόλια γράφονται με την μορφή:

```
<!-- ... -->
```

Οι επεξηγήσεις εκτέλεσης (processing instructions) γράφονται όπως στο παράδειγμα :

```
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"?>
```

όπου το `xml-stylesheet` είναι ο στόχος και το τμήμα `type="text/xsl" href="mystyle.xsl"` αποτελεί την τιμή.

Από τη στιγμή που ένα αρχείο XML είναι απλά ένα αρχείο κειμένου, οποιοσδήποτε κειμενογράφος μπορεί να χρησιμοποιηθεί για την συγγραφή του, σε αντίθεση με αρχεία άλλων τύπων, τα οποία απαιτούν συγκεκριμένους κειμενογράφους για την επεξεργασία τους.

Ένας XML parser είναι ένα εργαλείο, το οποίο έχει τη δυνατότητα της δενδρικής αναπαράστασης ενός αρχείου XML, από την λεκτική απεικόνισή του.

Η χρήση της XML διευκολύνει την ανάπτυξη λογισμικού αλλά την επέκταση της διαλειτουργικότητας. Κάθε έγκυρο XML έγγραφο πρέπει να συμβαδίζει με το DTD ή το Schema που περιγράφει τη δομή του. Η XML διαχωρίζει τα δεδομένα από τον τρόπο παρουσίασης. Αυτή την στιγμή υπάρχει ένας μεγάλος αριθμός markup γλωσσών που έχουν αναπτυχθεί για συγκεκριμένους σκοπούς και παράλληλα βασίζονται στο XML πλαίσιο.

Η Geography Markup Language (GML)

Η Geography Markup Language – GML (OGC, 2004) αποτελεί μια XML μορφοποίηση για την αναπαράσταση οντοτήτων του πραγματικού κόσμου, όπως δέντρα, κτήρια, δρόμους κ.ά. Οι οντότητες αναπαρίστανται ως *γνωρίσματα* (*features*) και μπορούν να περιγράψουν τόσο γεωμετρικές όσο και μη γεωμετρικές ιδιότητες. Για παράδειγμα, ένα κτήριο μπορεί να φέρει γνωρίσματα, τα οποία αναπαριστούν τη γεωγραφική τοποθεσία (γεωμετρική ιδιότητα) αλλά και τον τύπο του κτηρίου αυτού (μη γεωμετρική ιδιότητα). Η GML έχει σχεδιαστεί έτσι ώστε να είναι δυνατή η υποστήριξη της κωδικοποίησης και των δύο τύπων ιδιοτήτων, όπου οι μη γεωμετρικές εξ αυτών, μπορούν να συσχετιστούν μέσα από την ενοποίηση (integration) με άλλα XML σχήματα (schemas). Παρακάτω, παρουσιάζεται η GML αναπαράσταση ενός κτηρίου. Η γεωγραφική τοποθεσία αναπαριστάται ως μια γεωμετρική ιδιότητα με όνομα *point*, ενώ ο τύπος, η κατάσταση, καθώς επίσης και άλλες ιδιότητες, αναπαρίστανται ως μη γεωμετρικές.

```
<gml:featureMember>
    <topol:build fid="road.2545">
        <topol:geom>
            <gml:Point
                srsName=http://www.opengis.net/gml/srs/epsg.xml#32633
            >
                <gml:coordinates
                    xmlns:gml="http://www.opengis.net/gml"
                    decimal="."
                    cs=","
                    ts=" "
```

```
                    >
                        357080,7766653
                    </gml:coordinates>
                </gml:Point>
            </topol:geom>
            <topol:type>Outhouse</topol:type>
            <topol:status>2</topol:status>
            <topol:started>10101</topol:started>
            <topol:updated>19940210</topol:updated>
        </topol:build>
    </gml:featureMember>
```

Δομή

Η δομή ενός GML εγγράφου είναι ιδιαιτέρως ευέλικτη. Γενικά, αποτελεί από μια σειρά από *Ιδιότητες Features* που αναπαριστούν οντότητες του πραγματικού κόσμου. Οι ιδιότητες αυτές αποτελούν απογόνους ενός FeatureCollection, το οποίο λειτουργεί ως ομαδοποίηση (container) των παραπάνω. Μια από τις αιτίες που παρέχουν στη GML μορφοποίηση αυτή την ευελιξία αποτελεί το γεγονός ότι κάθε Feature είναι με τη σειρά του FeatureCollection. Με τον τρόπο αυτό, μια οντότητα μπορεί να αναπαρασταθεί από συναθροίσεις άλλων ιδιοτήτων. Για παράδειγμα, ένα πάρκο με δέντρα, πράσινες περιοχές, νερό και δρόμους. Ενώ καθεμιά από αυτές τις οντότητες μπορεί να αναπαρασταθεί ανεξάρτητα από μια ιδιότητα, το πάρκο δύναται να οριστεί ως μια Feature ή FeatureCollection που αποτελείται από όλα τα παραπάνω (τα δέντρα, τους δρόμους κοκ). Επιπλέον, η GML υποστηρίζει τον ορισμό άλλων συσχετίσεων μεταξύ διαφορετικών ιδιοτήτων με την χρήση της XLink (DeRose, et al., 2001).

Υποστηριζόμενες γεωμετρικές ιδιότητες

Το GML πρότυπο παρέχει ένα μηχανισμό για την κωδικοποίηση των αποκαλούμενων από το Open Geospatial Consortium – OGC (OGC, 2008), απλών ιδιοτήτων (simple features). Με τον όρο απλή, το OGC αναφέρεται σε «… ιδιότητες, των οποίων τα γεωμετρικά χαρακτηριστικά περιορίζονται σε *απλές* γεωμετρίες, για τις οποίες ορίζονται συντεταγμένες σε δύο ή/και τρεις διαστάσεις και η απεικόνιση μιας καμπύλης αποτελεί θέμα γραμμικής παρεμβολής» (Lake, 2001). Με άλλα λόγια, αυτό σημαίνει ότι το GML πρότυπο στοχεύει κυρίως στην αναπαράσταση γεωμετρικών ιδιοτήτων σε δύο ή τρεις διαστάσεις. Ακολουθεί μια λίστα με τις απλές γεωμετρικές κλάσεις, όπως αυτές ορίζονται από το OGC:

- Point
- LineString
- LinearRing
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- MultiGeometry

Παρακάτω, παρουσιάζονται κάποια παραδείγματα, επεξηγώντας τις τέσσερις πρώτες ιδιότητες. Οι ιδιότητες που φέρουν το πρόθεμα *Multi*, αποτελούν απλώς συλλογές ιδιοτήτων, οι οποίες απαρτίζονται από μια ή/και περισσότερες βασικές γεωμετρικές ιδιότητες.

```
<gml:Point
    srsName="http://www.opengis.net/gml/srs/epsg.xml#32633">
    <gml:coordinates
        xmlns:gml="http://www.opengis.net/gml"
        decimal="."
        cs=","
        ts=" ">
            357080,7766653
    </gml:coordinates>
</gml:Point>
```

```
<gml:LineString>
    <gml:coordinates
        xmlns:gml="http://www.opengis.net/gml"
        decimal="."
        cs=","
        ts=" ">
            357015,7766698 357127,7766654
            357389,7766583 357406,7766595
    </gml:coordinates>
</gml:LineString>

<gml:Polygon>
    <gml:outerBoundaryIs>
        <gml:LinearRing>
            <gml:coordinates decimal="." cs="," ts=" ">
                80,340 160,340 160,280 80,280 80,340
            </gml:coordinates>
        </gml:LinearRing>
    </gml:outerBoundaryIs>
    <gml:innerBoundaryIs>
        <gml:LinearRing>
            <gml:coordinates decimal="." cs="," ts=" ">
                100,330 130,330 130,290 100,330
                90,290 130,290 130,290 100,330
            </gml:coordinates>
        </gml:LinearRing>
        <gml:LinearRing>
            <gml:coordinates decimal="." cs="," ts=" ">
                150,335 150,320 140,335 150,335
            </gml:coordinates>
        </gml:LinearRing>
    </gml:innerBoundaryIs>
</gml:Polygon>
```

GML 3.0

Με βάση την τρίτη έκδοση του GML προτύπου (Cox, et al., 2004), υπάρχει πλήρης συμβατότητα με την παλαιότερη έκδοσή του, τη 2.1.2. Η κύρια διαφορά μεταξύ των δύο αυτών εκδόσεων, έγκειται στο γεγονός της επέκτασης του συνόλου των παρεχόμενων ιδιοτήτων με την υποστήριξη περισσότερων γεωμετριών, δυναμικών ιδιοτήτων, τρισδιάστατων αντικειμένων, εξ' ορισμού μορφοποίηση (default styling) κ.ά. Η παρούσα εργασία κάνει χρήση του πρόσφατου GML 3.1.1 προτύπου, λόγω της υποστήριξης ιδιοτήτων που αφορούν σε κινούμενα αντικείμενα, ιδιότητες δηλ., οι οποίες μεταβάλλουν τη γεωγραφική τους θέση μέσα στο πεδίο του χρόνου. Κρίθηκε λοιπόν αναγκαία η χρήση χαρακτηριστικών (πχ χρόνος), τα οποία έχουν εισαχθεί στην τρίτη έκδοση του προτύπου.

```
<gml:featureCollection xmlns:gml="http://www.opengis.net/gml">
    <gml:featureMember>
        <gml:TimePeriod>
            <gml:begin>2001/02/09T16:32.53</gml:begin>
            <gml:end>2001/02/09T16:33.23</gml:end>
        </gml:TimePeriod>
        <gml:LineString>
            <gml:posList>
                37.9923843504678 23.7769293591931
                37.9934031121064 23.7766176385097
            </gml:posList>
        </gml:LineString>
    </gml:featureMember>
</gml:featureCollection>
```

H Keyhole Markup Language (KML)

Η Keyhole Markup Language – KML (Google Inc., 2008) αποτελεί ένα πρότυπο αναπαράστασης γεωγραφικών δεδομένων, το οποίο βασίζεται στην XML (Bray, et al., 2006) για γραφικά δύο ή/και τριών διαστάσεων.

Τα γεωγραφικά δεδομένα παρέχουν πληροφορίες για διάφορες οντότητες που υπάρχουν στην επιφάνεια της γης και είναι αποτέλεσμα είτε του ανθρώπινου είτε φυσικού παράγοντα, όπως οδικά δίκτυα, κτήρια, σύνορα, αλλά και ποταμοί, λίμνες κ.λπ. Στη διαδικασία της οπτικοποίησης οι προαναφερθείσες οντότητες μπορούν να αναπαρασταθούν γεωγραφικά. Για τη δημιουργία οπτικών αναπαραστάσεων, ανεξάρτητα από το μέσο παρουσίασης, μπορούν να χρησιμοποιηθούν βασικά γραφικά στοιχεία όπως σημεία (points), γραμμές (linestrings) και πολύγωνα (polygons). Το στοιχεία σημείου φέρουν την έννοια της τοποθεσίας και αποτελούν τα πλέον βασικά από τις τρεις ανωτέρω κατηγορίες. Τα στοιχεία γραμμής αποτελούνται από πλειάδες στοιχείων σημείου και παρουσιάζουν τόσο έννοιες κατεύθυνσης όσο και τοποθεσίας. Τέλος, τα στοιχεία πολυγώνου αναπαριστούν έννοιες που σχετίζονται με έκταση, κατεύθυνση και τοποθεσία, αποτελούμενα από δυσδιάστατους πίνακες στοιχείων σημείου.

Με απώτερο στόχο την αύξηση της χρηστικότητας ενός οπτικοποιημένου αποτελέσματος, η αναγνώριση και ανάλυση των προς παρουσίαση δεδομένων, κρίνεται απαραίτητη. Η επιπλέον πληροφορία που παράγεται και συνοδεύει το αποτέλεσμα αυτό, αναπαριστάται υπό τη μορφή κειμένου.

Δομή

Τα εργαλεία Google Maps (Google Maps, 2008) και Google Earth (Google Earth, 2008), χρησιμοποιούν το KML ως εξωτερικό πρότυπο χωρο-χρονικών δεδομένων. Με την χρήση του KML, ο χρήστης μπορεί να εισάγει γεωγραφικά δεδομένα σε κάποιο από τα παραπάνω εργαλεία και να κάνει υπέρθεση των δεδομένων αυτών επάνω στα δεδομένα που παρέχονται από την Google.

Πέραν της δυνατότητας εισαγωγής απλών σημείων σε ένα χάρτη της Google, το KML, διαθέτει δυνατότητες χειρισμού των εξής δυσδιάστατων γεωμετρικών σχημάτων:

- Σημεία – Points: μπορούν να αναπαρασταθούν οπτικά στην επιφάνεια της γης είτε ως εικονίδια (icons) είτε ως ετικέτες (labels) ή και τα δύο και να τοποθετηθούν σε διαφορετικά υψόμετρα.
- Γραμμές (Lines): μπορούν να τοποθετηθούν σε διάφορα υψόμετρα όπως και τα σημεία. Υποστηρίζονται επίσης οι *πολύ-γραμμές*, σύνθετα δηλ. στοιχεία που αποτελούνται από ένα σύνολο στοιχείων γραμμής.
- Πολύγωνα (Polygons): μπορούν να δημιουργηθούν σε δύο ή/και τρεις διαστάσεις έχοντας σταθερή μορφή ή φέροντας εσωτερικά όρια (inner boundaries). Ως εκ τούτου, μπορούν να δημιουργηθούν σύνθετα τρισδιάστατα αντικείμενα σε διάφορα υψόμετρα.

Ο χειρισμός της εμφάνισης αυτών των γεωμετρικών σχημάτων, μπορεί να πραγματοποιηθεί με της εξής τεχνικές: ορίζοντας συντεταγμένες, δημιουργώντας προεξοχές (extruding) για τρισδιάστατα σχήματα και ομαδοποιώντας συλλογές στοιχείων. Ο ορισμός συντεταγμένων επιτυγχάνεται με τη δήλωση των αντίστοιχων συντεταγμένων και το ύψος (elevation) από την επιφάνεια της θάλασσας. Οι δημιουργία προεξοχών σχετίζεται με την τοποθέτηση ενός στοιχείου σε μια συγκεκριμένη θέση και χρησιμοποιώντας την ετικέτα `<extrude>`. Η ομαδοποίηση ολοκληρώνεται με την χρήση της `<MultiGeometry>` ετικέτας και χρησιμοποιείται για οργανωτικούς σκοπούς.

Αναφορικά με την έννοια του χρόνου, αυτή αναπαριστάται στο KML πρότυπο με την χρήση της `<TimeSpan>` ετικέτας, η οποία αποτελεί ένα σύνθετο τύπο δεδομένων, σε όρους XML. Ο εν λόγω τύπος, αποτελείται από δύο απλούς τύπους δεδομένων `<begin>` και `<end>`, οι οποίοι αναπαριστούν την έναρξη και τη λήξη της κίνησης ενός αντικειμένου υπό τη μορφή χρονόσημων (timestamps).

Ένα πλήρες KML αρχείο, το οποίο αναπαριστά τα στοιχεία που προαναφέρθηκαν, παρουσιάζεται παρακάτω. Πιο συγκεκριμένα, παρακάτω αναπαριστάται η κίνηση ενός αντικειμένου μέσα σε μια χρονική περίοδο.

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
    <Document>
        <Folder>
            <name>VisualHermes</name>
            <Style id="blueLine">
```

```
                    <LineStyle>
                        <color>ffff0000</color>
                        <width>3</width>
                    </LineStyle>
                </Style>
                <Placemark>
                    <name>Trajectory ID: 6 - Object ID: 0420</name>
                    <description>Trajectory Segment: 1</description>
                    <TimeSpan>
                        <begin>2001-02-12T17:45:36Z</begin>
                        <end>2001-02-12T17:45:51Z</end>
                    </TimeSpan>
                    <styleUrl>#blueLine</styleUrl>
                    <LineString>
                        <altitudeMode>relative</altitudeMode>
                        <coordinates>
                            23.76964,37.94409,0
                            23.76831,37.94306,0
                        </coordinates>
                    </LineString>
                </Placemark>
            </Folder>
        </Document>
    </kml>
```

Στο εν λόγω αρχείο, αφού οριστούν τα αρχικά στοιχεία ώστε αυτό να συμμορφώνεται με το KML πρότυπο δεδομένων, δημιουργείται ένα στοιχείο παρουσίασης με την χρήση της `<Placemark>` ετικέτας. Το στοιχείο αυτό περιλαμβάνει ένα όνομα (ετικέτα `<name>`), μια περιγραφή (ετικέτα `<description>`), καθώς επίσης και ένα σύνολο από ετικέτες που καθορίζουν τα χωρο-χρονικά χαρακτηριστικά του. Οι ετικέτες αυτές είναι οι:

- `<TimeSpan>`: Ορίζει, μέσω των `<begin>` και `<end>` ετικετών τα χρονόσημα για την έναρξη και τη λήξη της κίνησης.
- `<LineString>`: Ορίζει μέσω της `<coordinates>` ετικέτας της συντεταγμένες για την αρχή και το τέλος της κίνησης.
- `<styleURL>`: Αποτελεί την ταυτότητα για τα χαρακτηριστικά της παρουσίασης του εν λόγω στοιχείου από το εργαλείο οπτικοποίησης. Τα χαρακτηριστικά αυτά έχουν ήδη οριστεί, στην αρχή του αρχείου (ετικέτα `<Style>`) και αφορούν τον χρωματισμό (ετικέτα `<color>`) και το πάχος της γραμμής (ετικέτα `<width>`).

Εργαλεία παρουσίασης

Για την οπτικοποίηση ενός KML αρχείου απαιτείται κάποιου είδους εφαρμογή. Η Google, ως ο κύριος πάροχος του KML, προσφέρει δύο διαφορετικούς τύπους εφαρμογών, τις οποίες ένας χρήστης μπορεί να χρησιμοποιήσει για την οπτικοποίηση των δεδομένων που διαθέτει. Το Google Earth αποτελεί μια επιτραπέζια εφαρμογή (desktop application) με πληθώρα λειτουργιών. Το Google Maps αποτελεί μια υπηρεσία Ιστού (Web service), με την οποία ο χρήστης λαμβάνει την οπτικοποίηση των δεδομένων του μέσω μιας εφαρμογής πλοήγησης στο Διαδίκτυο (Web browser). Ως δια-δικτυακή εφαρμογή, το Google Maps, φέρει λιγότερες δυνατότητες από το αντίστοιχο Google Earth.

Η eXtensible Stylesheet Language Transformation (XSLT)

Η XSLT (Clark, 1999) είναι μια συναρτησιακή (functional) γλώσσα προγραμματισμού, με την χρήση της οποίας καθορίζεται ο τρόπος που ένα XML έγγραφο μετατρέπεται σε μια διαφορετική μορφή. Αν και αποτελεί σύνηθες φαινόμενο, το παραγόμενο αρχείο δεν απαιτείται να είναι επίσης ένα XML έγγραφο.

Βασική ροή

Οι κανόνες μετατροπής που καθορίζουν τον τρόπο μορφοποίησης ενός εγγράφου δηλώνονται εντός των κανόνων προτύπου (template rules) και αποθηκεύονται σε ένα stylesheet. Κάθε πρότυπο (template) φέρει μια ιδιότητα αντιστοιχίας (match attribute). Η ιδιότητα αυτή περιλαμβάνει ένα πρότυπο (pattern), το οποίο αντιστοιχεί στον κόμβο πηγή (source node) ή στους κόμβους όπου πρόκειται να εφαρμοστεί ο κανόνας μορφοποίησης. Το πρότυπο αναγνωρίζεται από μια γλώσσα που ονομάζεται XPath (Clark, et al., 1999). Για την εκτέλεση της πραγματικής μετατροπής, εκτός από μια ομάδα κανόνων που είναι αποθηκευμένη σε ένα stylesheet και ένα XML έγγραφο πηγή, στο οποίο θα εφαρμοστούν οι κανόνες αυτοί, απαιτείται η ύπαρξη ενός XSLT επεξεργαστή (XSLT processor). Πρόκειται, κατ' ουσία, για συγκεκριμένο λογισμικό, το οποίο αναλύει τόσο το έγγραφο πηγή, όσο και το έγγραφο των κανόνων, δημιουργώντας δύο διαφορετικές δενδρικές δομές. Στο δένδρο πηγή εν συνεχεία, εκτελείται μια αναζήτηση κόμβων, οι οποίοι πληρούν τις προϋποθέσεις κάποιου/-ων από τους κανόνες μορφοποίησης. Τελικά, εφαρμόζονται (εκτελούνται) τα πρότυπα που βρίσκουν αντιστοιχία. Παρακάτω, παρουσιάζεται ένα παράδειγμα ενός ιδιαίτερα απλοϊκού XSLT stylesheet, το οποίο χρησιμοποιείται για την επεξεργασία του GML εγγράφου που έπεται. Το αποτέλεσμα της μετατροπής παρουσιάζεται στο τέλος.

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Καθορίζει τη έκδοση της γλώσσας, τα namespaces κ.λπ. -->
<xsl:stylesheet
 version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:msxsl="urn:schemas-microsoft-com:xslt"
 xmlns:ext="http://goutsidis.gr/extension"
 exclude-result-prefixes="gml msxsl ext"
>

<!-- Καθορίζει τον τύπο και την κωδικοποίηση  του παραγόμενου
     εγγράφου-->
<xsl:output
 method="xml"
 version="1.0"
 encoding="utf-8"
 indent="yes"
 media-type="application/vnd.google-earth.kml+xml"
/>

 <!-- Κανόνας προτύπου με μια ιδιότητα αντιστοιχίας -->
 <xsl:template match="gml:Point">
  <xsl:element name="Point"> <!-- Το element που δημιουργείται --
>
   <xsl:element name="coordinates"> <!-- Νέο element απόγονος -->
   <!-- Ορίζει στον επεξεργαστή να εξάγει την στοιχειο-σειρά του
        τρέχοντος κόμβου. Καθώς ο εν λόγω κανόνας αντιστοιχεί
στο
        "gml:Point" element, η τιμή αυτή θα είναι η στοχειο-
σειρά
        του "gml:Point" κόμβου. -->
    <xsl:value-of select="." />
   </xsl:element>
  </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```

Το βασικό stylesheet ορίζει έναν κανόνα προτύπου (template rule), ο οποίος αντιστοιχεί στον `gml:Point` κόμβο.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<gml:Point>
    23.86301,37.99958
</gml:Point>
```

```
<?xml version="1.0" encoding="utf-8"?>

<Point>
    <coordinates>
        23.86301,37.99958
    </coordinates>
</Point>
```

XSLT στοιχεία

Η XSLT ορίζει 37 στοιχεία (elements), τα οποία μπορούν να κατηγοριοποιηθούν σε τρεις επικαλυπτόμενες κατηγορίες· ρίζα (root), επίπεδο κορυφής (top level) και στοιχεία οδηγιών (instruction elements). Σε αυτή την ενότητα παρουσιάστηκε μόνο το `xsl:element`. Η XSLT χρησιμοποιείται κυρίως για τη δημιουργία νέων XML εγγράφων που βασίζονται στη συγχώνευση και τη μορφοποίηση ενός συνδυασμού άλλων πηγών. Κατά την παραγωγή νέων εγγράφων, συχνά παρουσιάζεται η ανάγκη δημιουργίας νέων XML elements. Παρά το γεγονός ότι η XSLT υποστηρίζει τη χρήση κυριολεκτικών στοιχείων αποτελέσματος (literal result elements), η χρήση του `xsl:element` μας επιτρέπει τον καθορισμό του ονόματος του element κατά την εκτέλεση (runtime).

XSLT συναρτήσεις

Ενώ οι λειτουργίες της XPath επικεντρώνονται σε κόμβους, καθώς επίσης και στις τιμές αυτών, η XSLT παρέχει επιπρόσθετες συναρτήσεις για πιο γενικευμένους σκοπούς. Παραδείγματα τέτοιων, κοινά χρησιμοποιούμενων, συναρτήσεων είναι οι *document()* και *current()* συναρτήσεις. Η *document()* συνάρτηση επιτρέπει τη φόρτωση εξωτερικών XML εγγράφων κατά τη διάρκεια της επεξεργασίας. Η *current()* συνάρτηση χρησιμοποιείται κυρίως σε επαναληπτικές διαδικασίες (loops), όπου αναπαριστά τον τρέχοντα επεξεργαζόμενο κόμβο.

Επιπλέον, είναι δυνατή και η χρήση *τρίτων* συναρτήσεων, οι οποίες δύνανται να έχουν αναπτυχθεί σε άλλη γλώσσα προγραμματισμού (πχ C#) και βρίσκονται στο ίδιο ή σε εξωτερικό XSLT έγγραφο. Η χρήση τέτοιων συναρτήσεων βέβαια, προϋποθέτει την ύπαρξη κατάλληλου XSLT επεξεργαστή, ο οποίος θα είναι σε θέση να τις εκτελέσει. Στην παρούσα εργασία, γίνεται χρήση τρίτων συναρτήσεων για τη μορφοποίηση συγκεκριμένων τιμών εντός των παραγόμενων αρχείων.

Ο VisualHERMES wrapper

Ανάλυση απαιτήσεων

Η τυποποίηση των δεδομένων που ανταλλάσσονται μεταξύ των διάφορων μερών του συστήματος, πραγματοποιείται με την χρήση του GML προτύπου, με δεδομένο το γεγονός ότι πρόκειται για διακίνηση γεωγραφικών πληροφοριών. Το GML, αποτελεί ένα XML πρότυπο μορφοποίησης δεδομένων, το οποίο επιτρέπει τη μεταφορά και αποθήκευση γεωγραφικών πληροφοριών, περιλαμβάνοντας τόσο χωρικά όσο και χρονικά στοιχεία των γεωγραφικών οντοτήτων, καθώς επίσης και χειρισμό ιδιοτήτων για κινούμενα αντικείμενα. Το GML πρότυπο έχει σχεδιαστεί έτσι ώστε να υποστηρίζει τη μέγιστη δυνατή δια-λειτουργικότητα και αυτό επιτυγχάνεται μέσω της παροχής βασικών γεωμετρικών ετικετών (όλα τα συστήματα που υποστηρίζουν το GML πρότυπο, κάνουν χρήση των ίδιων γεωμετρικών ετικετών), ενός κοινού μοντέλου δεδομένων (οντότητες και ιδιότητες) και ενός μηχανισμού για τη δημιουργία και το διαμοιρασμό σχημάτων εφαρμογών (application schemas), επιτρέποντας με τον τρόπο αυτό τη μοντελοποίηση της σημασιολογίας των χωρο-χρονικών πληροφοριών.

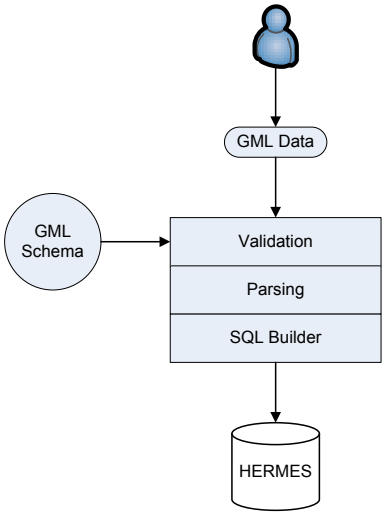
Η υποστηριζόμενη δια-λειτουργικότητα από το ενδιάμεσο επίπεδο χειρισμού των δεδομένων από και προς το σύστημα διαχείρισης της βάσης δεδομένων τροχιών, ολοκληρώνεται γύρω από την παροχή ενός σχήματος (schema), με το οποίο καλούνται να συμμορφώνονται τόσο τα εισερχόμενα, όσο και τα

εξερχόμενα δεδομένα. Σε επίπεδο ανάπτυξης, αυτό μεταφράζεται στο διαχωρισμό της λειτουργίας του ενδιάμεσου επιπέδου σε δύο μέρη:

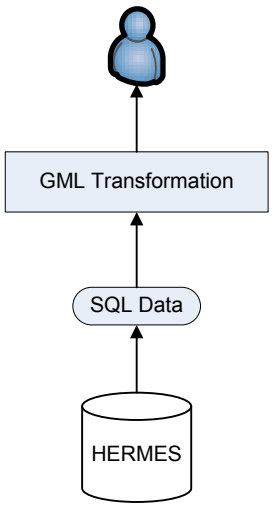
1. Τα δεδομένα που απαρτίζουν τα ερωτήματα (queries) που πρόκειται να εκτελεστούν στο σύστημα διαχείρισης πρέπει να:
   - Συμμορφώνονται με το παρεχόμενο πρότυπο (GML Schema),
   - Επικυρώνονται σε επίπεδο σημασιολογίας και
   - Μετατρέπονται σε SQL όρους

   έτσι ώστε να εκτελεστούν επιτυχώς (Εικόνα 11) και
2. Τα δεδομένα, τα οποία απαρτίζουν τα αποτελέσματα που προκύπτουν από την εκτέλεση των ερωτημάτων, πρέπει να:
   - Μορφοποιούνται βάσει του παρεχόμενου, από το ενδιάμεσο επίπεδο, σχήμα

   έτσι ώστε ο αποδέκτης τους να είναι σε θέση να εκτελέσει τις όποιες επιπλέον διαδικασίες ανάλυσης σε ένα σύνολο δεδομένων, το οποίο υπόκειται σε μια γνωστή και κοινά αποδεκτή μορφοποίηση (Εικόνα 12).



Εικόνα 11: Αποστολή ερωτήματος



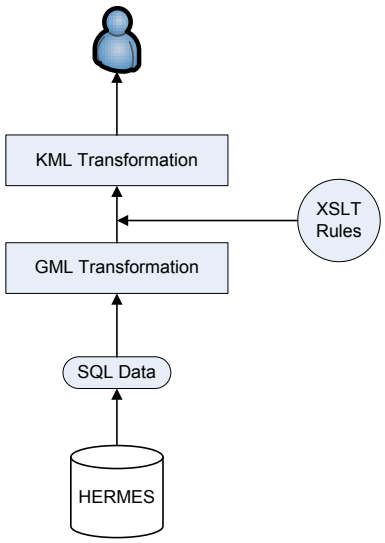
Εικόνα 12: Λήψη αποτελέσματος

Τέλος, για την υλοποίηση της διαδικασίας οπτικοποίησης των αποτελεσμάτων και την παρουσίαση αυτών με τη βοήθεια μηχανών τρίτων κατασκευαστών, όπως Google Maps/Earth, απαιτείται η εφαρμογή μετασχηματισμών των GML δεδομένων σε μία μορφή, η οποία κρίνεται αποδεκτή από τις προαναφερθείσες μηχανές. Στην παρούσα υλοποίηση, τα αποτελέσματα θα πρέπει να μετατραπούν με βάση το KML πρότυπο. Επειδή, τόσο το GML, όσο και το KML, βασίζονται στο γενικότερο πρότυπο μορφοποίησης δεδομένων XML, η απαιτούμενη μετατροπή μπορεί να πραγματοποιηθεί με την χρήση XSLT κανόνων μορφοποίησης (Εικόνα 13).

Φυσικά, όλες οι παραπάνω λειτουργίες, από πλευράς υλοποίησης, ενοποιούνται και ολοκληρώνονται κάτω από την ομπρέλα μιας δια-δικτυακής εφαρμογής, η οποία παρέχει στον χρήστη ένα λιτό και κατανοητό περιβάλλον εργασίας, εντός του οποίου μπορεί να πραγματοποιηθεί η σύνταξη και αποστολή ερωτημάτων, η λήψη GML αποτελεσμάτων και η οπτικοποίηση KML δεδομένων, μέσα από μία εφαρμογή περιηγητή διαδικτύου, αποκρύπτοντας ταυτόχρονα όλες τις εσωτερικές διαδικασίες του όλους συστήματος.

Αναφορικά με τη σύνταξη και την αποστολή των επιθυμητών ερωτημάτων του χρήστη προς το σύστημα, δίνονται τρεις (3) δυνατότητες:

1. Η σύνταξη ενός SQL ερωτήματος χειροκίνητα,
2. Η δημιουργία ενός SQL ερωτήματος, βάσει επιλογών που παρέχονται στον χρήστη, μέσω κατάλληλων πεδίων επιλογής και
3. Η μεταφόρτωση (upload) ενός XML αρχείου, το οποίο υπόκειται σε ένα προκαθορισμένο σχήμα και το οποίο περιλαμβάνει τα δεδομένα για τη συναρμολόγηση του κατάλληλου SQL ερωτήματος από το σύστημα.



Εικόνα 13: Δημιουργία KML αποτελέσματος

Τα αποτελέσματα που μπορεί να λάβει ένας χρήστης μετά την επιτυχή εκτέλεση ενός ερωτήματος, με οιαδήποτε από τις προαναφερθείσες μεθόδους, είναι δύο ειδών:

1. GML αρχείο αποτελεσμάτων: Ο χρήστης λαμβάνει μια υπερσύνδεση (hyperlink), μέσω της οποίας έχει τη δυνατότητα μεταφόρτωσης (download) του τελικού αποτελέσματος, στον τοπικό του υπολογιστή και
2. KML αρχείο αποτελεσμάτων: Ο χρήστης λαμβάνει μια υπερσύνδεση (hyperlink), μέσω της οποίας ανακατευθύνεται σε μια δεύτερη σελίδα της εφαρμογής, από όπου, σε συνδυασμό με την υπηρεσία Maps της Google, είναι σε θέση να λάβει το τελικό αποτέλεσμα του ερωτήματός του, παρουσιαζόμενο στο χάρτη.

Σχεδιασμός

Η προσπάθεια επέκτασης της δια-λειτουργικότητας του συστήματος HERMES, οδήγησε στην ανάπτυξη ενός νέου εργαλείου-εφαρμογής με το όνομα VisualHERMES. Το εν λόγω εργαλείο, μπορεί να εγκατασταθεί και να λειτουργήσει, χωρίς να απαιτείται κάποια αλλαγή στο υπάρχον σύστημα (Εικόνα 14). Με άλλα λόγια, το VisualHERMES, αποτελεί μια εναλλακτική διεπαφή του χρήστη με το υπάρχον σύστημα, προσφέροντας βέβαια δυνατότητες όπως λήψη αποτελεσμάτων μορφοποιημένων βάσει του GML προτύπου, καθώς επίσης και οπτικοποίησης των αποτελεσμάτων αυτών με την χρήση των μηχανών Maps και Earth της Google.

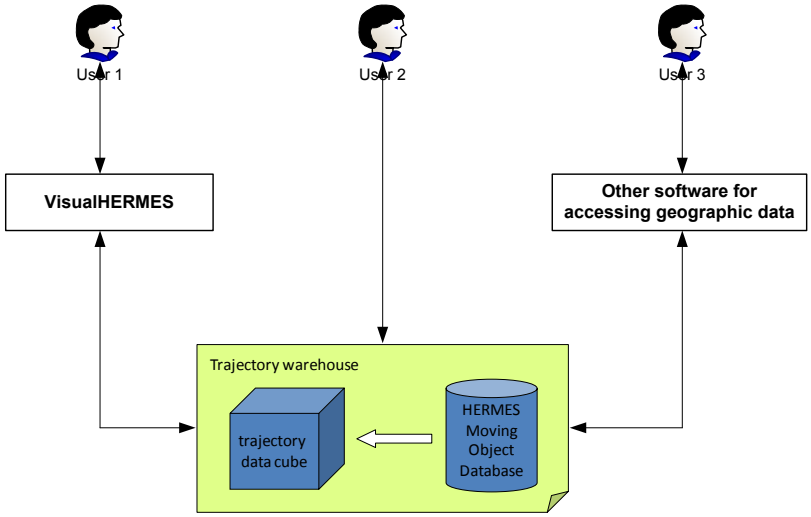
Το σύστημα VisualHERMES, βασίζεται στην αρχιτεκτονική τριών επιπέδων ανάπτυξης εφαρμογών (3-tier architecture). Η επιλογή έγινε κυρίως με γνώμονα τις δυνατότητες επέκτασης που θα μπορεί να έχει μελλοντικά το σύστημα και ταυτόχρονα, οι όποιες αλλαγές πρέπει να πραγματοποιηθούν, να είναι πλήρως στοχευόμενες σε ένα ή περισσότερα επίπεδα. Επιπλέον, ένα ανταγωνιστικό πλεονέκτημα που φέρει η συγκεκριμένη αρχιτεκτονική είναι αυτό της απομόνωσης του εκάστοτε επιπέδου, με τρόπο τέτοιο, ώστε να είναι δυνατή η συνέχιση της λειτουργίας της όλης εφαρμογής ακόμη και αν αντικατασταθεί πλήρως κάποιο ή κάποια από τα επίπεδά της (Εικόνα 15).

Η ανάπτυξη του εκάστοτε επιπέδου, με εξαίρεση αυτού της παρουσίασης, ακολουθεί τη λογική της δημιουργίας δυναμικών βιβλιοθηκών (DLL) των Windows. Η συγκεκριμένη προσέγγιση προσφέρει ένα ακόμη πλεονέκτημα στο όλο σύστημα, καθώς με τον τρόπο αυτό επιτυγχάνεται η αποσύνδεση του επιπέδου παρουσίασης, δηλ. της εφαρμογής, με την οποία έρχεται σε επαφή ο τελικός χρήστης. Έτσι λοιπόν, καθίσταται δυνατή η ανάπτυξη άλλων τύπων εφαρμογών, όπως εφαρμογές για Windows, εφαρμογές για Web, εφαρμογές για PDAs, υπηρεσίες Ιστού (Web Services) κ.λπ., οι οποίες θα παρέχουν όλη τη δια-λειτουργικότητα που προσφέρεται μέσω του εν λόγω wrapper και ταυτόχρονα θα μπορούν να στοχεύουν σε διαφορετικές συσκευές και πλατφόρμες.

Στο επίπεδο πρόσβασης δεδομένων (Data Access Layer – DAL), πραγματοποιούνται όλες οι απαιτούμενες διαδικασίες, για να επιτευχθεί, τόσο η σύνδεση με τη βάση δεδομένων, όσο και ο χειρισμός

των ερωτήσεων και αποκρίσεων σε όρους SQL. Το συγκεκριμένο επίπεδο της εφαρμογής, είναι επιφορτισμένο με δύο σημαντικές εργασίες, οι οποίες είναι οι εξής:

1. Αποστολή των SQL ερωτημάτων του χρήστη προς το σύστημα HERMES και
2. Λήψη των πρωτογενών δεδομένων από τις αποκρίσεις του συστήματος HERMES



Εικόνα 14: Το VisualHERMES ως εναλλακτική διεπαφή

Όπως γίνεται κατανοητό, για να είναι σε θέση το εν λόγω επίπεδο να επικοινωνήσει με το σύστημα HERMES, θα πρέπει πρώτα να έχει εγκατασταθεί η ανάλογη σύνδεση, λειτουργία την οποία επίσης επωμίζεται, κάνοντας χρήση του εκάστοτε παρόχου δεδομένων, μέσω του οποίου επιτυγχάνεται και η επιθυμητή σύνδεση.

Αναφορικά με τη λήψη των πρωτογενών δεδομένων, θα πρέπει να σημειωθεί, ότι σημασιολογικά τα αποτελέσματα που αναμένει ο wrapper, έχουν να κάνουν με τροχιές κινούμενων αντικειμένων. Οι τροχιές αυτές, αποτελούνται από δύο δομικά συστατικά, τα οποία έχουν να κάνουν με τις γεωγραφικές συντεταγμένες που φέρει το προς παρατήρηση αντικείμενο, καθώς επίσης και τον χρόνο (Timestamp), κατά τον οποίο έχει πραγματοποιηθεί η λήψη των εκάστοτε συντεταγμένων. Επιπλέον, επειδή πρόκειται για αναπαράσταση τροχιών κινούμενων αντικειμένων, γίνεται αντιληπτό ότι απαιτείται η ύπαρξη ενός ζεύγους δύο σημείων για κάθε παρατήρηση, ένα για το σημείο και τον χρόνο εκκίνησης και ένα για το αντίστοιχο σημείο και χρόνο παύσης της κίνησης του κινούμενου αντικειμένου. Με βάση τα παραπάνω, ο wrapper, καταλήγει να έχει στη διάθεσή του, μια απόκριση, την οποίας τα αποτελέσματα υπόκεινται στην κάτωθι μορφοποίηση:

```
(X1, Y1) – (X2, Y2) # TimeStamp1 – TimeStamp2
```

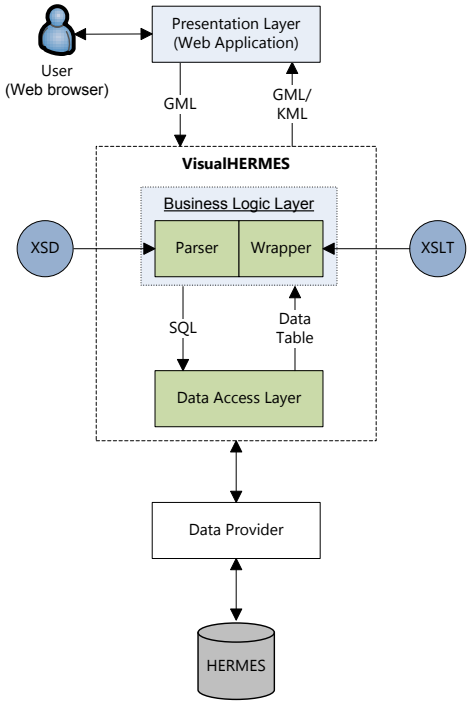
Φυσικά, το αποτέλεσμα της εκάστοτε απόκρισης δύναται να περιέχει πολλαπλές τέτοιες παρατηρήσεις για κάθε κινούμενο αντικείμενο, δημιουργώντας με τον τρόπο αυτό έναν ορμαθό από διαδοχικά σημεία και χρόνους.

Όσον αφορά τα SQL ερωτήματα που μπορεί να αποσταλούν από τον wrapper και δη το επίπεδο χειρισμού δεδομένων, προς το σύστημα HERMES, πρόκειται για τυπικά ερωτήματα που βασίζονται στο SQL πρότυπο σύνταξης ερωτημάτων και δύνανται να κάνουν χρήση τόσο χωρικών όσο και χρονικών τελεστών για την αναπαράσταση του επιθυμητού ερωτήματος. Μέρος των τελεστών αυτών παρέχεται από το τμήμα χωρικής υποστήριξης της Oracle (Spatial Oracle) και ένα άλλο μέρος από το σύστημα HERMES. Ο wrapper βέβαια, κάνει απλά χρήση των τελεστών αυτών, χωρίς κάποια επιπλέον διαδικασία.

Επιπρόσθετα, στο επίπεδο χειρισμού δεδομένων, έχει ανατεθεί η μετατροπή των πρωτογενών δεδομένων σε δομές κατάλληλες για την επεξεργασία τους από το αμέσως επόμενο επίπεδο. Με τον τρόπο αυτό επιτυγχάνεται η αποσύνδεση του συγκεκριμένου επιπέδου από το επόμενό του. Με άλλα λόγια, το επίπεδο αυτό μπορεί να παράγει το επιθυμητό αποτέλεσμα με οιονδήποτε τρόπο του ανατεθεί, αρκεί να καταλήγει πάντοτε στις ίδιες δομές δεδομένων.

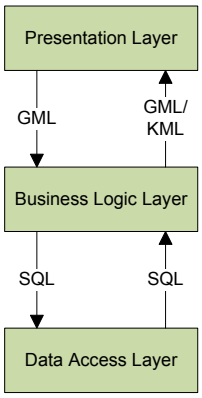
Για τη μέγιστη ευελιξία του wrapper, το επίπεδο χειρισμού δεδομένων, δεν κάνει χρήση (εντός του κώδικα) κάποιου συγκεκριμένου παρόχου δεδομένων. Εντούτοις, έχει γίνει χρήση γενικευμένου κώδικα, ο οποίος έχει την ικανότητα να προσαρμόζεται στον εκάστοτε πάροχο που θα του ανατεθεί, για την

ολοκλήρωση της όποιας εργασίας, επιτυγχάνοντας έτσι βέλτιστη προσαρμοστικότητα και διαφάνεια. Η δήλωση του επιθυμητού παρόχου δεδομένων, πραγματοποιείται από ένα εξωτερικό αρχείο ρυθμίσεων.



Εικόνα 15: Η αρχιτεκτονική του VisualHERMES

Το επίπεδο επιχειρησιακής λογικής (Business Logic Layer – BLL), είναι υπεύθυνο για τη μοντελοποίηση των πρωτογενών δεδομένων, που λαμβάνει από το επίπεδο χειρισμού δεδομένων, σε επιχειρησιακά αντικείμενα, τα οποία είναι σε θέση να περάσουν από τις απαραίτητες μετατροπές, ώστε να καταστούν κατάλληλα για τη δημιουργία του επιθυμητού αποτελέσματος. Παράλληλα, το συγκεκριμένο επίπεδο του wrapper, είναι επιφορτισμένο και με τη διαδικασία της επεξεργασίας και μετατροπής των GML ερωτημάτων σε SQL όρους και την αποστολή αυτού, στο επίπεδο χειρισμού δεδομένων και την τελική εκτέλεση του ερωτήματος (Εικόνα 16).



Εικόνα 16: Ο διττός ρόλος του επιπέδου επιχειρησιακής λογικής

Μέσω λοιπόν των κατάλληλων συναρτήσεων που έχουν αναπτυχθεί εντός του επιπέδου επιχειρησιακής λογικής, καθίσταται εφικτή η μετατροπή των επιμέρους επιχειρησιακών αντικειμένων σε αρχεία, τα περιεχόμενα των οποίων συμμορφώνονται με τα GML και KML πρότυπα.

Όπως προαναφέρθηκε, ο ρόλος του συγκεκριμένου επιπέδου του wrapper, είναι διττός και αφορά τόσο στη μορφοποίηση των αποτελεσμάτων, όσο και στη μετατροπή των GML ερωτημάτων σε SQL όρους. Για τις ανάγκες της SQL μετατροπής από GML, εντός του συστήματος έχει υλοποιηθεί μια σειρά από επιμέρους επιχειρησιακά αντικείμενα, επίσης υπό τη μορφή κλάσεων και τα οποία είναι σε θέση να αναπαραστήσουν τους όρους ενός SQL ερωτήματος, καθώς επίσης και ένα διαισθητικό τύπο του ερωτήματος αυτού. Οι κλάσεις που έχουν υλοποιηθεί, μπορούν να συνοψισθούν στις εξής:

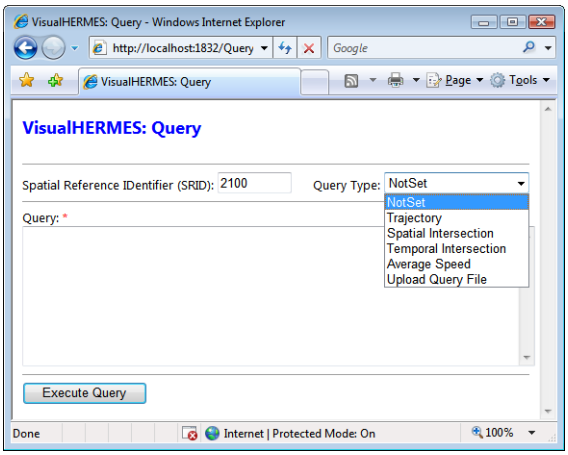
- QueryType: Λεκτικό του τύπου του ερωτήματος (Trajectory, Spatial, Temporal κ.λπ.),
- TimePoint: Αναπαράσταση μιας χρονο-σφραγίδας (TimeStamp),
- TemporalWindow: Αναπαράσταση ενός χρονικού παραθύρου που απαρτίζεται από δύο TimePoints,
- SpatialWindow: Αναπαράσταση ενός γεωγραφικού παραλληλόγραμμου, το οποίο απαρτίζεται από δύο Points που αναπαριστούν τις επάνω-δεξιά και κάτω-αριστερά γωνίες του και
- Query: Αναπαράσταση των όρων που δύνανται να εμφανίζονται σε ένα ερώτημα όπως τύπος ερωτήματος, κωδικός αντικειμένου, κωδικός τροχιάς, χρονικό παράθυρο κοκ.

Ο συνδυασμός λοιπόν του τύπου του ερωτήματος, με τους επιμέρους όρους αυτού, θέτουν την εφαρμογή, μέσω των αντίστοιχων συναρτήσεων που έχουν υλοποιηθεί, σε θέση να συναρμολογήσει το κατάλληλο SQL ερώτημα και μέσω του επιπέδου χειρισμού δεδομένων, να αποσταλεί προς το σύστημα HERMES για εκτέλεση.

Σε προηγούμενη αναφορά, έγινε λόγος για τους διάφορους τύπους εφαρμογών, οι οποίες μπορούν να κάνουν χρήση του VisualHERMES wrapper, έτσι ώστε να παρέχεται στον τελικό χρήστη ένα οργανωμένο περιβάλλον εργασίας, μέσα από το οποίο θα μπορεί να εκτελέσει τα επιθυμητά ερωτήματα, καθώς επίσης και να λάβει τα αντίστοιχα αποτελέσματα. Για τον σκοπό αυτό, δημιουργήθηκε μια διαδικτυακή εφαρμογή, βασισμένη στην τεχνολογία ASP.NET της Microsoft (Microsoft ASP.NET, 2008) και η οποία δίνει στον χρήστη τη δυνατότητα μέσα από μια εφαρμογή περιήγησης (Internet Explorer, Mozilla Firefox κ.ά.), να συνδεθεί με την υπηρεσία και να διεκπεραιώσει την επιθυμητή εργασία.

Υλοποίηση

Για την όσο το δυνατό ευκολότερη και πιο άμεση επαφή των χρηστών με το σύστημα VisualHERMES, έχει υλοποιηθεί μια δια-δικτυακή εφαρμογή, μέσω της οποίας ο χρήστης μπορεί να αποστείλει τα επιθυμητά ερωτήματα προς το σύστημα HERMES και να λάβει τα αντίστοιχα αποτελέσματα αφενός μεν μορφοποιημένα βάσει του GML προτύπου σε ένα αρχείο, αφετέρου δε οπτικοποιημένα με χρήση της υπηρεσίας Maps της Google. Όλα τα παραπάνω βέβαια, ολοκληρώνονται με την χρήση μιας εφαρμογής περιήγησης, όπως ο Internet Explorer, ο Mozilla Firefox κ.ά. (Εικόνα 17).



Εικόνα 17: VisualHERMES - Οθόνη δημιουργίας ερωτημάτων

Δημιουργία ερωτημάτων

Από την αρχική οθόνη της εφαρμογής, ο χρήστης μπορεί να επιλέξει τον τρόπο, με τον οποίο θα συναρμολογήσει το προς αποστολή ερώτημα. Οι πιθανές περιπτώσεις είναι οι εξής:

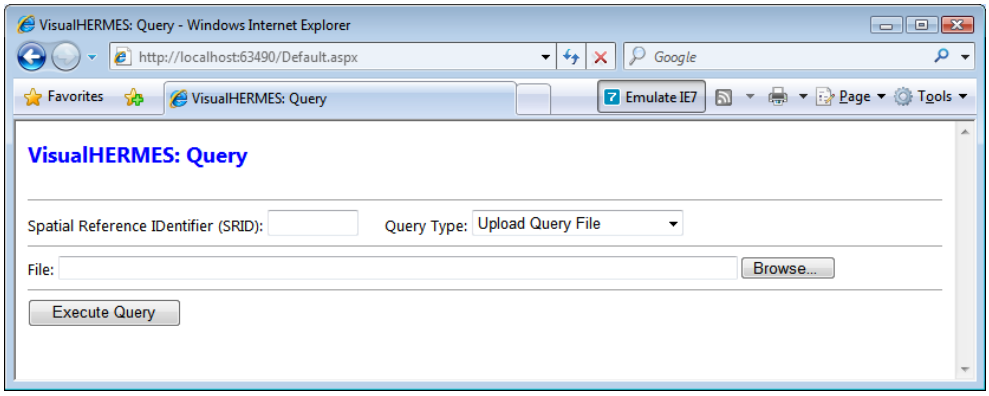
1. Χειροκίνητη συγγραφή του SQL ερωτήματος στο πεδίο Query,
2. Επιλογή ενός από τους προκαθορισμένους τύπους ερωτημάτων και
3. Αποστολή ενός GML αρχείου, στο οποίο θα υπάρχουν μορφοποιημένοι κατά GML, οι όροι που θα συμπληρώνουν το τελικό ερώτημα.

Στην περίπτωση που ο χρήστης επιλέξει τον NotSet τύπο ερωτήματος, παρουσιάζεται μια περιοχή κειμένου, στην οποία μπορεί να συντάξει ένα οποιοδήποτε έγκυρο ερώτημα SQL, και το οποίο στη συνέχεια αποστέλλεται στο σύστημα HERMES, μέσω του VisualHERMES wrapper. Στο σημείο αυτό θα πρέπει να σημειωθεί ένας περιορισμός που υπάρχει από το σύστημα, έτσι ώστε να είναι δυνατή η μοντελοποίηση όλων των επιστρεφόμενων δεδομένων. Ο περιορισμός αυτός έχει να κάνει με τα πεδία των αποτελεσμάτων που επιστρέφονται και τα οποία πρέπει να περιλαμβάνουν τον κωδικό του κινούμενου αντικειμένου (OBJECT_ID), τον κωδικό της τροχιάς (TRAJ_ID) και τέλος τις πραγματικές τροχιές που επιστρέφονται από το HERMES (MPOINT).

Όταν επιλεγεί κάποιος από τους προκαθορισμένους τύπους ερωτημάτων, παρουσιάζεται στον χρήστη μια ομάδα πεδίων, τα οποία καλούνται να συμπληρωθούν, έτσι ώστε το σύστημα να λάβει τις απαραίτητες τιμές για τη δημιουργία των ερωτημάτων. Οι επιλογές που έχει ο χρήστης στη διάθεσή του είναι οι:

1. Trajectory
2. Spatial Intersection
3. Temporal Intersection
4. Average Speed/Direction

Στην περίπτωση που ο χρήστης επιλέξει την αποστολή του ερωτήματος με χρήση GML αρχείου, βρίσκεται μπροστά σε ένα πεδίο, στο οποίο πρέπει να εισάγει τη διαδρομή και το όνομα του αρχείου αυτού, όπως παρουσιάζεται στον τοπικό του υπολογιστή.



Εικόνα 18: VisualHERMES - Οθόνη αποστολής GML αρχείου ερωτημάτων

Πρόκειται για ένα αρχείο κειμένου, του οποίου τα περιεχόμενα πρέπει να συμμορφώνονται με το XML πρότυπο αναπαράστασης δεδομένων, έτσι ώστε να είναι συντακτικά ορθό. Πέραν αυτού όμως, το προς αποστολή αρχείο, θα πρέπει να συμμορφώνεται και με ένα δεύτερο σχήμα, εντός του οποίου καθορίζεται το επιτρεπτό λεξιλόγιο που μπορεί να χρησιμοποιηθεί. Το σχήμα αυτό είναι καθορισμένο εκ των προτέρων, και μόνο αν το προς αποστολή αρχείο αξιολογηθεί επιτυχώς από τους δύο αυτούς ελέγχους θα είναι σε θέση να υποστεί την όποια επεξεργασία από το σύστημα VisualHERMES.

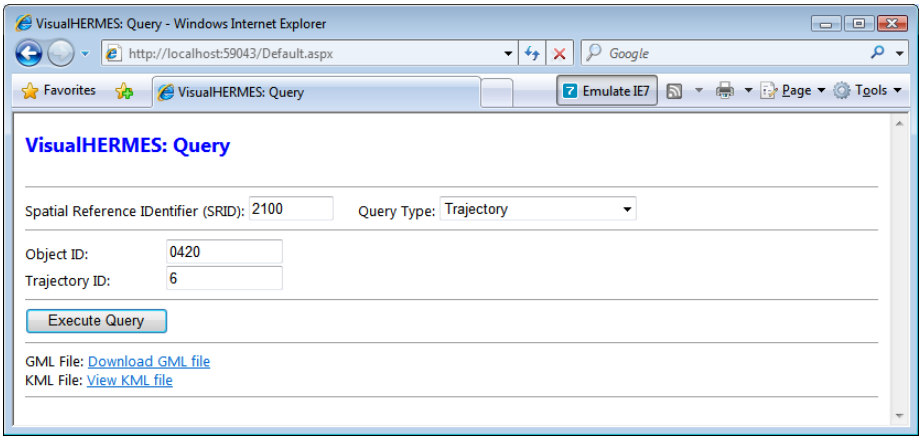
Δημιουργία αποτελεσμάτων

Με δεδομένο ότι έχουν δημιουργηθεί τα κατάλληλα επιχειρησιακά αντικείμενα για την περιγραφή των τροχιών που επεστράφησαν από το σύστημα HERMES, η επόμενη διαδικασία έχει να κάνει με τη δημιουργία των τελικών αρχείων που θα επιστραφούν στον χρήστη. Η εν λόγω διαδικασία, αποτελείται από δύο διακριτές υπο-διαδικασίες:

1. Δημιουργία του GML αρχείου, όπου ο wrapper διατρέχει όλα τα διαθέσιμα αντικείμενα τροχιών, που έχουν πριν δημιουργηθεί, και την προσθήκη σε αυτά των κατάλληλων ετικετών και
2. Δημιουργία του KML αρχείου, όπου ο wrapper, συμβουλευόμενος ένα εξωτερικό αρχείο κανόνων μορφοποίησης XSLT, μετατρέπει τα περιεχόμενα του GML αρχείου σε όρους KML.

Για τη δημιουργία του GML αρχείου, ο κώδικας του wrapper λειτουργεί αυτόνομα, χωρίς κάποια επιπλέον παρέμβαση. Αντιθέτως, για τη δημιουργία του KML αρχείου, ο wrapper, συμβουλευόμενος ένα εξωτερικό αρχείων μετασχηματισμών XSLT, μετατρέπει το αρχικό GML έγγραφο στο αντίστοιχο KML.

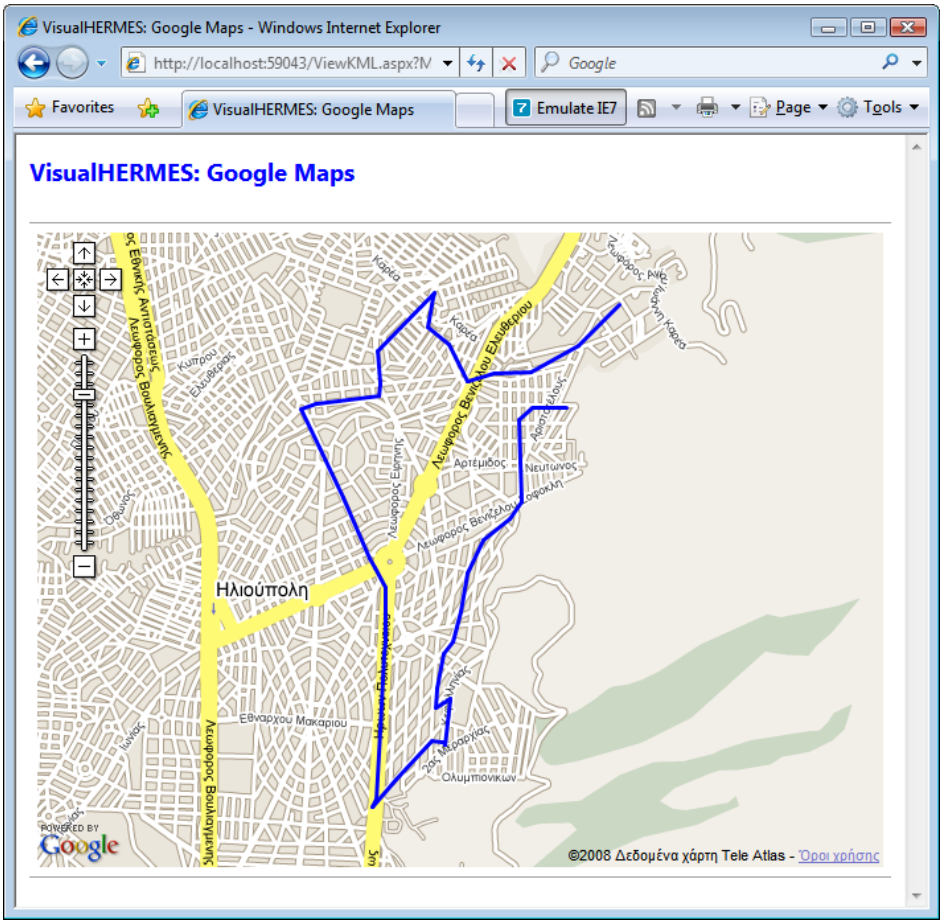
Τελικά, τα αρχεία που δημιουργεί ο wrapper VisualHERMES, παρατίθενται στον χρήστη που απέστειλε το ερώτημα, ως δύο υπερσυνδέσεις διαδικτύου (Hyperlinks).



Εικόνα 19: VisualHEMRES - Οθόνη λήψης αποτελεσμάτων

Το πρώτο εξ αυτών, αναπαριστά το GML αρχείο και το οποίο ο χρήστης μπορεί να μεταφορτώσει (Download) στον τοπικό του υπολογιστή.

Αναφορικά με το δεύτερο αρχείο, το οποίο αναπαριστά το KML αποτέλεσμα του αρχικού ερωτήματος, ο χρήστης μπορεί επίσης να το μεταφορτώσει στο τοπικό του υπολογιστή, καθώς επίσης και να δει τα αποτελέσματά του με τη βοήθεια της υπηρεσίας Maps της Google. Μέσω της διεπαφής προγραμματισμού εφαρμογών της υπηρεσίας Maps, έχει δημιουργηθεί μια δεύτερη σελίδα στο σύστημα, η οποία έχει ως στόχο να κάνει υπέρθεση των αποτελεσμάτων που έχει στη διάθεσή του το σύστημα VisualHERMES επί των αντίστοιχων χαρτών που παρέχονται από την Google (Εικόνα 20).



Εικόνα 20: VisualHERMES - Οθόνη οπτικοποίησης KML αποτελεσμάτων

Από το σημείο αυτό, ο τελικός χρήστης έχει στη διάθεσή του μια σειρά εργαλείων, τα οποία παρέχονται από τη μηχανή οπτικοποίησης της υπηρεσίας Maps, με τη βοήθεια των οποίων μπορεί να κινηθεί εντός του παρεχόμενου χάρτη, να προβεί σε μεγεθύνσεις ή σμικρύνσεις σε επιλεγμένες περιοχές

κ.λπ. Επιπλέον, κάθε τμήμα της οπτικοποιημένης τροχιάς, φέρει επιπρόσθετες πληροφορίες σχετικά με τον αύξοντα αριθμό της, τον κωδικό της όπως αυτός παρέχεται αρχικά από το σύστημα HERMES και τέλος τον κωδικό του κινούμενου αντικειμένου.

Συμπεράσματα

Το GML αποτελεί ένα ανοιχτό πρότυπο, βασισμένο στην XML τεχνολογία (Bray, και συν., 2006) και (Bray, και συν., 2006), το οποίο περιγράφει χωρικά και χρονικά δεδομένα για μεταφορά και αποθήκευση στο περιβάλλον του Διαδικτύου. Οι εξελίξεις που υπήρξαν το τελευταίο χρονικό διάστημα σε σχέση με την επέκταση του προτύπου στην έκδοση 3 (Cox, και συν., 2004), για την αναπαράσταση τοπολογιών, τρισδιάστατων γεωμετριών, επιφανειών και κινούμενων αντικειμένων, προσέδωσαν μια νέα δυναμική στην αποδοχή και την υιοθέτηση του προτύπου από την κοινότητα. Η επιτυχία της αποδοχής αυτής λοιπόν, έχει καταστήσει το GML ως μια λύση στο πρόβλημα του διαμοιρασμού των δεδομένων μεταξύ ετερογενών ομάδων χρηστών.

Τα XML δεδομένα δύνανται να οπτικοποιηθούν με διάφορες μεθόδους. Για την οπτικοποίηση χωρο-χρονικών πληροφοριών, οι οποίες βασίζονται στο XML, σε έναν web browser, απαιτείται η μετατροπή τους σε μια μορφή γραφικών, που ο τελευταίος να μπορεί να ερμηνεύσει. Υπάρχουν διάφορες επιλογές, οι οποίες παρέχουν λύσεις στο συγκεκριμένο αντικείμενο, καθεμιά από τις οποίες φέρει τα δικά της, μοναδικά, χαρακτηριστικά. Στην παρούσα έρευνα, επελέγη το KML πρότυπο μορφοποίησης (Google Inc., 2008), λόγω της γειτνίασής του με αυτό του XML και επειδή υπάρχουν ήδη υλοποιημένες μηχανές οπτικοποίησης των αποτελεσμάτων, χωρίς να απαιτείται η εγκατάσταση επιπρόσθετου λογισμικού (plug-ins). Μία τέτοια λύση, είναι και η υπηρεσία Maps της εταιρίας Google (Google Maps, 2008).

Το XSLT (Clark, 1999), αποτελεί μια εισήγηση του W3C, για τη μετατροπή ενός XML εγγράφου σε ένα άλλο. Με δεδομένο ότι το GML αποτελεί μια αναπαράσταση βασισμένη στο XML, το XSLT είναι σε θέση να μετατρέψει GML έγγραφα σε KML έγγραφα. Οι επιμέρους κανόνες μετατροπής ορίζονται εντός των XSLT stylesheets. Με την χρήση των stylesheets αυτών, μπορούν να παραχθούν επιπρόσθετες πληροφορίες από το αρχικό GML έγγραφο, οι οποίες δύνανται να εισαχθούν στο τελικό KML έγγραφο.

Τεχνικά, είναι δυνατή η ανάπτυξη λογισμικού για τη μετατροπή των GML δεδομένων σε οποιαδήποτε μορφή αναπαράστασης γραφικών, αλλά η μετατροπή από GML σε KML, με χρήση XSLT, φέρει κάποια πλεονεκτήματα. Κάνει χρήση της XML, μιας τεχνολογίας, επί της οποίας βασίζονται πολλές λειτουργίες του σημερινού Διαδικτύου και υποστηρίζεται από τους μεγαλύτερους παραγωγούς λογισμικού. Η διαδικασία παραγωγής διαφορετικών αναπαραστάσεων κρίνεται σχετικά απλή, διότι το XML, επιτρέπει το διαχωρισμό των δεδομένων περιεχομένου από τα δεδομένα παρουσίασης. Επιπλέον, στην περίπτωση όπου τα GML δεδομένα χρησιμοποιούν ένα προκαθορισμένο σχήμα εφαρμογής, οποιοδήποτε σύνολο δεδομένων μπορεί να αναπαρασταθεί με βάση το ίδιο stylesheet.

Το σύστημα HERMES (Pelekis, et al., 2006), αποτελεί μια επέκταση στο πακέτο λογισμικού Spatial του συστήματος διαχείρισης αντικειμενο-σχεσιακών βάσεων δεδομένων της Oracle στην έκδοση 10g (Oracle Corp., 2003). Μέσω την εν λόγω επέκτασης, επιτυγχάνεται η διαχείριση χωρο-χρονικών δεδομένων (τροχιών) για κινούμενα αντικείμενα, τα οποία μεταβάλλουν τη θέση ή/και το μέγεθός τους, ανά τακτά χρονικά διαστήματα ή συνεχώς. Επιπλέον, παρέχει όλη την απαιτούμενη υποδομή για την υποστήριξη αποστολής ερωτημάτων για τα κινούμενα αντικείμενα που διαχειρίζεται, με τη βοήθεια και χωρο-χρονικών τελεστών. Λόγω του γεγονότος ότι το σύστημα HERMES, αποτελεί μια πηγή δεδομένων (data source) τροχιών κινούμενων αντικειμένων σε τρίτες εφαρμογές, κρίθηκε σκόπιμη η υλοποίηση ενός wrapper, ο οποίος θα ήταν σε θέση να μορφοποιεί τις πληροφορίες αυτές σε όρους GML, πριν την αποστολή τους, με απώτερο στόχο την επέκταση της δια-λειτουργικότητας του συστήματος. Επιπρόσθετα, ο χρήστης θα μπορούσε να λάβει μια οπτικοποιημένη παρουσίαση των αποτελεσμάτων που ζήτησε, επί χάρτου, με τη βοήθεια της υπηρεσίας Google Maps, κάνοντας χρήση μιας thin client εφαρμογής, όπως ο web browser του. Ως εκ τούτου, τα δεδομένα μορφοποιούνται και σύμφωνα με τις επιταγές του KML.

Για τις ανάγκες της παρούσας έρευνας, σχεδιάστηκε και υλοποιήθηκε το σύστημα VisualHERMES. Μέσω του συστήματος αυτού, ο χρήστης είναι σε θέση να συνδεθεί σε μια διαδικτυακή υπηρεσία-τοποθεσία, από όπου μπορεί να αποστείλει ερωτήματα στο σύστημα HERMES, είτε επιλέγοντας έναν προκαθορισμένο τύπο, είτε αποστέλλοντας ένα GML έγγραφο ερωτήματος, να λάβει τα αποτελέσματα του ερωτήματος αυτού, μορφοποιημένα κατά GML και τελικά, κάνοντας χρήση της υπηρεσίας Google Maps, η οποία ολοκληρώνεται εντός του συστήματος VisualHERMES, να λάβει και την οπτικοποιημένη εκδοχή των αποτελεσμάτων του, αφού αυτά πρώτα μορφοποιηθούν με βάση το KML πρότυπο.