RADGIS – An improved architecture for runtime -extensible, distributed GIS applications

Thesis

Submitted in fulfilment of the requirements for the Degree of DOCTOR OF PHILOSOPHY of Rhodes University

by

Richard Michael Preston

November 2001

Abstract

A number of GIS architectures and technologies have emerged recently to facilitate the visualisation and processing of geospatial data over the Web. The work presented in this dissertation builds on these efforts and undertakes to overcome some of the major problems with traditional GIS client architectures, including application bloat, lack of customisability, and lack of interoperability between GIS products. In this dissertation we describe how a new client-side GIS architecture was developed and implemented as a proof-of-concept application called RADGIS, which is based on open standards and emerging distributed component-based software paradigms. RADGIS reflects the current trend in development focus from Web browser-based applications to customised clients, based on open standards, that make use of distributed Web services.

While much attention has been paid to exposing data on the Web, there is growing momentum towards providing "value-added" services. A good example of this is the tremendous industry interest in the provision of location-based services, which has been discussed as a special use-case of our RADGIS architecture. Thus, in the near future client applications will not simply be used to access data transparently, but will also become facilitators for the location-transparent invocation of local and remote services. This flexible architecture will ensure that data can be stored and processed independently of the location of the client that wishes to view or interact with it.

Our RADGIS application enables content developers and end-users to create and/or customise GIS applications dynamically at runtime through the incorporation of GIS services. This ensures that the client application has the flexibility to withstand changing levels of expertise or user requirements. These GIS services are implemented as components that execute locally on the client machine, or as remote CORBA Objects or EJBs. Assembly and deployment of these components is achieved using a specialised XML descriptor. This XML descriptor is written using a markup language that we developed specifically for this purpose, called DGCML, which contains deployment information, as well as a GUI specification and links to an XML-based help system that can be merged with the RADGIS client application's existing help system. Thus, no additional requirements are imposed on object developers by the RADGIS architecture, i.e. there is no need to rewrite existing objects since DGCML acts as a runtime-customisable wrapper, allowing existing objects to be utilised by RADGIS.

While the focus of this thesis has been on overcoming the above-mentioned problems with traditional GIS applications, the work described here can also be applied in a much broader context, especially in the development of highly customisable client applications that are able to integrate Web services at runtime.

Acknowledgements[*]

I would like to express my utmost thanks to my supervisors Prof. Peter Clayton and Dr George Wells for their constant support and valuable guidance while still allowing me to find my own way through the techno-jungle. Their open-door philosophy and constant willingness to help no matter what crisis or time constraints they were facing at the time were noted with much appreciation.

I would like to thank Prof. Peter Wentworth and Gillian McGregor for their enthusiastic willingness to join the team of proofreaders and for their valuable technical expertise. Henry Holland for his useful insights, and the many hours of thought-provoking discussion on GIS over coffee that would have dissolved the teaspoon if left in the cup too long.

A big thank you to Tina and Cheryl, as well as the rest of the members of the Rhodes University Computer Science department for all the small things that often get taken for granted, and for ensuring that I will always look back on my days in the department with fond memories.

Finally a warm word of thanks to my parents, my sister, and to all my very special friends for their support, especially during the writing of this thesis. This would not have been possible without you!

[*] This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Rhodes University and the NRF.

“Thanks to the Internet, back when you started reading this sentence …”

Found on “thought of the second” banner on Sun’s Java Developer web site.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

"All truth passes through three stages. First, it is ridiculed.
Second, it is violently opposed. Third, it is accepted as being self-evident."
Arthur Schopenhauer (1788-1860)

Geographic Information Systems (GIS) provide the user with the ability to explore geographic, or location-based data visually, allowing specialised searches on, and analysis of, both spatial and a-spatial (attribute) data. Traditional GIS applications provide a wide variety of tools for manipulating this data, from the digitising of spatial data to complex spatial analysis tools, conversion utilities, statistical analysis, as well as charting and presentation functionality.

A number of books and papers have been published that highlight the many advantages of digital mapping, and the ability of GIS applications to mine geospatial data effectively [Robertson *et al.* 1984] [Panel-GI 2000]. These advantages include increased speed, analytical and visualisation capabilities, as well as efficiency of data storage, integration of spatial and attribute data, and the ability to perform "finer-grained" spatial analysis [CTRE 1998] [Niemeier *et al.* 1993].

However, there are also a number of problems that hamper its effective deployment and use within organisations. These include problems associated with the resource-intensive nature of GIS applications due to the large data sets that are used, and the complexity of the algorithms used to process the data, as well as problems associated with the design of traditional GIS applications, such as application bloat and lack of interoperability between GIS software packages.

The work presented in this thesis describes our solution to these problems through the creation of an *extensible* client-side framework that provides the user with the ability to add distributed interoperable geospatial services to the client application at *runtime*. This new client-side GIS architecture was implemented as a proof-of-concept application called RADGIS (Runtime Application Development of GIS), and is based on open standards and emerging distributed component-based software paradigms.

The RADGIS architecture enables the creation of small, highly specialised GIS clients that are easily extensible should the user's needs change, or if the user requires a once-off or seldom-used service. This has been achieved through the creation of GIS services that are implemented by wiring together local and distributed objects using an XML-encoded descriptor language we have developed called DGCML (Distributed GIS Component Markup Language).

GIS applications typically provide extensive functionality, most of which is very specific to the field of geospatial data analysis, although some of it is more general in application. While the focus of this thesis is on the development of an improved client-side GIS architecture, much of the work presented in this thesis can be applied to the development of component-based applications in general. However the benefits of using our approach are maximised when applied to a domain that has been the focus of large-scale standardisation efforts, such as GIS. It is also easier to expound the virtues of an approach when one is able to provide concrete examples of how such an approach is able to solve specific problems identified within that domain.

If our objectives for the research presented in this dissertation were to be summarised in a single sentence it would be:

To develop runtime -extensible, highly-customisable, distributed-component based GIS and Location-based Service clients.

The remainder of this chapter expands on some of the problems associated with current GIS applications that can be attributed to the resource intensive nature of GIS, as well as the large number of diverse operations that may be performed. In particular, it focuses on two driving forces that have been the motivation for our approach to component-based GIS applications, namely reducing application bloat by disaggregating GIS applications into interoperable components (rather than interoperable software suites), and providing a high level of location-transparency for accessing local and remote data and services

.

1.1. Problems associated with current GIS Applications

One of the biggest challenges that organizations face today when deploying and using software-intensive systems is managing the complexity inherent in such systems, while being able to rapidly adapt to change, without a breakdown in the transfer of knowledge and experience when moving from one system to another [Brown 2000].

Monolithic Geographic Information Systems (GIS) packages of today are being replaced by new forms of geo-processing, based on new interoperable principles and standards. These changes are necessary due to the non-interoperability of current GIS products, which has lead to conceptual diversity, product specificity and the limited transfer of knowledge when a user changes from using one particular GIS package to another [Heywood *et al.* 1998][Open GIS Consortium, 1998a].

Traditional GIS applications are generally very large applications that are expensive to license, have a very steep learning curve [Albrecht 1996], and are difficult to use. Traynor *et al.* [1995] argue that proficient use of traditional GIS applications requires a solid understanding of the fields of geography, cartography and database management systems, as well as being computer literate, and that they very often require specialist knowledge.

Traditional GIS applications are well-suited to GIS experts. However there is a growing realisation that there are a number of users, in a variety of fields, who require spatial analysis tools similar to those found in GIS applications. These users generally require only a small subset of the functionality provided by traditional GIS applications, and do not necessarily wish to become experts in GIS in order to make use of GIS applications.

While there is no denying that traditional GIS applications fulfil an extremely valuable role, many problems associated with these GIS applications have been identified. The following is not meant to be an exhaustive list, but are described in more detail below due to their relevance to the research presente d in this dissertation:

?? "Application bloat" – many GIS applications attempt to provide as much functionality as possible. As a result, these GIS applications have become extremely complex systems that are prone to serious application bloat.

?? Lack of Interoperability – In the past, due to the lack of standards, GIS software from one vendor would not work with software from another. This lack of interoperability between software products from different vendors has been a major stumbling block to sharing geospatial data, particularly across the Web.

?? Location Transparency – while many GIS applications do provide access to data sets located on various different platforms with a large degree of location transparency, the ultimate goal is to extend location transparency to incorporate both data and services. This will allow processing of data to be performed wherever it makes most sense without requiring the user to be aware of the physical location of either the data set being processed, or the tool that is performing the processing.

1.1.1. Application Bloat

Classic application bloat within major software packages is becoming more and more common as developers try to add functionality to cater for every eventuality to their applications. However, different users have very different requirements, and may end up using very little of the overall functionality provided by an application. This is particularly true in GIS, where users typically only make use of a fraction of the available functionality found in most traditional GIS applications [Gunther and Muller 1999]

Ironically, it is also likely that an advanced user will require functionality not provided by the core application, even though it contains many features that will never be used. Such a user must then install additional modules, which may integrate with the original application, but are often stand-alone tools that must be executed outside the core application.

This is not entirely the developer's fault as it is impossible for developers to provide systems tailored to each individual's requirements. Most applications are developed based on generalised requirements, i.e. for groups of users with similar core requirements, but differing application environments that require specialized features. In addition, many end-users only discover what functionality they require once they start working on a particular project.

Application bloat is not unique to GIS applications. According to Dey *et al.* [1997a], the main downfall of most current software suites is their poor ability to integrate individual tools/services. They point out that tightly integrated suites of tools/services currently available are unsatisfactory because:

?? They require designers to predict how end users will want to integrate the tools/services provided; and

?? They force users to use particular tools/services with no opportunity to replace or add tools/services to the application.

The latter is mainly due to vendors wishing to capitalize on vendor "lock-in" and customer loyalty, by making it extremely difficult or costly for a user to change tool or service provider. The result is a large application, written by a single software house with general expertise, rather than individual components, written by experts in that field, that can be integrated into larger applications.

A solution to this problem would be to allow users to decide which tools they require for the job at hand. Should they determine, at any stage, that they do not have the necessary tools for a task, they should have the opportunity to locate implementations of these tools, and integrate them in their existing application, rather than having to purchase an additional package or separate stand-alone utility.

1.1.2. Interoperability

Interoperability has been a major stumbling block within the computing industry as a whole in the last decade. Until recently, the lack of standards within GIS hindered the widespread adoption of GIS applications because organisations were reluctant to buy into software from a vendor that did not provide a simple migration path, or interoperate with software from other vendors [Ferris 1998].

However, it is clear from the numerous standardisation efforts presently underway, particularly within the many XML dialects, that a high priority has been placed on developing interoperable systems based on open standards. In addition, the OpenGIS® Specification being developed by the Open GIS Consortium represents an evolution in GIS solutions, in which proprietary data models and software functions are made interoperable and extensible.

The term "interoperability" is used to describe the ability of software (possibly distributed on multiple machines) from multiple vendors to freely exchange data between systems. This is *possible* when each system has knowledge of the other systems' proprietary formats, but is *guaranteed* when data is transferred between systems that support an open standards format. The use of open data standards, i.e. data formats whose internal structures have been openly published, allows developers to create "small" interoperable components independently of each other. This permits GIS users to perform tasks that require functionality from more than one vendor, based on the integration of these software components.

According to Goodchild *et al.* [1997], the adoption of open standards will lead to the development of "similar" systems that make use of the same vocabulary, follow the same conventions and ensure that interoperability over a wide range of systems becomes possible. This will, in turn, result in a *simplification* of data formats, improve interaction between the user and a particular system, and reduce the amount of knowledge required by a user to be effective with respect to that type of system. Therefore users could achieve the same outcome with less knowledge, and training in one system, e.g. ARC/INFO, would not be wasted if the user was transferred to another similar system, e.g. MapInfo.

Open systems facilitate interoperability by allowing vendors to produce competing products that are interchangeable with existing components. This healthy competition ensures that end users are able to replace the implementation of a particular service with another, possibly superior service, without changing the base application code. The use of interoperable components also facilitates the development of highly scalable systems, and the packaging of particular services, resulting in lower costs and products tailored for specific end-users' requirements.

The ability to utilise interoperable services in the creation of task-oriented clients is of great benefit to the GIS community [Albrecht 1996]. Application developers have the flexibility to select services based on the requirements of the end-user by choosing the service implementations that are best suited to the task at hand. In addition, developers of applications for traditionally non-GIS users can implement small, customised applications that require a small subset of a traditional GIS application's functionality, by combining interoperable services rather than developing them from scratch.

This can only be achieved through the development of vendor-neutral, standards-based frameworks that enable the discovery and integration of multiple online geodata sources and services distributed over the Web. According to Kenn Gardels [*current*], the future success of GIS as a technology, and as a paradigm of spatial understanding, will depend on the seamless integration of these diverse methods into a comprehensive system for scientific investigation and environmental planning.

1.1.3. Location transparency

Goodchild et al. [1997] describe *transparency* as the ability to work at a conceptual level rather than having to be aware of the implementation issues, thus providing "*a uniform view of multiple, heterogeneous, distributed, and autonomous participating systems*".

The ability to provide a high level of location transparency within GIS applications allows users to utilise geospatial data and services without necessarily being aware of where the datasets are stored or where the geoprocessing is being performed. This reduces the complexity of using distributed data and services, which might otherwise prove unmanageable for all but expert users.

Location transparency is an extremely important feature of new applications that embrace the trend towards a highly-networked model of computing. This has a dramatic impact on the architecture of applications, and as technology improves, the distinction between accessing a local or remote resource will slowly fade altogether.

1.2. Motivation

While there will always be a need to provide the bundled functionality found in traditional GIS applications, there is a growing need to enable non-specialist users to make use of GIS operations in a more user-friendly manner, and customised to their particular field of interest.

At a specialist meeting held in December 1997, under the auspices of the Varenius Project, the workshop agreed that in the future GIS applications would become [Goodchild *et al.* 1997]:

?? *Distributed* – enabling a user to access data and processing, as well as collaborate with other users located throughout the world. For example it would be possible for a user at location A to send data from location B to a server at location C to be processed, and have the results returned to location A for display.

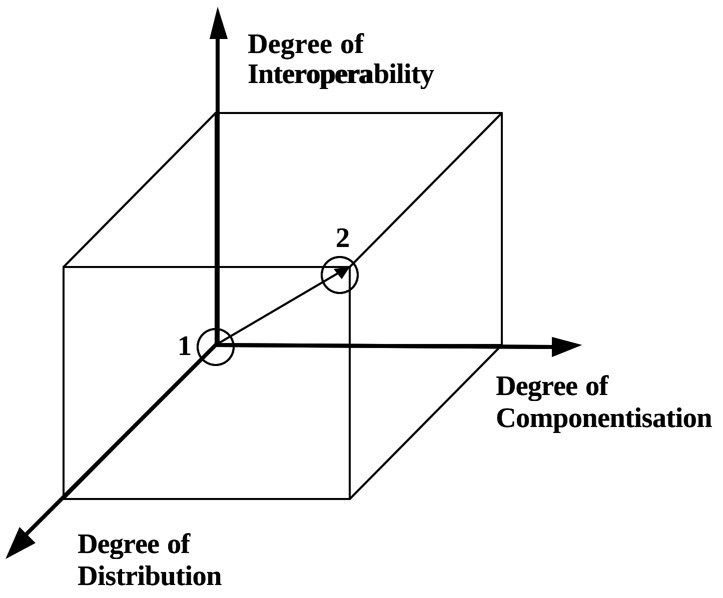
?? *Disaggregated* – as the use of interoperable, standards-based Commercial-off-the-Shelf (COTS) software components, developed by different vendors, replaced monolithic applications developed by a single vendor.

?? *Decoupled* – as disaggregated components are no longer part of a single application, but are distributed over many networked systems.

?? *Interoperable* – a clear precondition for all three of the above-mentioned points.

This vision of the future of GIS applications addresses most of the problems that we outlined earlier with respect to GIS software, including reducing application bloat, lack of interoperability and location transparency. Of particular importance to the success of this vision are the recent developments in component-based architectures and distributed-object computing which encourage flexible plug and play systems that are extensible, and allow heterogeneous components to interoperate across diverse platforms and network protocols.

The successful integration of these technologies will lead to the creation of GIS applications composed of distributed services, implemented as interoperable components (see *figure 1.1*).



1. **Local non-interoperable GIS package**
2. **Fully interoperable distributed GIS components**

Figure 1.1. The new trend towards distributed interoperable GIS components [Alameh 2001]

The problems with GIS applications that we outlined in *section 1.1*, as well as the vision expressed by Goodchild *et al.* [1997] for the future of GIS applications, are not restricted to the domain of GIS applications, but are in fact general application development issues for most resource-intensive Internet-based applications.

However, we have focussed our research on the field of GIS for three main reasons:

?? an overriding interest in the field of GIS,

?? the large amount of standardisation which has occurred within this discipline over the past few years, and

?? by constraining our discussion to the field of GIS, we are able to address particular issues and provide a more pertinent way to discuss the benefits of our approach by looking at specific scenarios.

Our approach to solving these problems does not make use of GIS-specific technology. Instead we have applied emerging computing principles and technologies, such as component-based software development, XML and distributed-object technologies to the field of GIS.

1.3. State-of-the-art

GIS applications have undergone many architectural changes over the past two decades that have been in keeping with advances in tiered application development in general. This initially saw GIS applications evolve from a single, tightly-coupled or single-tier application to a two-tier client-server architecture. This change was made to separate the data management duties from the operations and analysis logic, and the rendering and user interface, allowing a database other than the GIS vendor's proprietary database to be used. More recently, GIS application development has progressed to using the N-tier architecture, primarily to facilitate distributed processing, Web-based mapping and "thin-client" spatial data viewers.

Another trend worth noting is the increasing prevalence of Java-based GIS applications and location-based services, which is evidence of the wide-spread industry adoption of Java as a powerful tool for developing platform-independent distributed GIS software.

1.3.1. GIS Standardisation

A number of standardisation bodies are currently working on different aspects of GIS, most noticeably the Open GIS Consortium [1998a] and ISO Technical Committee 211 [ISO/TC 211 2000]. In general, the OGC is concerned with software specifications, while ISO/TC 211 concentrates more on data standards. However, in order to ensure that their work, which is often complementary (e.g. the work done in defining the geometry model), does not result in competing standards these two standardisation organisations are fully coordinated [Open GIS

Consortium, 1998a]. The OGC has agreed to submit their specifications for ISO approval via ISO/TC 211, and a "Class A Liaison" between the ISO and the OGC ensures that their efforts are harmonized, and that mutual experiences and results are shared.

The Open GIS Consortium (OGC) has dedicated much time and effort towards solving the interoperability issues outlined previously. Its major objective was to produce a single operational model for all spatiotemporal applications that would enable an application developer to combine geospatial data, and any geospatial function or process, available on the Web.

To date, the major achievements of OGC include [Cox 2000]:

?? the *Simple Features Specification* – a subset of the ISO/TC 211 data model required to support basic GIS systems;

?? the *Geographic Markup Language (GML)* – an XML encoding of the Simple Features Specification; and

?? the *Web Map Server Interface Specification* – for producing maps of georeferenced data, based on a standardised request mechanism.

In addition, the OGC has defined geospatial domain-specific business objects to ensure that the OpenGIS® Services Architecture[1] can be realized with standards-based, Commercial-Off-The-Shelf (COTS) products available from multiple vendors [Open GIS Consortium 2001a].

These OpenGIS® standards, developed by the Open GIS Consortium, are being closely followed and adopted by all the major GIS vendors including Integraph, ESRI, Bentley, and MapInfo. Therefore, by developing open standards for geoprocessing, the OGC is actively shaping the future of GIS applications and enabling "*geoprocessing to become an integral part of the evolving distributed computing paradigm in which applets, middleware, components, e-commerce tools, and object request brokers give any networked computing device real-time access to a huge universe of data and processing resources*" [Open GIS Consortium, 1998a].

[1] The OpenGIS® Service Architecture is a framework of services that are required for the development and execution of geospatial applications.

1.3.2. Component-Based Software Development

Component-Based Software Development (CBSD) allows systems to be developed from a number of existing interoperable system elements with exposed interfaces and hidden implementations. Therefore a system no longer needs to be built from scratch, but can be developed by selecting, reconfiguring, adapting, assembling and deploying encapsulated, replaceable, interoperable components [Barroca *et al.* 2000] [Clements *et al.* 2000].

The development of systems using existing, pre-tested components has a number of tangible and intangible benefits, including shortened system development cycles, increased productivity through component reuse, higher quality systems, reduced time-to-market, as well as reduced development and maintenance costs. See *section 2.2* for a more comprehensive list of benefits.

The integration of the Web and component-based systems implemented using distributed object technology provide a number of additional benefits, which include the ability to share computing and data resources, platform and operating system independence, increased efficiency by distributing the workload across multiple machines, and the ability to use the same distributed resources on a number of different devices [Fan *et al.* 2000] [Fingar *et al.* 1997]. See *section 2.4* for a more comprehensive list of benefits associated with the use of distributed component technologies.

There are three major distributed component or component-oriented middleware technologies that are currently used for developing enterprise-scale component-based applications, namely the OMG's CORBA Specification [OMG 1999] [Siegel 2000], Sun's Enterprise JavaBeans (EJBs) [Sun Microsystems 2000b] [Roman 2000] and Microsoft's Distributed Common Object Model (DCOM[1]) [Sessions, 1998a]. We have decided to focus on the use of CORBA objects and EJBs because of the platform-independent nature of CORBA and Java (see *section 2.4*).

[1] DCOM, COM, COM+, information available at http://www.microsoft.com/com/

1.3.3. Technology convergence (Where is GIS heading?)

The convergence of the Web and distributed object technology has resulted in a multitude of new applications and services. Just as the value of a fax machine increases with use and with an increase in the number of other fax machines to which it can connect and communicate, each distributed application and service added to the Web increases the value of the Web directly in terms of the functionality it provides, and indirectly by increasing the value of other interoperable applications and services.

David Schell, acting president and CEO of the Open GIS Consortium, believes that in the future data content providers, connectivity service providers, platform providers, service technology providers, and vertical service providers will slowly replace the traditional GIS provider [GeoInformatics 2001]. This view is shared by a number of people including Alameh [2001] (see *figure 1.2*), and Brox and Kuhn [2001] who argue that the future market for geographic information will be a market of geographically referenced information products generated by technical and organizational services applied to data. Indeed this can already be seen in the increasing number of stand-alone GIS services, location-based services, and the integration of geospatial information into mainstream IT applications, particularly through the use of Oracle Spatial database technology.

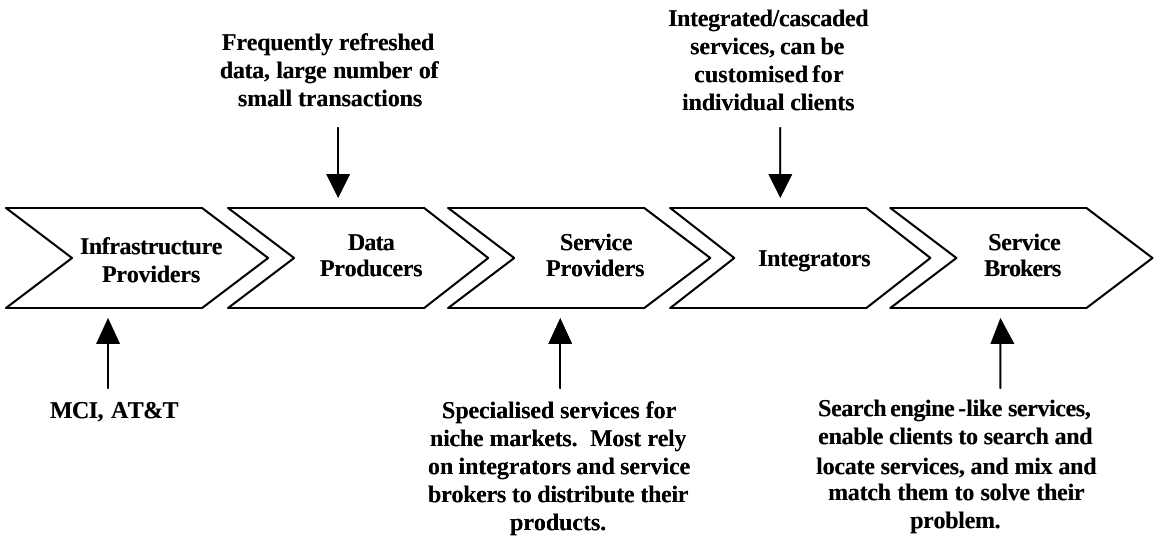

Frequently refreshed data, large number of small transactions
Integrated/cascaded services, can be customised for individual clients
Infrastructure Providers
Data Producers
Service Providers
Integrators
Service Brokers
MCI, AT&T
Specialised services for niche markets. Most rely on integrators and service brokers to distribute their products.
Search engine -like services, enable clients to search and locate services, and mix and match them to solve their problem.

Figure 1.2. Potential value chain for the future GIS marketplace [Alameh 2001]

The work undertaken by the Open GIS Consortium aims to make interoperability easier and more powerful by defining open standards for storing, delivering and processing geospatial data [Lake 2001b]. The following emerging technologies, which are currently the focus of much of the OGC's efforts, are very likely to become the foundation of future GIS:

?? GML – a geospatial data standard that is likely to be widely adopted as an exchange format for GIS services.

?? The OpenGIS® Service Architecture – a framework of interoperable services required for the development of geospatial applications.

?? Web Map Servers and Web Feature Servers – for the efficient access to geospatial data, especially when serving the requested map as GML.

?? Catalog and Registry Servers – extremely valuable services that allow users to register and locate data or services, based on associated metadata stored about that data source or service.

While certainly not the focus of this thesis, it would be a gross oversight if one neglected to mention the huge impact that wireless communication is having on Internet applications and mobile E-commerce in general, and within the field of GIS in particular. Of specific interest to us is the relatively recent emergence of location-based services, born out of the convergence of wireless communication, the Web and GIS technologies.

In contrast to GIS applications, location services are particular applications of spatial and analytic functions found in GIS applications, which filter their content or change their behaviour, based on the user's (specified) location. Location services hide the complexity of GIS tools by providing an easy-to-use interface for a specific service. This interface makes use of one or more GIS tools behind the user's back in order to provide the location-based service, and thus no longer requires that the user be knowledgeable in geography or cartography. In fact, if the user interface is simple enough, very little computer literacy is required for a user to be able to make use of complex GIS operations transparently through location services.

Mobile phones are the most widely used form of wireless device, and therefore offer the greatest potential market for location services. While some mobile device manufacturers have embedded, or pla n to embed, GPS devices in their mobile devices, existing mobile phone users will be able to benefit from an increasing variety of location-based services as the accuracy with which a service provider is able to triangulate their position improves.

The ability to accurately determine the location of a mobile phone user opens up a plethora of new services and business opportunities which were previously unavailable, or limited in their efficiency and effectiveness, because they were dependent on the location of the user. However, it may be surprising to learn that the main impetus behind increasing the accuracy with which a mobile phone user's position may be calculated in the United States is not commercial, but rather the provision of emergency services. This took the form of a US Federal Communication Commission E911 mandate requiring that by the 1st of October 2001, wireless service providers in the US must be able to provide the location of mobile handset users to within approximately 125m, 67 percent of the time [Lopez 2000][1].

However, the relatively limited accuracy currently available has not deterred a growing number of industries from providing location services that deliver Web mapping, street routing, traffic reports and electronic yellow pages to Web and wireless devices.

According to the International Data Corporation [IDC 2000], the GIS market has realised fairly linear growth in the range of 10-15% annually since its establishment, centred primarily on providing mapping and spatial analysis tools to specialist users. However, despite its infancy, the rate of growth for location services has already exceeded that of the traditional GIS market.

Location-based services are of interest to us for two reasons: the first is that the introduction of location services is moving the GIS community towards component-based GIS applications, based on open systems; and secondly because the provision of location services over the Web gives us another GIS-related application against which we can evaluate our research into building applications at runtime.

[1] Due to the complexity, lack of equipment and cost of implementing the necessary technology in the base-stations, this deadline has subsequently passed without a single wireless service provider meeting this requirement.

Other topical research areas within the field of GIS that are receiving much attention are the display of 3D geospatial information [Dykes et al. 1999] [Preston *et al.* 1997, 1999c] [Reddy *et al.* 1999-2000], the use of mobile agents in distributed GIS [Conde 1998] [Preston *et al.* 1999b], and the inclusion of time-based data and operators to provide spatiotemporal or 4D GIS support [Langran 1992] [Pequet and MacEachren 1998] [Preston *et al.* 1998a, 1998b] [Snodgrass *et al.* 1998].

1.4. How does our research address these problems?

A number of distributed GIS-related research projects are currently being undertaken, including DisGIS (Distributed GIS) [Berre et al. 2000], plug-and-play GIS components [Lemmens 2001], [Tsou and Buttenfield 1998] and CommonGIS [Voss and Birlinghoven 2000]. Each of these systems aims to address one or more of the problems outlined previously. However, the RADGIS provides an improved client architecture because it is a runtime-extensible architecture, based on open GIS standards, that allows a user to determine what functionality is provided by their client application as well as how the client application integrates the interoperable GIS services.

This ability to add functionality, as and when required, rather than attempt to provide built-in functionality for every eventuality, ensures that the RADGIS architecture does not suffer from the "application bloat" associated with traditional GIS clients. It also guarantees that the user does not have to change to a different client application when additional functionality, not provided by the current client, is required.

One of our design goals was to create a system that is extensible through the asynchronous addition of, or upgrade to, system components. The ability to facilitate tight integration of software components, without the need for unnecessary component adapters, requires consensus on the names, types, and the semantics of components' input and output. Interoperability of GIS components is assured through the adoption of open GIS standards, which are currently the focus of a number of standardisation bodies, most noticeably the Open GIS Consortium and ISO/TC 211.

The use of Web mapping services and online geoprocessing tools that allow users to access remote geospatial data and to process their data sets over the Web, is both possible and highly desirable. Autonomous components and services are the easiest to integrate into the RADGIS client application, due to their ability to work independently of the client application. However, through the use of standardised naming conventions we are able to demonstrate the ability for the RADGIS architecture to facilitate "tight" integration of distributed services with the core client application and other services.

Interoperability, according to [Goodchild et al. 1997], also means commonality in user interaction, which can be achieved through the development of interfaces that can be customized to provide a familiar 'look and feel' to the user. RADGIS allows the user to make use of a single customisable client application, which improves upon the limited transfer of knowledge inherent in switching from software developed by one vendor to another. This is achieved by allowing the user to make use of new services within a familiar client framework, and by allowing the user or service developer to customise the GUI of a service to provide a familiar "look and feel".

One of the problems with implementing a distributed GIS application, which is addressed by the RADGIS architecture, is the problem of invoking distributed GIS tools for which no compile-time knowledge exists. This is particularly true for complex operations that require the user to interact with distributed objects using a GUI. However, rather than download compiled GUI classes, we have decided to create easily modifiable XML documents that provide meta-information about the GIS service and allow one to wire-together Java objects to create a GUI or specify batch processes. A few XML-based scripting languages have been developed that allow one to wire together JavaBeans (including Swing components), e.g. Bean ML, or to specify Swing GUI's in XML, e.g. XwingML. However, we have developed our own wiring language (DGCML) that goes beyond the functionality provided by these scripting languages, and which serves four main purposes:

?? the deployment of GIS services for integration with the RADGIS client at runtime;

?? the specification of a GUI for the GIS service;

?? the ability to specify remote method calls to CORBA objects and EJBs; and

?? the provision of links to the associated help files which can be integrated into the RADGIS client's help system.

Another issue addressed by the RADGIS architecture is that of location transparency. The ability to specify GIS services using DGCML allows the user to make use of a service without necessarily knowing where the processing is taking place. The method-calls invoked by a particular service may be made on objects executing locally or on objects implemented as CORBA objects or EJBs residing on remote machines. If alternate codebases are provided for a particular service, it is possible for the RADGIS system to elect which codebase to use based on the location of the data to be processed. Therefore RADGIS affords the user of a service a high level of location transparency when performing geoprocessing operations.

During the course of our research, we have observed the tremendous industry attention that is being given, especially within the field of GIS, to the provision of location-based services. We have therefore used location-based services as an example to show how our RADGIS architecture can be extended to other application domains. We also believe that the Location Services model reflects the future of GIS applications in terms of being distributed, disaggregated, decoupled and interoperable.

Our approach focuses on a number of architectural issues currently being addressed by the GIS community, and in doing so, provides a flexible and extensible solution that caters both for the novice and expert user, in a wide range of GIS-related tasks. Not only will this improve the level of geospatial data access and use among traditional GIS users such as cartographers, planners, scientists, and environmental protection agencies, but it will also enable non-technical users to access this information as well (especially through the use of location services).

1.5. Thesis Organisation

This introductory chapter mapped out the problems inherent in current GIS architectures, particularly monolithic GIS client applications, that are to be addressed in this thesis. It then proceeded to provide a concise overview of current developments within the field of GIS, as well as future trends, providing some examples of research that is being undertaken in similar areas.

The RADGIS architecture was introduced briefly in order to explain how we intended solving the problems with current GIS architectures outlined previously. This chapter also explained why we decided to constrain our focus to GIS applications even though many of the problems with current GIS architectures that have been identified are also inherent in other resource-intensive applications. However, because our solution makes use of general component-based software development techniques and distributed object technologies, the work described here can still be applied in a much broader context.

The remainder of this thesis is structured as follows:

Chapter 2 (*CBSD*): The RADGIS architecture relies heavily on distributed Component-Based Software Development (CBSD) techniques and the use of distributed-object technologies such as CORBA Objects and EJBs to implement distributed geospatial services. This chapter provides background information required by the reader to fully understand the implications (both the inherent problems as well as the intended benefits) of the chosen component-based design.

Chapter 3 (*XML – Enabling CBSD and Deployment*): The specification and deployment of GIS services that can be integrated with the RADGIS client at runtime, is done using a meta language we developed called DGCML. This chapter provides some background on XML, and provides some examples of its use as a deployment language and as a wiring language, before describing our DGCML vocabulary and the functionality it provides.

Chapter 4 (*Factors influencing the design of RADGIS*): This chapter presents a classification of GIS application architectures as well as a brief overview of the work that has been undertaken by the Open GIS Consortium in the standardisation of GIS, in order show the relevance of our work and where it integrates with this global view of where GIS is headed. It concludes with a description of what our RADGIS architecture does and what it hopes to achieve.

Chapter 5 (*Implementation of the RADGIS Application*): Provides a detailed description of how our the RADGIS client application was implemented, including the ability to visualise 3D geospatial data and integrate distributed GIS services developed and deployed using DGCML.

Chapter 6 (*Discussion*): This penultimate chapter highlights some design considerations for distributed object developers and component integrators, as well as the implications of using the RADGIS architecture on end-users. It then lists the qualitative benefits of using the RADGIS architecture before taking a brief look at some of the technological and business-related issues that would need to be addressed in a commercial implementation of the RADGIS architecture.

Chapter 7 (*Concluding Remarks*): The final chapter of this thesis is devoted to providing a critical assessment of the RADGIS architecture, including its limitations, and motivates the contributions that the research undertaken in this dissertation has made to the field of distributed GIS.

This thesis describes an *architecture* that is made up of a number of new and emerging technologies. Therefore, instead of providing background chapters and then detailing our work separately, each chapter contains some background material necessary for explaining the design considerations that were taken into account in the development and implementation of the RADGIS client.

The reader's attention is also drawn to the Glossary of Terms, located immediately after the appendices. This should provide a useful and convenient resource for locating concise definitions of various technical terms and acronyms used in this dissertation.

Chapter 2

Component-Based Software Development

As a common witticism goes,

"the only difference between a software component and a virus is the author".

A Gartner Group study [Gartner Group] estimates that the market for pre-built components will have grown from $1.4 billion in 1997, to more than $8 billion in 2002. Furthermore, i is estimated that by 2003, at least 70 percent of all new applications will be deployed as a combination of pre-assembled and/or newly created components, integrated to form complex business systems.

According to Stojanovic [2000], component-related research can be also found under various subjects such as module interconnection languages (MILs) [Prieto-Diaz and Neighbors 1986], module interface specification and analysis [Perry 1989], megaprogramming [Wiederhold *et al*. 1992], domain-specific software architectures (DSSAs) [Fischer 1994], software generators [Batory and Geracy 1996], object-oriented frameworks and patterns [Gamma *et al.*, 1995] [Fayad *et al.* 1999] and architecture description and configuration languages (ADLs) [Garlan and Perry 1995].

This provides a clear indication of the importance placed on CBSD, and illustrates the paradigm shift from developing relatively small, centralised monolithic systems to complex enterprise systems composed of distributed components that may accessed across a corporate intranet or the Internet.

This chapter will present a brief introduction to some of the work currently being done in the field of Component-Based Software Development (CBSD) in order to highlight the benefits and the complexity involved in developing reusable components that are compatible and substitutable with other components within a component framework.

We then present two distributed component technologies, CORBA and EJBs, that have been used in our RADGIS application to implement distributed GIS tools that can be used together with our DGCML vocabulary to develop and deploy customised GIS services.

2.1. CBSD Basics

Component-Based Software Development (CBSD), and the concept of 'Commercial Off-The-Shelf' (COTS) components, is generating tremendous interest from industry and researchers alike, because of the many potential benefits to be gained from developing applications using plug-and-play reusable software components, rather than building the whole system from scratch [Hernandez *et al*. 2000] [Brown 1997] [Brown and Wallnau 1998] [Szyperski 1998].

In theory, developing an application using the CBSD paradigm is a simple task of browsing a component catalogue or library, selecting the appropriate components, and then reconfiguring, adapting, assembling and deploying them [Barroca *et al.*, 2000]. However, in practice, CBSD is very seldom simply a matter of plug-and-play development, and component re-use is often difficult to achieve across application domains.

2.1.1. Terminology

Looney *et al.* [1998] state that the lack of an agreed definition of what comprises a software component has led to some confusion as to how to identify and re-use components. Therefore we will briefly introduce some CBSD terminology to clarify what we mean by software components, componentware, and component frameworks.

2.1.1.1. Software Component

According to [Schneider and Nierstrasz 1999], software components are *static abstractions with plugs*, i.e. they encapsulate their implementation and only interact with their environment through well-defined interfaces

A software component or, for the purposes of this dissertation, simply a component, has a number of characteristic properties, including [Booch *et al.*, 1999] [Schneider and Nierstrasz 1999] [Szyperski 2000]:

?? it is a unit of independent deployment;

?? it has no observable state;

?? it must be instantiated in order to be used;

?? it is a replaceable part of a system; and

?? it should come with clear specifications of the services/events it provides and the services/events it requires.

2.1.1.2. Componentware

The term *componentware* [Gartner Group 1997][Sessions 1998b] defines applications assembled from a set of *software components* [Ring and Ward-Dutton 1998] [Bergner *et al.* 1999]. These software components are not used in isolation, but are elements of a *component framework* .

2.1.1.3. Component Framework

A component framework is an architectural template that facilitates the efficient development of complex systems using components. It determines the interfaces that components may have, and how the components are plugged together, i.e. a component framework is a collection of collaborating software components and architectural styles [Schneider and Nierstrasz 1999].

2.1.2. Granularity of components

Components can be divided into two major categories [Hurwitz 1998]:

?? large-grained components that implement complete units of business functionality; and

?? fine-grained components that implement small units of functionality. Fine-grained components are generally combined with other fine-grained components to provide a large-grained component.

By working at the higher level of abstraction provided by large-grained components, it is possible to start working with business processes rather than having to deal with the inner workings of fine-grained components. Therefore, large-grained components have the potential to deliver greater productivity to developers than fine-grained components [Hurwitz 1998].

2.1.3. Components and Objects

Although both components and objects increase software reusability and simplify the software development process, they are not the same, but may be considered to be orthogonal concepts [Hernandez *et al.* 2000]. Components capture software's static nature, whereas objects capture its dynamic nature, i.e. components come to life through objects.

If one is using an object-oriented programming language to implement components, then in the simplest case, a component is simply a class, although in general a component would normally contain one or more classes or immutable prototype objects. However, for non-object oriented programming languages, a component could also be implemented using traditional procedures, functional programming constructs, assembly language, or any other approach [Szyperski, 2000].

In contrast to components (see 2.1.1.1), objects are units of instantiation that have a unique identity, they have state that can be persisted, and they encapsulate both state and behaviour [Szyperski, 2000]. Another significant difference between objects and components revolves around the use of inheritance [Hurwitz 1998], as the focus of component technology is not on inheritance, but the combination and integration of different software components [Fan *et al.* 2001]. For a more complete discussion on the distinction between components and objects, refer to [Leavens and Sitaraman 2000][Szyperski 1998][Szyperski 2000].

2.1.4. Component Interoperability

Interoperability is a major challenge for software developers in general, and within component-based software development in particular.

However, due to the development of standardised interfaces by the Open GIS Consortium, in their Abstract and Implementation Specifications, many these requirements for component interoperability will be easier to fulfil than in other application domains that have not undergone rigorous standardisation.

In the development of our RADGIS architecture we have not implemented automated component substitution, but have rather left the choice of component selection up to the application developer, or the end-user, who would be able to query a Trader or Directory Service and receive a list of equivalent components that suited his/her particular requirements. It would therefore be up to the component developer to ensure that a particular component fulfils its intended role.

There are three types of component interoperability [Hernandez *et al.* 2000]: signature interoperability, semantic interoperability and protocol interoperability. While the research in this thesis is not concerned with determining equivalence for compatibility and substitutability of components (see *figure 2.1*), we wish to briefly draw the reader's attention to these issues.

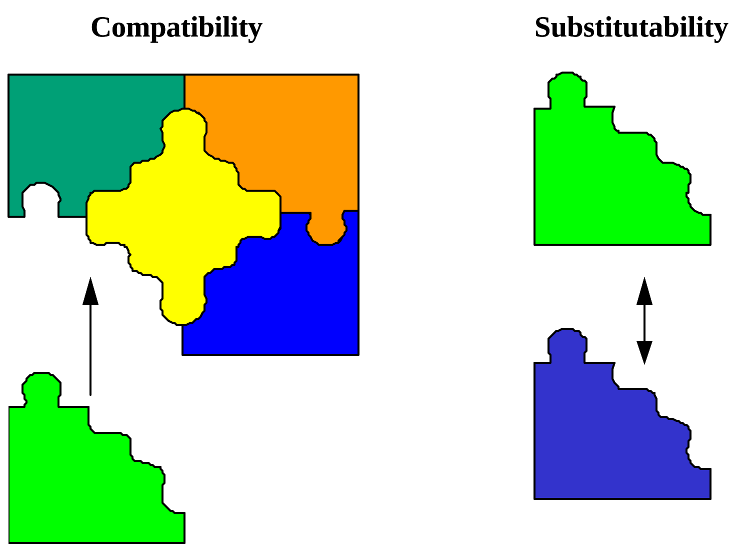


Figure 2.1. Compatibility versus Substitutability

2.1.4.1. Signature Interoperability

Component interoperability at the signature level is based on the names, parameter types and return types of the components' operations (methods), and is used for determining the compatibility and substitutability of components. The compatibility of two components is determined by their ability to work together properly if connected, i.e. determined by whether or not the data and messages exchanged between them are correctly understood.

Checking the substitutability of component A by component B, at the signature level, is an attempt to determine whether all the services offered by component A are also offered by component B. The only differences allowed, in the services offered by component B, are more specific inputs and more general outputs, i.e. it works on the principle of object subtyping.

However, without standardisation, there is no guarantee that the same naming conventions for methods, parameter types and return types, will be used by different vendors. There is also no guarantee that two methods that have equivalent signatures will perform the same operation, or handle the same boundary conditions equivalently, or perform the same algorithm with the same accuracy, and thus it is impossible to perform a complete comparison of two components simply on the signature level.

2.1.4.2. Semantic Interoperability

The term semantic interoperability was first introduced by [Heiler 1995]. In contrast to signature interoperability, which simply checks for compatibility and substitutability of components based on method signatures, semantic interoperability is an attempt to ensure that both requesting and providing components share a common understanding of the meaning of the requested services and data. Thus semantic interoperability includes ensuring agreement on, for example, the algorithms for computing the requested values, the side effects of methods, or the source or accuracy of requested data elements [Hernandez *et al.* 2000].

At the semantic level, compatibility is now determined by whether or not the behaviour provided by a component is the same as that required by the client component. This can be approached, for example, by showing that the pre-conditions of a component's methods are met by the calling components when invoking them, and that the methods' post-conditions satisfy the calling component's expectations.

Substitutability at the semantic level is based on "behavioural subtyping" [America 2000], and means that the behaviour of the subclass instances must be consistent with that of the superclass instances. This includes associating behaviours to signatures and identifying subtypes that conform to their supertypes both syntactically and semantically.

Computing semantic interoperability is undecidable in the formal sense. Therefore semantic interoperability is far more difficult to compute than signature interoperability, since it is not simply reliant on operational or behavioural semantics, but may also be dependent on the context in which the components are used. For a more in-depth look at semantic interoperability and behavioural subtyping, the reader is referred to works by [America 2000][Liskov and Wing 1994][Dhara and Leavens 1996].

2.1.4.3. Protocol Interoperability

The protocol interoperability level, first identified by [Yellin and Strom 1997], builds on top of the signature level and deals with the relative order in which an object expects its methods to be called, the order in which it invokes other objects' methods, and the blocking conditions and rules that govern the object interactions.

Two components are considered compatible at the protocol level if the restrictions imposed on the interaction of each component when they call each other are preserved and their communication is dead-lock free [Hernandez *et al.* 2000].

Substitutability at the protocol level is determined by two main issues [Canal *et al.* 2000] [Yellin and Strom 1997]:

?? All operations of component A are supported by component B, i.e. all messages accepted by component A are also accepted by component B, and component B's outgoing messages when implementing component A's services are a subset of component A's outgoing messages; and

?? The relative order of incoming and outgoing messages of both components is consistent.

2.1.5. Components and Scripting languages

Thus far we have introduced the concepts of components and frameworks, but have not indicated how components can be wired together to express compositions.

Scripting-languages have become increasingly popular for configuring and connecting components to develop small flexible applications quickly and easily. Indeed, according to [Szyperski 2000], "wiring" components is surprisingly productive for relatively simple applications.

Although these languages typically offer high-level abstractions for connecting components in a flexible manner, they generally support a single, specific architectural style, and are designed with a specific application domain in mind (e.g. graphical user interfaces – see XwingML, *section 3.2.1*) [Ousterhout 1998] [Schneider and Nierstrasz 1999].

2.1.6. Component Customisation and Re-use

In our discussion of CBSD up to this point, we have focussed on integrating COTS components directly into the component framework, and have given little consideration to the possibility of customising the component first. This is sometimes necessary to allow a component to better fit the requirements of a particular system in which it is to be integrated.

Conventional CBSD approaches do not generally support the ability to adapt, tailor and customise components in order to provide tight integration based on the user's specific requirements [Stojanovic 2000]. While this does not present as many problems for components that have been developed based on widely-accepted standards in a specific domain, and that have agreed-upon interface specifications, it does pose many problems for CBSD in general.

An inability to adapt, tailor and customise components means that components can only be reused in very specific cases, and reduces the ability to reuse components that could otherwise be reused. This results in a number of similar components that provide the same underlying functionality, but that have been rewritten for different applications, clearly defeating the goals of CBSD.

If one has access to the source code of a particular component, it is possible to modify components before reusing them. However, this is not generally possible or advisable because it increases the possibility of errors occurring in the component code, and very few vendors would be prepared to release the source code for their components. Therefore, the parameterisation and configuration of components provides an extremely valuable mechanism to facilitate the tailoring of a component according to application-specific requirements and the environment in which it is to be deployed. This also removes the requirement that the component integrator have access to the source code of the components. These two techniques increase the ability of a component to be reused in different applications and illustrate the spectrum of reuse possible, which Basili *et al.* [1994] characterise as: reused verbatim, slightly modified, extensively modified and new.

2.2. Expected Benefits of CBSD

The additional work required initially to develop reusable components is well worthwhile. The ability to rapidly assemble complex systems, based on well-defined interfaces, using existing, pre-tested components results in a number of quantitative and qualitative benefits, including [Fan *et al.* 2000] [Herzum and Sims 2000] [Szyperski 1998]:

?? shortened system development cycles;

?? increased productivity through reuse of software;

?? increased customisation;

?? increased performance;

?? increased maintainability;

?? improved reliability due to the use of higher quality components that have been reused in a number of different applications, reducing the probability that they have errors;

?? reduced time-to-market;

?? reduced development costs;

?? reduced maintenance costs; and

?? since components are encapsulated, it is possible to make changes to their implementation without affecting all the systems that rely on them.

It is because of these numerous benefits that GIS applications are slowly becoming disaggregated, component-based applications. As an extension to this paradigm, one of the core aims of this research is to provide GIS and Location Service integrators with an extensible architecture that will allow them to select, configure, customise and then deploy tightly integrated assemblies of (local and/or remote) components, based on standardised frameworks, rather than build whole systems from scratch.

2.3. Expected problems and limitations

While we have highlighted the many benefits to be derived from the use of CBSD, many of these benefits also add to the complexity of creating component-based applications, for example the necessity to be able to integrate components developed at different times, by different people, and possibly with different end-users in mind. One of the greatest potentials for problems to arise is that one cannot always foresee the incompatibilities that might arise when components are used in combination. Therefore one of the key issues for building applications from reusable components is that of interoperability.

There has been much debate about whether it is better to create very specific components with well-defined, reasonably-scoped functionality, or more flexible general-purpose components [OMG 1999]. More specific components are generally economic to design and reuse, whereas more abstract (generic) components can be reused in a wider range of applications, possibly across different application domains. However, the trade-off of generality is an increase in the cost of development, and the time taken to understand the additional requirements, document the component, as well as to test the component in a range of applications. For in-depth discussions on the merits of both approaches, please refer to [Niagara 2000] [Veryard 2000] and [Stojanovic 2000].

Another potential problem with the use of software components, especially distributed components that are integrated into the client application at runtime, is the issue of trust. The use or reuse of a software component entails an implicit trust relationship that the component will do what its specification says - nothing more and nothing less [Looney *et al.* 1998]. This emphasises the value of trademarks and brand names in the absence of secure mechanisms to ensure that the data that is being provided is accurate, and the services that are being invoked are providing the correct results.

Providing solutions to these problems fall outside the scope of this thesis, but is currently the subject of much research within the field of CBSD in general. This section has briefly outlined some of the potential issues that should be borne in mind when developing distributed component-based software in order to ensure that the reader is able to assess the merits of such an approach.

2.4. Components and Distributed Systems

IBM has recently announced plans to invest $4 billion to build 50 computer server farms around the world [Associated Press 2001]. They believe that pay-per-use, on-demand computing power and storage-capacity, will become a household commodity such as electricity. Although IBM are initially marketing the ability to obtain extremely large amounts of processing power for resource intensive applications, such as climate prediction, server farms may one day provide the ability for users and service providers to make use of them as temporary sites for processing intermediate results before returning the final result of a request that required services and data from a number of disparate locations.

Thus it is no longer necessary for a complex computational operation to be handled by a single service provider, or for the originating client to process intermediate results. The client may issue a set of requests (similar to the idea of an itinerary for mobile agents, or service chaining) which access remote services and data, and all the intermediate results are processed (in a location transparent manner) on a server farm.

This model of operation may be a little "futuristic" at the moment, but still emphasises the usefulness of distributed object technology that allows one to implement distributed services. These services may one day make use of processing power obtained from the above-mentioned server farms in a location transparent manner, which if implemented correctly, would not require any change to client applications.

Distributed object technologies enable the invocation of methods on distributed objects residing anywhere on a network, possibly running on heterogeneous platforms and operating systems, as if they were local objects. With the development of "bridging" technologies, e.g. CORBA's IDL mappings, it is also possible for clients to access distributed objects irrespective of the programming language and compiler used to create the distributed objects.

Distributed object technology encompasses not only the original object-oriented model, but also component technology [Fan *et al.* 2000]. Unlike objects that execute locally as part of a client-application, distributed objects live within their own dynamic library outside of an application, and are considered to be components because of the way they are packaged [Asaipillai 1997]. This distinction is made clearer when one considers the definition of a software component by Szyperski [1998] as *"a unit of composition with contractually specified interfaces and explicit context dependencies that can be deployed independently and is subject to third-party composition"*.

The use of distributed object computing provides a number of solutions to problems associated with existing monolithic applications, including [Fan *et al.* 2000] [Fingar *et al.* 1997]:

?? The ability to share (expensive) resources;

?? Platform and operating system independence, which allows one to distribute components of an application to computing platforms that best fit the task of each object;

?? The ability to leverage legacy code as part of new applications, or on different platforms;

?? The ability to use the programming language best suited for a particular task;

?? Increased efficiency, because the workload can be distributed across multiple machines;

?? Applications can be deployed internally on a company's intranet or globally across the Internet; and

?? The same distributed resources can be made use of by a number of different devices such as desktops, workstations, PDAs, and mobile phones. This means that one no longer needs to implement the same business logic multiple times for use by different devices.

Thus many organisations are beginning to implement component-based n-tier applications based on popular distributed object technologies such as CORBA, Enterprise JavaBeans (EJB), and DCOM [Morais 2000]. We have chosen to restrict our research to the use of CORBA objects and EJBs because both of them are platform independent solutions that provide tight integration with the Java programming language (reasons why we chose to use Java, based on the RADGIS application's need to integrate a number of diverse emerging technologies, are given throughout the thesis).

2.4.1. CORBA

The Common Object Request Broker Architecture (CORBA) is a distributed, object-oriented, middleware specification developed by the Object Management Group (OMG) that aims to facilitate portability and interoperability of objects across heterogeneous networks [Fan *et al.* 2000]. Possibly the biggest advantage that CORBA has over other distributed object technologies such as EJB's and DCOM is that the programming language and compiler used to create the server objects may be different from the programming language used to implement the client objects, thus providing programming language independence. In addition, the location of distributed CORBA objects as well as the specific platform and operating system they execute on, are totally transparent to clients. CORBA therefore provides an ideal mechanism for creating 3-tier (or n-tier) distributed applications, which goes beyond providing simple interoperability [Orfali and Harkey 1998].

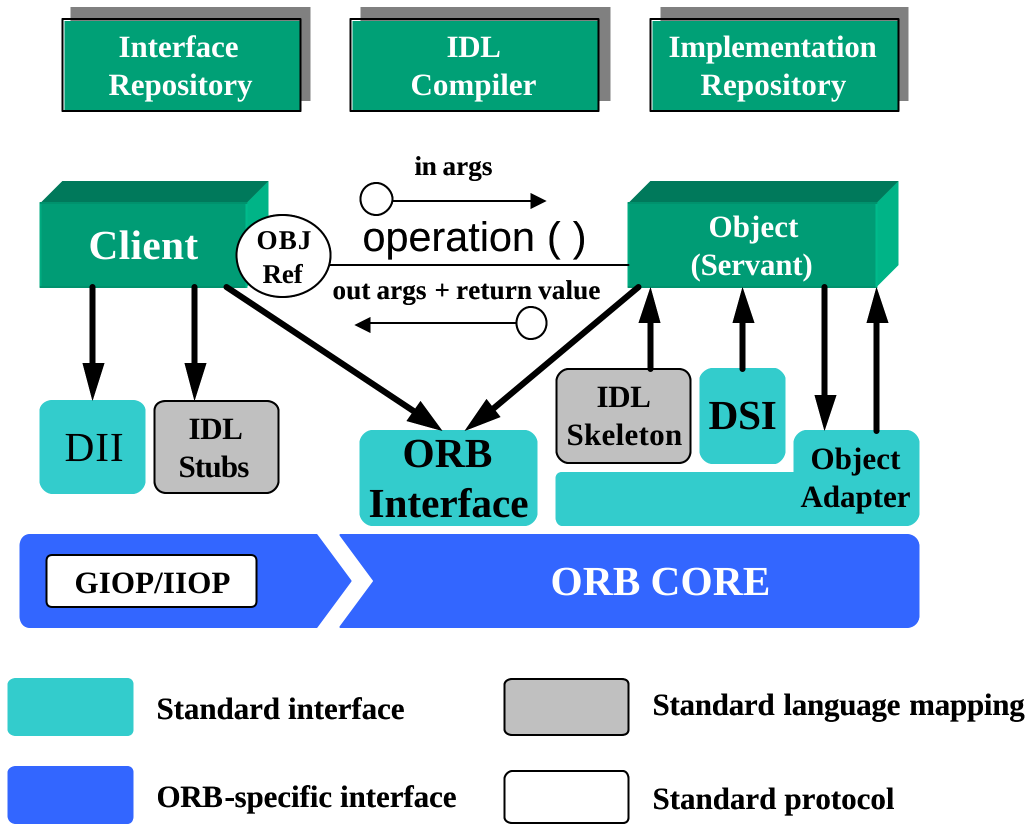


Figure 2.2. CORBA ORB Architecture [Schmidt 2001]

Five of the main components of CORBA are [Asaipillai 1997] [Fan *et al.* 2000] [Orfali and Harkey 1998]:

?? The Object Request Broker (ORB) – ORBs are the core of the CORBA specification. They intercede on the client and server objects' behalves, handling the flow of messages between the client application and the distributed object(s) using the General Inter-ORB Protocol (GIOP), the Environment-Specific Inter-ORB Protocols

(ESIOPs) for interoperation over specific networks, or most often the Internet Inter-ORB Protocol (IIOP).

?? The Interface Definition Language (IDL) – is a programming language independent, declarative language used to define the services provided by objects independent of their implementation, i.e. the IDL is used to specify the modules, interfaces, data types, methods, argument types and return types of the distributed objects.

?? The Interface Repository (IR) – is a searchable, persistent storage mechanism for IDL interface declarations. It allows client applications to navigate an object's inheritance hierarchy and provides descriptions of all operations that an object supports as well as type information necessary for issuing requests using the Dynamic Invocation Interface.

?? The Dynamic Invocation Interface (DII) – is a generic client-side stub capable of forwarding any request to any object, using runtime interpretation of request parameters and operation identifiers to perform the marshalling and unmarshalling of requests. The DII is therefore often used together with information obtained from the IR, and is extremely useful for making use of distributed objects for which compile-time knowledge of their interfaces was unknown[1].

?? Object Adapters (OA) – act as an interface between various object implementations and the ORB, providing services such as the generation and interpretation of object references, mapping object references to implementations, object method invocation, and the activation and deactivation of objects and implementations (see *figure 2.3*)

Of particular interest to us, with respect to the integration of GIS services into the RADGIS client at runtime, is CORBA's capability to perform dynamic discovery of objects and services, because CORBA objects are self-describing and introspective. CORBA's dynamic facilities, including the Trader Service, Dynamic Invocation Interface (DII), and the Interface Repository, allow the creation of extremely flexible systems that allow runtime discovery and late-binding [Orfali and Harkey 1998]. This is especially useful in the distributed Web

[1] This is in contrast to the normal mode of operation whereby IDL declarations are compiled into programming language-specific stubs, allowing clients to invoke operations on known objects.

environment where a user is able to discover new services and then make use of them transparently.

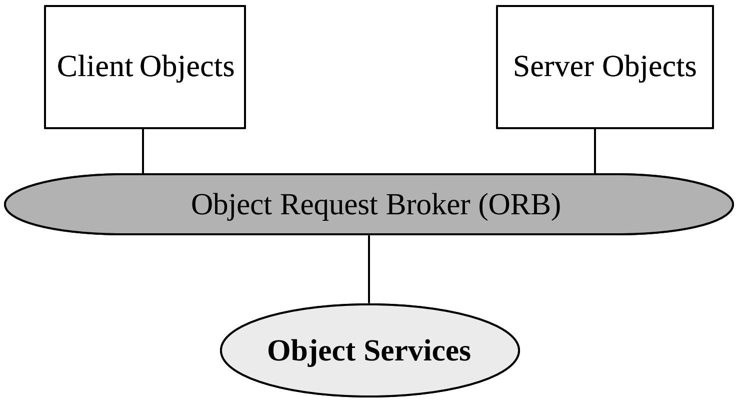


Figure 2.3. Simplified Object Management Architecture

2.4.1.1. CORBA Trader Service

The CORBA Trader Service offers a brokerage facility, i.e. it allows objects to publicise their services and bid for jobs. Clients may specify queries in a formalised constraint language, and the trader responds by providing zero, one or more matching CORBA object references that the client can use to bind to a server object of its choice. Traders are particularly useful in systems where there are many objects offering the same service. In this case it becomes impractical to insist that every object be uniquely identified by name [Resnick 1996]. The Trader Service therefore provides a mechanism to differentiate services implemented as CORBA objects by their properties, e.g. cost, implementation algorithm, platform on which it is running, type of input data set, and output type.

Currently, the CORBA Trader specification supports only passive self-registration of objects with the Trader Service. However, it is not unlikely that in the future, the trader will be able to actively seek out server objects in a manner similar to that of current Web search engines. Orfali and Harkey [1998] and Resnick [1996] describe an Object Web environment containing spiders, crawlers, and bots, that will dynamically discover CORBA objects and store them in up-to-date Traders.

2.4.1.2. CORBA Component Model

The vendor-neutral CORBA Component Model (CCM) forms part of the CORBA 3 Specification developed by the OMG. It is an enhancement to the current, widely-adopted CORBA 2 Specification that extends the basic architecture defined in the EJB Specification, providing a framework for building, assembling and deploying components, referred to as CORBAcomponents [ScreamingMedia 1999].

The CCM specification defines two level of component: basic and extended [Pharoah *et al*. 2000]. Basic components have the same capabilities as EJBs (as defined by release 1.1 of the EJB Specification), and interoperability between basic CCM components and EJBs is possible using the IIOP protocol. This also means that it will be possible to seamlessly access EJBs using CORBA CCM clients implemented in a wide range programming languages, including Java, COBOL and C++.

According to Jon Seigel [2001], the 3 major features of CORBAComponents are:

?? A *container environment* that provides support for transactions, security, and persistence, and as well as interface and event resolution.

?? *Integration with Enterprise JavaBeans.* It will be possible for Enterprise JavaBeans (EJBs) to act as CORBAcomponents, and therefore EJBs can also be deployed in a CORBAcomponent container. The advantage of using CORBAcomponents, as opposed to using EJBs, is that they can be written in multiple languages and support multiple interfaces.

?? A multi-platform *software distribution format*, including an installer and XML-based configuration tool.

While we have not made use of the CORBA Component Model in our research, because an implementation of this standard was not yet available, it holds much potential for the development of distributed applications, which are composed of components that are independent of implementation language and platform.

2.4.2. Enterprise JavaBeans

The Enterprise JavaBeans (EJB) standard defines a component architecture for deploying components called enterprise beans, in an EJB container. The EJB container provides runtime services to the components, such as life-cycle management, threading, transaction support, security and persistence. Thus the Enterprise JavaBeans model is simpler than the CORBA model because the EJB container is responsible for the provision of these services [Sun Microsystems 2001b]. (Hence the introduction of the CORBA Component Model, see *section* 2.4.1.2).

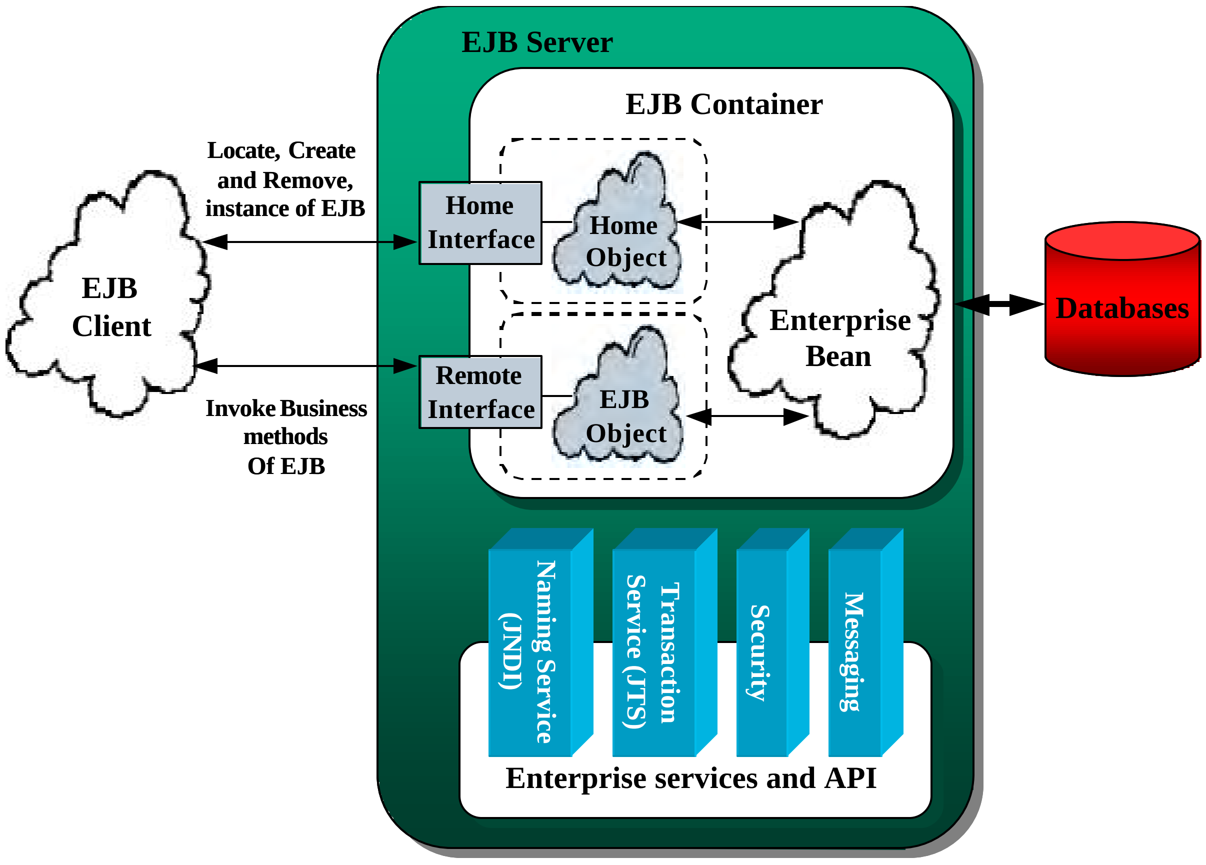


Figure 2.4. The basic EJB Model [Raj 1998]

An enterprise bean is never accessed directly by the client, but rather through the EJB container, which intercepts the method calls and provides the required services. This means that components are implemented without the complexity of explicit middleware code required to implement these services. Instead, should transaction support or security be required by a particular component, they could be specified declaratively at the method level at deployment time.

In contrast to CORBA, which is implementation language independent, EJBs are based on the Java programming language. However, just like CORBA, EJBs can be deployed on any platform and operating system that supports the EJB standard. Interoperability between EJBs and CORBA objects is made possible through the use of the Internet Inter-ORB Protocol (IIOP), which is now supported by Java RMI. This is particularly useful for allowing non-Java clients to make use of EJBs.

The EJB architecture therefore simplifies the development of highly scalable, distributed, component-based, enterprise systems that have strong transaction and security support [Fan *et al.* 2000].

2.5. Summary

In this chapter we have provided some background information on CBSD issues that were taken into consideration when building our RADGIS architecture and specifically in the development of the DGCML meta language for creating and deploying GIS services based on local and distributed components.

It provided insight into issues of compatibility and substitutability of components, as well as factors affecting the reuse of components. The overriding benefits of CBSD as well as the inherent problems and limitations of this approach were discussed, and once again we stressed the need for interoperability, which is best achieved through the specification of and rigid adherence to standards.

"Stand-alone" GIS services (large-grained components) require very little integration with the client application. This is advantageous as it reduces potential complexity for the end-user, who would generally not have any programming skills, who wishes to "tightly" integrate new GIS services with the client application. However, it is also possible to simplify the integration of components that have dependencies on other components or on the client application if both the client and the component(s) conform to agreed-upon standards, such as the OpenGIS® Specification defined by the OGC.

Two distributed object technologies used in the development of our proof of concept RADGIS client were discussed briefly. Both the CORBA and EJB specifications are fairly similar. CORBA is an open standard that provides programming language independence, whereas the EJB specification is Java specific and was developed as part of the Java Community Process. The most important feature of these distributed object technologies is the ability for client applications to make use of distributed objects as though they were local. This in turn allows a large degree of location transparency when working with distributed resources, increasing the efficiency and flexibility of applications while reducing unnecessary complexity through the use of high-level distributed object programming constructs.

Thus we are now armed with an understanding of CBSD and the additional functionality provided by distributed object technologies. This will allow us to keep in mind the requirements that must be considered when developing the components and the component framework that are to become the extensible distributed GIS based on interoperable services.

Chapter 3

XML - Enabling CBSD and Deployment

"Make everything as simple as possible, but not simpler."

Albert Einstein (1879-1955)

This chapter will introduce some background information about XML in order to demonstrate how we have used it to develop a wiring language, called DGCML, which can be used in the development and deployment of component-based software. It also provides a description of what DGCML has been designed to achieve and, together with *section 2.4* on distributed component-based software development, provides our approach to generating applications dynamically at runtime.

Although the use of DGCML to create applications by wiring together components is a generic approach, which is not restricted to the GIS domain, there are a number of problems with trying to integrate components that have been developed by different vendors, possibly using different naming conventions. Therefore this approach will be most successful when used in a specific application domain, such as GIS, in which there has been a tremendous focus on standardisation and interoperability. This is the focus of the next chapter in which we explain how the development of our RADGIS application was influenced by different GIS architectures and the standardisation efforts undertaken by the Open GIS Consortium.

3.1. XML Basics

The Extensible Markup Language (XML) [Bray 1998], developed by the World Wide Web Consortium (W3C), is a standard for encoding data in plain text (i.e. not a binary format) that has seen rapid adoption by industry. It is a subset of the Standard Generalised Markup Language (SGML) that was specifically designed for ease of implementation, and for interoperability with both SGML and HTML.

While there has been a lot of hype surrounding the usefulness of XML (see [Kamthan 2000]), it does have a number of tangible benefits that make it an extremely valuable tool for developing distributed applications where high levels of interoperability are required. These are primarily due to the nature of XML which is self-describing, can accommodate both document and data structures, and allows one to define customized markup languages. Other important advantages include [Software AG (*current*)] [St. Laurent 1999]:

?? Clear separation of content from presentation. XML is not a presentation grammar – the tags in an XML document provide contextual information that can be used to interpret the meaning of the data. However, an XML document can be displayed in any number of different ways by simply applying different stylesheets to it.

?? Providing portable data. XML-encoded documents do not require a (vendor-) specific application for viewing or editing the data.

?? Support for multilingual documents and Unicode. This is an important consideration for the internationalisation of applications.

?? Ability to embed multiple data types. There is no limitation on the type of data that can be stored in an XML document. Thus it is possible for XML documents to contain a wide range of possible data types, from multimedia data (image, audio, video) to active components (Java applets, ActiveX).

?? Separation of file handling from application architecture. The parsing of XML documents tends to be assigned to a pre-built component, leaving the application developer to concentrate on processing the data.

?? Clean integration with OOP. The hierarchical structures of XML map well to objects and properties. This is particularly true of XML documents based on the XML Schema language, which supports data-types and namespaces. However, one must be aware of mapping issues such as when to aggregate elements or when to use references, and how to represent references between items.

?? Program composition based on document content. Mapping XML content to object structures, together with late-binding permits the runtime construction of programs based on XML document content. This has been shown to be useful for simplifying repetitive programming chores like GUI construction. (See relevant sections on BeanML, XwingML and our DGCML)

However, there are also some disadvantages associated with the use of XML. These include:

?? the verbose nature of XML – because XML is a plain text markup language, and there is a lot of structural/meta-information associated with XML documents, the XML-encoded information is less efficient than binary formats in terms of size and performance. The larger XML representations also impact on bandwidth requirements when transferring XML documents across a network/the Internet.

?? (tongue firmly in cheek) not being able to lock customers into a proprietary software solution and inscrutable file format. However, this is clearly not an issue if one is committed to the development of open systems.

The issue regarding the performance of XML, as a text-based representation as opposed to a binary representation, will not severely limit XML's usefulness and application as disk space, network bandwidth, and CPU's are constantly getting cheaper/faster. In addition, the use of compression techniques, such as XMLZip, allows for a significant reduction in the size of XML-encoded data. Compression techniques can also be used effectively in the end-to-end transmission of XML data, for example using HTTP/1.1, which can compress data on the fly, thus saving bandwidth as effectively as a binary format. Alternatively, if one makes use of the XML DOM API, XML files can be compressed based on the node level in the XML document. This allows the XML file to be uncompressed on the client side according to the specific node the user is referencing, rather than uncompressing the entire document [Kamthan 2000].

3.1.1. DTD versus Schema

A DTD is a set of rules, written using the DTD language, that specify which elements are allowed in an XML document, the order in which they can appear, and which of the elements have attributes [Maler 2000]. The DTD language, however, has two major short-comings: the lack of datatypes, and the inability to support the use of namespaces [Radiya and Dixit 2000].

The XML Schema specification, which was developed subsequent to the DTD, is a much richer and more extensible way to describe the rules for the content of a document than using a DTD. The benefits of using the XML Schema language as opposed to the DTD language include:

?? XML schema are written in XML as opposed to requiring the user to learn a separate XML Schema language. Therefore XML schema may be edited and processed with the same tools as XML documents. In contrast, DTD's are written in a non-XML, DTD-specific language.

?? The XML Schema language supports the definition of data types, the specification of numeric ranges, sets (not possible in a DTD), regular expressions and checks on text content. It also allow element content to be specified as unique through the use of keys.

?? Multiple elements may be defined with the same name but different content.

?? The XML Schema language provides support for Namespaces [Bray *et al.* 1999], which makes the reuse of entire XML vocabularies and/or individual structure definitions easier. (Although the development of the XML Namespace specification was carried out independently of the XML Schema specification, it is not possible to make use of XML Namespaces using the DTD language.)

Many of the benefits listed above are a direct result of the XML Schema having much in common with programming languages, e.g. object-reuse through inheritance, the creation of user-defined types, and namespace scoping. Despite the complexity introduced by the XML Schema, the overwhelming benefits listed above ensured that after a lengthy debate on the merits of the DTD versus XML Schema specifications, the W3C formally accepted the XML Schema specification as a W3C Recommendation on 1 May 2001 [Whitlock 2001].

At the time of writing this thesis, our DGCML grammar (see *section 3.4* and *section 5.2.2*) was specified using a Document Type Definition (DTD) file. While the advantages of using the XML Schema, as well as namespaces and linking, will provide additional flexibility to DGCML, there is currently no compelling reason, or additional functionality required that can only be achieved by using XML Schema. Therefore, although we will probably convert our DTD to an XML Schema representation at a later stage, once more mature products and support for this new standard become available, it currently adds little value to our proof-of-concept system.

3.1.2. SAX versus DOM

Two dominant standards exist for XML processing, namely the Simple API for XML (SAX) developed by members of the XML-DEV mailing-list, under the coordination of David Megginson [Megginson 2000], and the Document Object Model (DOM) which is currently being developed as a W3C specification [W3C DOM WG 2001].

SAX is an event-driven model that "reads" the document, and is supported by almost all Java XML parsers [St.Laurent 1999]. The module using the SAX reader supplies "callback" methods that are invoked as the SAX parser encounters tags, elements and properties in the XML document. It is up to the developer to supply the implementation of the "callback" methods to perform the desired operation, for example, to build an in-memory structure corresponding to the document.

SAX requires more programming that the DOM, and is limited in processing functionality in comparison with the DOM because the parser only knows about the current node, and can only process nodes that are children of the current node, i.e. it does not inherently provide a mechanism to access information about previous nodes. However, it is faster and more memory efficient than the DOM as it does not store the entire document model in memory at one time.

DOM, in contrast, is a tree-based model of the document, i.e. a DOM parser creates a hierarchical tree representing the entire data structure in memory, and passes this representation back to the application for manipulation as a Document Object. This can obviously cause problems when the document is large. However, this is a trade-off against the ability to perform more sophisticated processing than would be available using the SAX [Cagle 2000].

Our DCGML interpreter currently makes use of the DOM parser (as does BeanML), although it could just as easily have been implemented using a SAX parser because the DGCML interpreter simply reads the document from start to finish, building the GUI, and keeping hashtables of components and objects that may be referenced later in the document, or by components in a DGCML descriptor of another service, i.e. it is a single-pass interpreter. The size of the DGCML files is relatively small because they are intended to be easy to read and modify. Complex operations, which might otherwise cause the DGCML file to become unwieldy, would normally encapsulate some programming concept that would be best implemented as a Java class. That Java class would then be referenced in the DGCML descriptor to provide the desired functionality, simply and efficiently. Thus the performance issues associated with keeping the entire document in memory using the DOM parser are negligible.

3.2. XML for Wiring Components

The use of XML has not been limited to simply formatting documents, but has also been used to create vocabularies that allow one to compose applications with XML by wiring together components. One example is IBM's BeanML, which allows one to wire JavaBeans, and create GUI's by wiring together GUI components. Another example is XwingML, which allows the specification of GUI's composed of Swing components.

While both of these wiring languages were discovered subsequent to the development of our DGCML wiring language, they confirm the validity of our approach. Feature comparisons show the strengths of our language, which provides a superset of the operations available in these competing technologies. This is because DGCML was designed with three main goals in mind, the ability to *deploy services composed* of *local and* *remote* *components*, together with an *easily-configurable GUI* and an *easily-integrated help* system.

3.2.1. XwingML[*]

XwingML (pronounced "zwing-M-L"), developed by HP Bluestone Software, enables users to build XML documents that define a complete Java Swing GUI [BlueStone 1999]. Although XwingML is similar to BeanML (see *section 3.2.2*), it is a more specific vocabulary aimed at a particular area of Java development, i.e. GUI development, as opposed to BeanML, which is a more general vocabulary for wiring JavaBeans [StLaurent 1999].

A simple example of XwingML, taken from [Bluestone 2000], illustrates the sort of information stored in a XwingML descriptor file.

```
<JMenuBar>
    <JMenu text="File" mnemonic="F">
        <JMenuItem icon="open.gif" text="Open"
            actionListener="OpenFile"/>
        <JMenuItem icon="save.gif" text="Save"
            actionCommand="save"
            actionListener="SaveFile"/>
        <JMenuItem text="Exit"
            actionListener="com.bluestone.xml.swing.XwingMLExit"/>
    </JMenu>
</JMenuBar>
```

XwingML comes with an XML DTD that defines the full set of Swing/Java Foundation Class (JFC) classes and properties, as well as support for all Swing/JFC Listeners [BlueStone 1999]. It also includes templates for making a wide variety of GUI interfaces, including menus, frames, and dialog boxes [WebTechniques 1999]. The XwingML allows GUI's to be easily specified by simply editing an XML document. The XML document is then read in by the XwingML parser, which dynamically generates the Java GUI.

[*] XwingML is no longer supported/available for download from BlueStone

Specifying the GUI of an application separately as an XML document has a number of benefits, including [BlueStone 1999] [WebTechniques 1999]:

?? Java GUI creation without Java coding or compile cycle,

?? Java GUI defined in human-readable XML, which is closer to plain English and take less time to learn than Java,

?? Ease of code maintenance and re-use because there is a clear separation of the GUI code from the application logic.

XwingML's focus on representing Java Swing GUIs has enabled it to provide a fairly easy to use syntax for specifying relatively complex Swing GUIs. However, it is limited to using Swing components, and does not allow one to specify more general operations, such as working with non-GUI objects, as possible with BeanML and our DGCML. If the XwingML syntax were to be extended to allow the more generic operations provided by BeanML and DGCML, it would require the specification of a very detailed DTD or schema because of the use of element and attribute tags to represent the name and properties of each class.

3.2.2. Bean Markup Language (BeanML)

IBM Alphaworks' Bean Markup Language (BeanML) is an XML-based component configuration or wiring language that is customised for the JavaBean component model. It is not a full scripting language, but instead serves only to describe how JavaBean components relate to one another in terms of their configuration.

A simple example of BeanML, illustrates the sort of information stored in a BeanML descriptor file[1].

```
<bean class="javax.swing.JMenuBar">
  <add>
    <bean class="javax.swing.JMenu">
```

[1] In order to allow a syntactic comparison of BeanML, XwingML and DGCML, the simple code extracts implement the same trivial task of setting up part of a menu. The code extracts should not, however, be considered a reflection on the power and flexibility of the wiring languages.

```
              <args> <string value="File"/> </args>
              <property name="mnemonic" value="F"/>
              <add>
                <bean class="javax.swing.JMenuItem">
                  <args> <string value="Open"/> </args>
                  <property name="icon" value="open.gif"/>
                  <property name="mnemonic" value="O"/>
                  <event-binding name="action">
                    <script>
                        … open file dialog, etc.
                    </script>
                  </event-binding>
                </bean>
                <bean class="javax.swing.JMenuItem">
                  <args> <string value="Save"/> </args>
                  … Additional details for Save MenuItem omitted for brevity …
                </bean>
                <bean class="javax.swing.JMenuItem">
                  <property name="text" value="Exit"/>
                  <property name="mnemonic" value="x"/>
                  <event-binding name="action">
                    <script>
                        … tidy up, and exit gracefully
                    </script>
                  </event-binding>
                </bean>
              </add>
            </bean>
         </add>
   </bean>
```

There are many conceptual similarities between BeanML and our DGCML, as they both attempt to facilitate similar behaviour. However, there are also some fundamental differences, the most important of which is that BeanML does not support distributed application development. In addition, BeanML is a very general solution to wiring beans together, whereas our DGCML was built specifically for marking up descriptions of distributed GIS services, including deployment information, a description of the service, its GUI, and its associated help system.

BeanML is extensible[1], and is a more general solution for developing applications declaratively than our system. While it may have been possible to extend/customise BeanML to provide similar functionality, we have kept our original design rather than adopting and extending the BeanML approach in order to avoid the complexity of BeanML due to its generalised approach. Our system was developed to solve the specific problems associated with the development of a dynamic interface for GIS components, based on the inclusion of remote services implemented as EJB or CORBA Objects. In addition, one of the requirements was that it should be simple to modify, so that non-programmers could understand, and customise, any part of the GIS service descriptor by simply editing the XML text.

3.3. XML for deploying applications

With the rapid growth of the Internet, there can be little doubt that we are moving away from the standalone, or "unconnected," model of operation, and migrating towards a highly networked environment. This has resulted in the network becoming a powerful medium for software distribution. No longer does the distribution of software have to incur the cost overhead of producing CDs or floppy disks. Instead, wherever possible the network may be used to allow end-users to download software. This may be done actively using the "*pull*" paradigm, or passively though the "*push*" paradigm [van Hoff *et al.* 1997]. This clearly has an impact on how software is deployed and requires new deployment software technologies to be developed that support this new distribution medium [Hall *et al.* 1999].

Part of the functionality provided by our DGCML is the deployment of GIS services that may execute locally on the client machine, or remotely on a server, or as a combination of local and remote components. Therefore we have included a brief overview of two popular deployment mechanisms to illustrate the similarities in the information stored by each of them as well as the information stored in our DGCML descriptor file.

[1] BeanML allows one to develop special classes for event-handling, performing type conversion, and to overload the <add> operation for adding different beans.

3.3.1. OSD

The Open Software Description (OSD) specification is an XML-based, open industry data format for the automation of software distribution over the Internet or corporate intranets. It has been included here because there is overlap between the functionality that has been included in our DGCML wiring language, and information stored about software components/applications deployed using the OSD specification.

The OSD vocabulary is used to describe deployment-related information about software packages and their inter-dependencies, and can be used to deploy Java packages, Java standalone applications and platform native code [van Hoff *et al.* 1997]. Deployment occurs when the client, usually a browser, parses the OSD file and then downloads and installs the necessary components.

OSD was discovered when our DGCML design was already in an advanced stage. It has capabilities for identifying metadata beyond those in our DGCML description, such as operating system and version, processor, and language requirements as well as dependency information. Incorporating these capabilities into DGCML would be beneficial if our language were to become more than simply a mechanism to implement our proof-of-concept system RADGIS. However, DGCML could easily be extended to include such metadata without affecting its core structure.

The OSD specification provides an XML-encoded vocabulary that can be used to describe software components, their versions, their underlying structure, and their interdependencies with other components. Our DGCML also contains deployment information, but is aimed at finer-grained components that facilitate the development and deployment of GIS services. In addition, DGCML specifies the GUI for invoking the components as well as information necessary to integrate help for that service into the client application's help system.

A simple example of OSD, taken from a specification submitted to the W3C (see [van Hoff *et al*. 1997]), illustrates the sort of information stored in an OSD descriptor file.

```
<SOFTPKG NAME="com.foobar.www.Solitaire" VERSION="1,0,0,0">
    <TITLE>Solitaire</TITLE>
      <ABSTRACT>Solitaire by FooBar Corporation</ABSTRACT>
        <LICENSE HREF="http://www.foobar.com/solitaire/license.html" />
        <!-- FooBar Solitaire is implemented in native code for Win32, Java
             code for other platforms -->
        <IMPLEMENTATION>
            <OS VALUE="WinNT"><OSVERSION VALUE="4,0,0,0"/></OS>
            <OS VALUE="Win95"/>
            <PROCESSOR VALUE="x86" />
            <LANGUAGE VALUE="en" />
            <CODEBASE HREF="http://www.foobar.org/solitaire.cab" />
        </IMPLEMENTATION>

        <IMPLEMENTATION>
            <IMPLTYPE VALUE="Java" />
            <CODEBASE HREF="http://www.foobar.org/solitaire.jar" />

            <!-- The Java implementation needs the DeckOfCards object -->
            <DEPENDENCY>
                <CODEBASE HREF="http://www.foobar.org/cards.osd" />
            </DEPENDENCY>
        </IMPLEMENTATION>
</SOFTPKG>
```

While most of the above example is fairly self-explanatory, it is worth pointing out that the Solitaire software package specifies that there are two different implementations of Solitaire available. The first version is packaged in a standard windows CAB file, and requires Windows 95 or WinNT version 4.0 to run. The second version is a Java version, which does not have particular operating system requirements (for obvious reasons), but has a dependency on another piece of software whose deployment is specified by the cards.osd file. Should the software specified in the cards.osd file not be installed on the local machine at the time of installing the Java version of solitaire, it will be installed before the solitaire installation begins.

3.3.2. JNLP and Java Web Start

Due to the overwhelming emphasis being placed on making everything Web browser-enabled, there has been a strong drive by Sun to develop Java-based server-side technologies, as seen in the development of the Servlet and J2EE specifications, while the client-side has remained relatively neglected. In fact, according to [Rohaly 2000] *"it's been accepted for some time now that client-side Java is dead"*.

Many people are slowly realising that trying to make many different types of applications fit the Web browser style is not always possible or prudent, particularly due to browser limitations such as the limited graphics capabilities, relatively primitive GUI, and incompatibilities between different browsers and versions. Thus there is still a very real demand for client-side applications, as opposed to browser-based front-ends, that interact with server-side processes.

It was the simplicity of the web-browser and the built-in security that have made it such a popular development platform. However, with the help of the recently launched Java Web Start and Java Network Launch Protocol (JNLP), these same benefits, together with a number of others mentioned below, may yet ensure that Java becomes a viable client development platform.

Java Web Start, developed by Sun and its partners under the Java Community Process, is a client-side helper application that is invoked when a browser encounters a link with a MIME type application/jnlp and file extension .JNLP [Rohaly 2000]. Clients therefore no longer require a browser with a JVM in order to run these applications or applets. Instead, the Java Web Start Application Manager can be used to launch the applications or applets (and perhaps launch an external JVM if required), install the applications so that they may be executed through icons on the desktop, or even launch these applications when offline.

This approach allows one to launch client-side applications that provide richer functionality than HTML, without the need for a browser or applets. In addition, the use of JNLP provides a number of runtime features that make JNLP-based apps more attractive than applets, such as [Rohaly 2000] [Sun MicroSystems 2001a] [Liron 2000]:

?? guaranteeing platform compatibility by automatically detecting, installing, and using the correct version of the Java Runtime Environment for a particular application.

?? providing the ability to launch applications or applets from the browser or the desktop. (applet integration is built into Java Web Start, allowing existing applets to be deployed without modification.)

?? version checking ensures that newer versions of the application are downloaded as they become available.

?? classes used by the application are automatically cached and updated locally so that the start-up time is dramatically reduced after the first time the application is used.

?? enhanced security, which goes beyond the Applet sandbox model, ensures that local resources, such as the file-system, may be used in a secure manner without the need for signed code. However, it is also possible to grant additional permissions to signed code.

?? the ability to finely control how applications are downloaded. For example, it is possible to load one small JAR immediately, and the others on demand.

?? A *jardiff* mechanism facilitates incremental updates such that only the classes in a JAR that differ from the locally-cached copies are downloaded. Thus updates to an application need not require the entire new version to be downloaded.

Once again, the best way to illustrate what sort of information is stored in a .JNLP file is to provide a simple example. The .JNLP example below is Sun's "Draw" example application, available at http://java.sun.com/products/javawebstart/demos.html:

```
<jnlp>
      <information>
            <title>Draw</title>
            <vendor>Sun Microsystems, Inc.</vendor>
            <description>Draw</description>
            <description kind="short">A minimalist drawing Application
            along the lines of Illustrator.</description>
            <icon ref="http://www.swingteam.com/jumpjars/draw.jpg"/>
            <offline/>
      </information>
      <jre version="1.3.0 1.3 1.2"/>
      <codebase>
            <jar ref="http://www.swingteam.com/jumpjars/draw.jar"/>
      </codebase>
      <application mainclass="Draw"/>
</jnlp>
```

The .JNLP descriptor file is read by the Java Web Start Application Manager which then decides whether the application needs to be downloaded or updated, then runs the application. *Figure 3.1* provides a pictorial overview of the entire process, from selecting a link to a .JNLP file, to the launching of the application once all the necessary checks have been made and files downloaded.

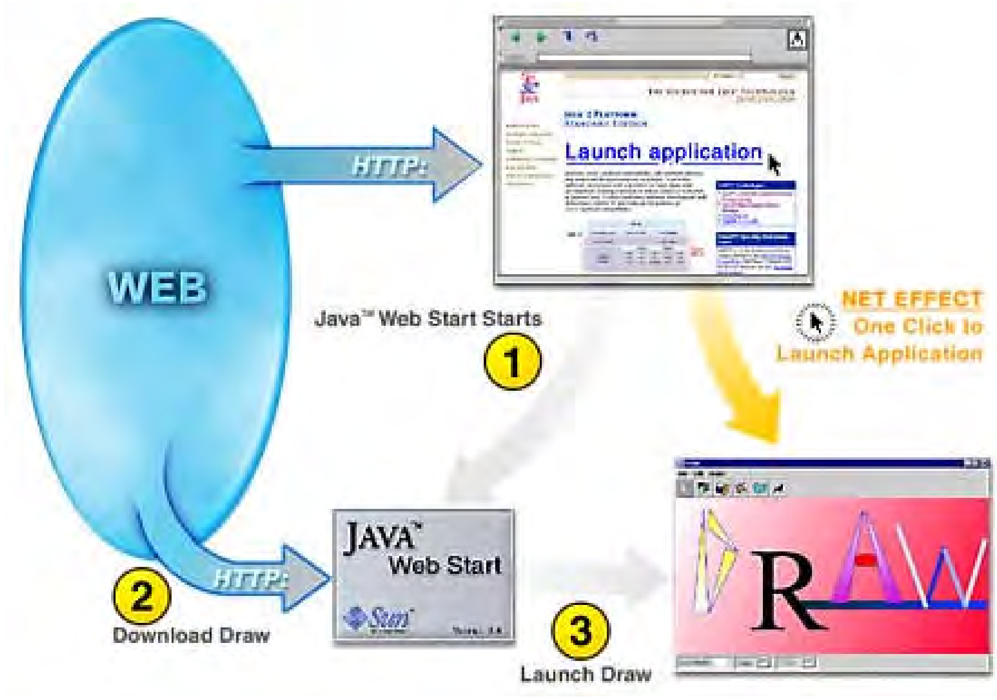


Figure 3.1. The Jav a Web Start architecture [Rohaly 2000]

3.4. Our Distributed GIS Component Markup Language (DGCML)

In the development of our proof-of-concept system, we have made use XML to transfer metadata about the implementations of individual GIS services. DGCML allows the developer of GIS services to make use of local and/or remote objects, and contains a description of the service, what components make up its GUI (if there is one associated with that service) as well as links to the necessary help files for that service[1]. DGCML has been designed specifically to be easy to understand and edit, and its programming style is close to Java.

The extract of DGCML below, based on the example used in XwingML and BeanML, illustrates the sort of information stored in a DGCML descriptor file

```
       … deployment information …
  <GUI>
       … other GUI components …
     <Component name="defaultMenuBar" type="MenuBar">
          <Component name="fileMenu" type="Menu">
              <Property name="text" value="File"/>

              <Component name="OpenMI" type="MenuItem">
                <Property name="text" value="Open"/>
                <MethodCall ReturnValueDest="_openIcon"
                            ReturnType="javax.swing.ImageIcon"
                            name="constructor">
                     <Param DataType="java.lang.String" Source="open.gif"/>
                </MethodCall>
                <Property name="icon" value="_openIcon"/>
                <Event type="action">
                     … MethodCalls …
                </Event>
```

[1] Further implementation-specific details are provided in Chapter 5, which deals with the implementation of the RADGIS application.

```
            </Component>
            <Component name="SaveMI" type="MenuItem">
              <Property name="text" value="Save"/>
                … additional details for Save MenuItem omitted for brevity …
            </Component>
            <Component name="ExitMI" type="MenuItem">
              <Property name="text" value="Exit"/>
              <Event type="action">
                  … MethodCalls …
              </Event>
            </Component>
        </Component>
    </Component>

</GUI>
      … help system inform ation …
```

Although DGCML was originally developed to support GIS services implemented as remote objects, it also supports the use of local GIS services that have downloaded to, and are executed on, the client machine as part of the client GIS application.

We have chosen to implement the local components as Java objects, deployed as JARs, and the remote components, as Enterprise JavaBeans (EJBs) and CORBA Objects. It should be noted that some services are implemented as a combination of finer-grained obje cts that may be local or remote. Thus, like BeanML, our DGCML facilitates the creation of hierarchies whereby one "complex" DGCML descriptor may contain a reference to one or more DGCML files that contain descriptions of "simple" components that may reside locally or remotely.

The idea of using XML to specify how JavaBeans may be wired together to create an application, is not novel (see *section 3.2.2* which describes BeanML). However our approach to the creation of an XML descriptor for GIS services, hat includes application metadata and the ability to wire Java objects and Swing components to provide a GUI, that facilitates the *invocation of local and/or remote methods*, and can be incorporated into a client GIS application at runtime, is novel.

3.5. Summary

In this chapter we have presented four XML technologies that overlap with, or have influenced, the development of our DGCML. BeanML and XwingML are markup languages that enable a user to compose applications based on JavaBeans, and develop GUI's based on the Swing/JFC classes respectively. OSD and JNLP are deployment descriptor vocabularies that allow applications to be deployed across the Internet.

Our DGCML can therefore be best described as being a combination of a deployment descriptor and a wiring language for components that implement GIS services, or even stand-alone applications. All that is required on the client-side is the DGCML interpreter that reads in the DGCML descriptor, downloads the necessary support classes, and builds the GIS service using Java Reflection and JavaBean APIs.

Chapter 4

Factors influencing the design of RADGIS

"I can't understand why people are frightened by new ideas.
I'm frightened of old ones." John Cage (1912-1992)

With the explosion of the Internet, one has seen a shift in focus from commodity-based economies to data-driven economies. Many companies have invested heavily in systems that will allow them to perform data mining, for example data visualisation and trend analysis, in order to retain competitive advantage. However, with the exception of specific industries that rely heavily on GIS functionality, e.g. mining, telecommunications, forestry and conservation, most businesses have not made use of spatial analysis to support business decisions, or in the derivation of income.

This is surprising when one considers that according to studies [Daisey 2000], including "GIS in business" conducted by Dutch-based Ravi Business Platform in collaboration with the Vrije Universiteit, Amsterdam and Manchester Metropolitan University, UK, an estimated 80 to 90 percent of business-related information, particularly business support systems, is geographical in nature [GeoEurope 2000]. However, as industry is made aware of the intrinsic value of location, through innovative developments such as Mobile Location Services[1], this is rapidly going to change.

Many GIS vendors and spatial data providers, on the other hand, have been quick to realise the potential of the Internet as the next-generation GIS platform and geospatial data distribution tool, and have moved with the times to ensure that they are able to tap into this particularly lucrative new market [Toon 1997][Gifford 1999].

There are two main areas where the development of GIS applications have benefited most from the explosion of the Web and Web technologies. These are the development of Web/Image Mapping Servers and browser-based front-ends that provide access to maps and geospatial data, as well as the development of sophisticated client-side GIS applications that integrate geospatial services and data, distributed across the Web, in a transparent manner.

Web browser-based GIS applications and Web/Image Mapping Servers have been criticised for their lack of functionality because they do not provide tools for performing analysis on the geospatial data, i.e. they are generally no more than data viewers or data explorers. The focus of our research has therefore been on the development of an extensible client-side GIS application. It is here that we have developed a novel approach towards allowing the user to create, and customize, their own component-based client-side GIS application, which has the ability to make use of distributed services and data. Our work therefore fills an important niche that has been neglected, or at best only partially addressed, until now.

The relatively new concept of Location (Based) Services has brought with it the promise of individual services that provide user-friendly applications of GIS operations. The introduction of location services has also added much value to our research. Not only does it add to the industry impetus to create re-usable GIS services, it also provides another application within the field of GIS for demonstrating the flexibility of our approach, i.e. in the development of an extensible client-side framework for utilising location services.

---

[1] The Location Services industry is currently the most rapidly expanding sector of GIS. Its growth rate has already exceeded that of the traditional GIS market [International Data Corporation 2000].

This chapter provides an overview of different GIS architectures currently available, as well as work done by the Open GIS Consortium, in order to illustrate the synergy between our approach and the work that is being undertaken by major developers and standardization bodies. It also discusses the following topics that form the theoretical foundation of our approach to RADGIS, which is introduced at the end of the chapter:

?? The origin of a standardised, XML-encoded data transfer format called GML, based on the OGC's Simple Features Specification. Components developed for our RADGIS application must, in addition to any proprietary formats, support GML so that geospatial data transfer between distributed components within the RADGIS application can be standardised on GML, simplifying interoperability.

?? Work done by the OGC with respect to the provision of interoperable geospatial services (OpenGIS® Service Architecture), which forms the basis for implementing our local and distributed services.

?? Location (Based) Services, which are being considered as a special application of our RADGIS architecture, as well as fuelling the market for re-usable GIS components.

4.1. GIS Architectures

It is generally accepted that GIS applications can be logically disaggregated into three main functions [Morais 2000], namely:

?? *data management* – the storage and retrieval of spatial and a-spatial attribute data, generally via a database management system.

?? *operations and analysis* – the "business logic" that implements the logical processing of the data, including things like feature overlay, image manipulation and analysis, and map projection.

?? *rendering and user interface* – the software components that the user interacts with that support the presentation of data.

GIS applications have undergone many architectural changes over the past two decades that have been in keeping with advances in tiered application development in general. GIS systems developed during the late 1970s and early 1980s were based on the single unconnected workstation model, and did not separate the tasks of *data management*, *operations and analysis*, and the *rendering and user interface*. Instead, these components were tightly coupled and sold as part of a single entity (or single-tier application).

However, with the emergence of the client-server systems architecture in the early 1990s, aided by the introduction of relational databases and relatively inexpensive desktop computers, GIS applications slowly disaggregated the *data management* duties from the *operations and analysis*, and *rendering and user interface*.

This facilitated the use of a database from a vendor other than the GIS application vendor, as well as the ability to make use of remote data access, i.e. accessing data from a machine other than the machine on which the GIS application was running. Remote Data Access (RDA) provides an effective and scaleable way to distribute processing and storage over a network, separating the data repositories from client applications [Grady, *current*]. Thus, a single copy of the dataset could be accessed by multiple users simultaneously (according to the transaction rules of the database), reducing the possibility of data inconsistency due to keeping multiple copies of a dataset up-to-date. Many of today's GIS installations still make use of this "two-tier" architecture, for example storing their spatial data on a central server, such as ArcSDE or Oracle Spatial database, and using a client such as ArcView or Arc/Info to manipulate this data.

The most recent evolutionary stage in systems architecture is the three-tiered architecture, also referred to as the n-tiered approach. In n-tiered architectures, there is an explicit decoupling of the *operations and analysis* (business logic) component of GIS applications from the *user interface* and *data management* components.

One of the key benefits derived from separating application functionality into multiple tiers, is the ability to replace the implementation of a particular tier without affecting the implementation of the other tiers, e.g. changing the database used to store the data should not affect the *operations and analysis*, and *user interface* tiers.

The focus of our research is on the development of re-usable GIS *operations and analysis* components, such that the business logic of our GIS applications is made up of interchangeable components, possibly developed by different vendors at different times.

One of the biggest challenges is how these components are to interact with other software components, about which they have no compile-time knowledge. Thus during the design of the components, the emphasis should be on non-specialised and non-proprietary interaction, to ensure extensible, yet robust components that result in the greatest possible level of reuse.

The successful implementation of an n-tiered component-based application can therefore derive much benefit from the use of industry standards, and improvements in component software technology [Morais 2000]. Fortunately, within the field of GIS, there have been a number of standardisation efforts seeking to create industry standards for:

?? geospatial data representation and transfer formats, e.g. the OGC's Simple Features Specification and GML, ISO/TC 211's geospatial data model, the Federal Information Processing Standard (FIPS 173) - Spatial Data Transfer Standard (SDTS), and various others by the Federal Geographic Data Committee (FGDC) and the US Geological Survey (USGS) Group; and

?? the implementation of services based on open standards, e.g. the OGC's OpenGIS® Service Architecture, Web Mapping Server, Web Feature Server, and Catalog services, and the ISO/TC 211's geospatial services.

Emerging GIS software varies widely in performance, quality, feature set, cost and, most importantly, fundamental system architectures. These architectural differences have a significant impact on how the software performs in an Internet-based computing environment [Gifford 1999]. Therefore, before taking a look at the standardisation that is being undertaken by the OGC, we briefly describe the different GIS architectures and types of GIS applications currently available in order to categorise our RADGIS application, and explain why our approach is novel.

The applications have been divided into two categories: client-side GIS applications and server-side GIS applications. The classification of a GIS application as server-side or client-side, is based on where GIS services are executed as opposed to where the resultant map is visualized. Thus a client-side application that simply provides an interface for viewing a geospatial image generated by a GIS server is not considered to be a client-side GIS application, as the complex GIS calculations and data remain on the server.

4.1.1. Server-side GIS Applications

Server-side GIS, and in particular Internet Mapping, is a relatively simple and cost-effective method of allowing anyone with access to the Web to access maps and GIS-based data and services, based on easy-to-use browser-based formats or thin clients. In addition, the server-side architecture centralises control over the geospatial data and services, which significantly simplifies the deployment, security/access control permissions and maintenance of server-side GIS applications.

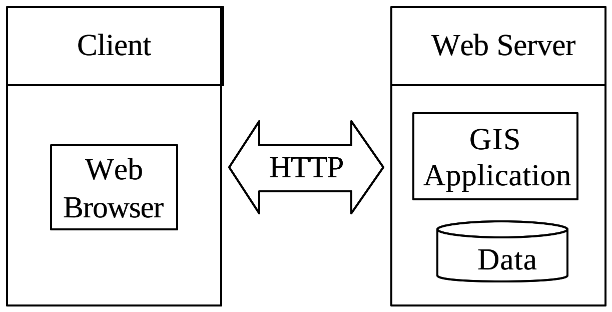


Figure 4.1. Server-side Architecture

Table 4.1 below summarises some the advantages and disadvantages of the server-side architecture. For a more comprehensive discussion on server-side and client-side GIS architectures, the reader is referred to [Gifford 1999].

Advantages to Server-Side GIS

Adherence to Standards

?? Can adhere to all Internet/Web standards

?? Can be accessed with standard Web browser

?? Eliminates platform issues as much as possible

Performance

?? Significant GIS functionality can reside on the server

?? Large GIS databases can be accessed on the server

?? Low bandwidth requirements

?? Performance per request is predictable

Cost of Ownership

?? Centralised administration of data and GIS application software

?? User support is minimal

Disadvantages to Server-Side GIS

Adherence to Standards

?? No viable vector formats

?? One-click functionality from Web clients [1]

?? Low graphics quality

?? Primitive GUI

Performance

?? Creates many requests

Information re-transferred for each request

Table 4.1. Advantages and disadvantages of Server-side GIS [Gifford 1999]

[1] It is often necessary to allow the user to select multiple features before executing an operation.

4.1.1.1. Web Map Servers (WMS)

A number of Web/Image Mapping Servers have been developed recently that address the need for out-of-the-box, GIS and mapping solutions for publishing GIS maps on the WWW. These include products like ArcIMS, Internet Mapper, Autodesk MapGuide Server and MapObject IMS.

Traditional Web Map Servers obtain geospatial data from a spatial database, in response to a Web browser request, and then return the resultant map as an image that can be viewed by the Web browser, using standard HTTP. This provides an easy to use, automated interface for obtaining maps, but is very limited in terms of traditional GIS capability, as all the complex and proprietary software, in addition to the spatial and tabular data, remain on the server.

The use of a Web-browser provides a simple, standard mechanism for viewing the resultant maps, but there are also number of drawbacks to this approach, including the relatively primitive GUI, low graphics quality and one-click functionality of browsers.

The maps generated by WMSs are generally static images that do not contain any geospatial encoding. This is because most standard Web browsers only support GIF and JPEG images. Thus, even though one may be able to perform simple pan, zoom and navigation-type operations, it is not generally possible to perform any form of spatial analysis, query any information about points or areas on the maps, or use the resultant image as a data source for further geospatial processing. In addition, if the image needs to be modified, e.g. during a pan/zoom operation, turning a layer on or off, etc, it requires a request to be sent to the WMS to regenerate the image from the original dataset using the new set of parameters. This may result in many requests and responses being generated, which in turn could cause poor performance [Gifford 1999].

However, a growing number of WMS are now equipped to provide far more than static images. WMS that implement the Open GIS Consortium's WMS Specification (see [Open GIS Consortium 2000b] for more information) also have the ability to produce, for example, georeferenced images (geoTIFF) or Features[1] (GML), allowing the resultant map to processed further, for example using chaining, or directly in a client-side GIS application. However, once again we must reiterate that these formats are not supported by current Web browsers, and are therefore intended for use by more sophisticated clients.

Web Mapping Servers fulfil a very useful role within GIS, particularly for serving data on the Web. Thus, the restrictions mentioned above are not criticisms of their architecture, but simply illustrate their intended use as static image servers for viewing spatial data on the web, and not as full-blown distributed GIS application servers. However, it is worth noting their potential use as data servers in distributed GIS applications.

4.1.2. Client-side GIS Applications

Client-side GIS requires the installation of pre-packaged or downloaded GIS-enabled software on the client machine, such as traditional GIS applications or GIS Java applets, ActiveX components, or plug-ins for a Web browser [Marshall 2000].

Client-side architectures that interact directly with the geospatial data, as opposed to static map images, enable a large set of graphic and GIS operations to be performed locally. Thus unlike the WMS (see *section 4.1.1.1*), changes to the display, caused by operations such as panning or zooming can occur without re-transmitting a request to the server, resulting in a significant performance improvement. Thus, the major advantages of client-side solutions are the abilities to enhance user interfaces, e.g. the ability to perform multipoint feature selection, and improve performance through the reduction in network traffic.

The major disadvantage associated with client-side GIS solutions such as Java applets, ActiveX components, or plug-ins, is that they require software to be downloaded to the client machine (and installed) before the user is able to browse the data. This may be acceptable for

[1] The ability to return Features, e.g. as GML, rather than static images, was originally intended as a major extension to the Web Map Server specification. However, it has now become a separate interface specification, called the Web Feature Server specification [Vretanos 2001]

someone who intends using a particular product regularly. However, for someone who rarely uses mapping services, it is far easier to access a Web page that immediately allows him/her to start interacting with map data, than have to download different client software implementations when wishing to browse data from different vendors [Gifford 1999].

Table 4.2 below summarises some the advantages and disadvantages of the Client-side architecture. For a more comprehensive discussion on Client-side and Server-side GIS architectures, the reader is referred to [Gifford 1999].

Advantages to Client-Side GIS

Adherence to Standards

?? Document/graphics standards not required

?? Vector data can be used

?? Image quality not restricted to GIF and JPEG

?? Modern interfaces possible

?? Not restricted to single -click operations

Performance

?? Excellent performance for operations that occur locally

?? Less Internet traffic required

Disadvantages to Client-Side GIS

Adherence to Standards

?? Non-conformance limits user base

?? Requires users to obtain additional software

?? Platform/browser incompatibility

Performance

?? Initial download times can be substantial if databases transferred

?? Users must wait for software to download

?? Overall performance can be low with large databases

Table 4.2. Advantages and disadvantages of Client-side GIS [Gifford 1999]

4.1.2.1. Traditional GIS applications

Traditional GIS applications such as ARCInfo, MapInfo, ArcView, and TNTmips are generally very large applications that are expensive to license, have a very steep learning curve [Albrecht 1996], and are difficult to use. These applications are intended for use by GIS experts, who have a solid understanding of cartography. Although most provide the ability to install additional modules for specialist tasks, or develop customised functionality through scripting languages, they do not allow the application to be tailored towards users of different levels-of-expertise or based on the user's application domain.

While there is no denying that traditional GIS applications fulfil an extremely valuable role, many problems associated with these GIS applications have been identified, including application bloat, vendor lock-in, and lack of interoperability and location transparency. (Please refer back to *section 1.1* for a more detailed discussion of these problems).

4.1.2.2. Java Applets

A number of Java applets implementing rudimentary GIS functionality have emerged recently. However, there are a number of problems with implementing GIS clients as Java applets that limit the potential of this approach. These include having to download the applet each time the client is used, and security restrictions on applets, which must be signed with digital certificates in order to obtain special permissions to read/write to the hard drive and/or create a network socket connection to a server other than the server from which the Applet was downloaded. Furthermore, applet-signing mechanisms are complicated and vary between Netscape and Internet Explorer [Griscom 1999].

Applets rely on the Web browser for their Java Virtual Machine (JVM). Even though they are supposed to implement the same standard, there are often differences between the JVMs of different browsers, including incomplete implementations of certain features. Neither Netscape nor Microsoft currently allow one to upgrade the browser's JVM, and thus one either has to develop applets for old versions of the JVM, or make use of the Java plug-in. However, because the plug-in approach uses an external JVM, it is not possible for the Java applet to interact with the Web browser, which once again limits its functionality [Rohaly 2000].

Even if the above-mentioned problems could be avoided, execution of Java (applets) within a Web browser is much slower than standalone Java execution [Rohaly 2000]. This poor performance is a very real problem that may ultimately dissuade a user from using an applet-based GIS.

However, possibly the biggest problem with developing client-side GIS software as applets, is that they must be downloaded from the Web Server each time they are to be used. If the applet is too large, it will take too long to download and the user will invariably seek an alternative solution. Therefore, the functionality of the applet is severely limited by its size, and applets are generally implemented with a specific task in mind.

4.1.2.3. Plug-in approach

A plug-in is a program module that extends the functionality of Web browsers, enabling them to support new types of multimedia content[1]. In comparison with Java applets, most GIS plug-ins provide faster visualization and better levels of interactivity. In addition, GIS plug-ins do not suffer from the security restrictions imposed on Java applets, or the need to download the applet each time a data set is to be viewed because the code for the plug-in is installed on the client machine.

Plug-ins for most commercial GIS applications are becoming more and more prevalent, e.g. Autodesk mapGuide plug-in, ArcExplorer. Most of them are freely available, because they are fairly limited in functionality (in comparison to traditional GIS applications). They generally simply fulfil the role of data viewers/browsers for

[1] Technically, the term “plug-in” refers to a module that conforms to the Netscape Navigator standard, whereas modules designed for Internet Explorer use the ActiveX software standard. However, we will us e the generic sense of the term plug-in to refer to modules that conform to either standard.

proprietary spatial data formats, with limited analytic capability and/or image processing capabilities.

Due to their limited functionality and design, as well as problems affecting the interaction of different plug-ins within the Web browser environment[1], plug-ins and traditional GIS applications sit at the opposite ends of the customisability spectrum. Plug-ins are essentially small, specialised, standalone modules that cannot be extended, while traditional GIS applications suffer from serious application bloat, but cannot be trimmed of unnecessary features.

4.2. OpenGIS®

It is unlikely that a single architecture for developing GIS applications will triumph. Instead, because different users have different requirements and preferences, a number of models, from traditional GIS applications to specialist GIS applications based on highly distributed components will co-exist. The fundamental issue that will determine whether or not these systems co-exist successfully is that of interoperability.

The OGC's vision for the future of GIS is "*the full integration of geospatial data and geoprocessing resources into mainstream computing and the widespread use of interoperable geoprocessing software and geodata products throughout the information infrastructure*" [Open GIS Consortium 1999a]. In order to facilitate this vision, its working groups have developed abstract specifications and implementation specifications for its two central technology themes of sharing geospatial information and providing geospatial services (see *table 4.3* and *table 4.4*).

[1] While the interaction of certain combinations of plug-ins is possible using a particular Web browser architecture and version, differences in fundamental architecture in different Web browsers, as well as within versions of a particular Web browser, have severely limited the ability for developers to utilise a standard mechanism to allow plug-ins to interact effectively.

4.2.1. The Abstract Specifications

The Abstract Specification documents provide the theoretical background for the Implementation Specifications (see *section 4.2.2*), as well as providing a technically complete "language" to discuss issues of interoperability [Open GIS Consortium, 1999a].

Each of the topics described in the OpenGIS® Abstract Specification documents is composed of two models:

?? the Essential Model that describes a conceptual link between the software system and the real world, and

?? the Abstract Model (the core of the Abstract Specification) that describes how the eventual software system should work in an implementation neutral manner.

Figure 4.2 shows the dependencies between the topics described in the OpenGIS® Abstract Specification, while *table 4.3* provides a very brief description of each of them.

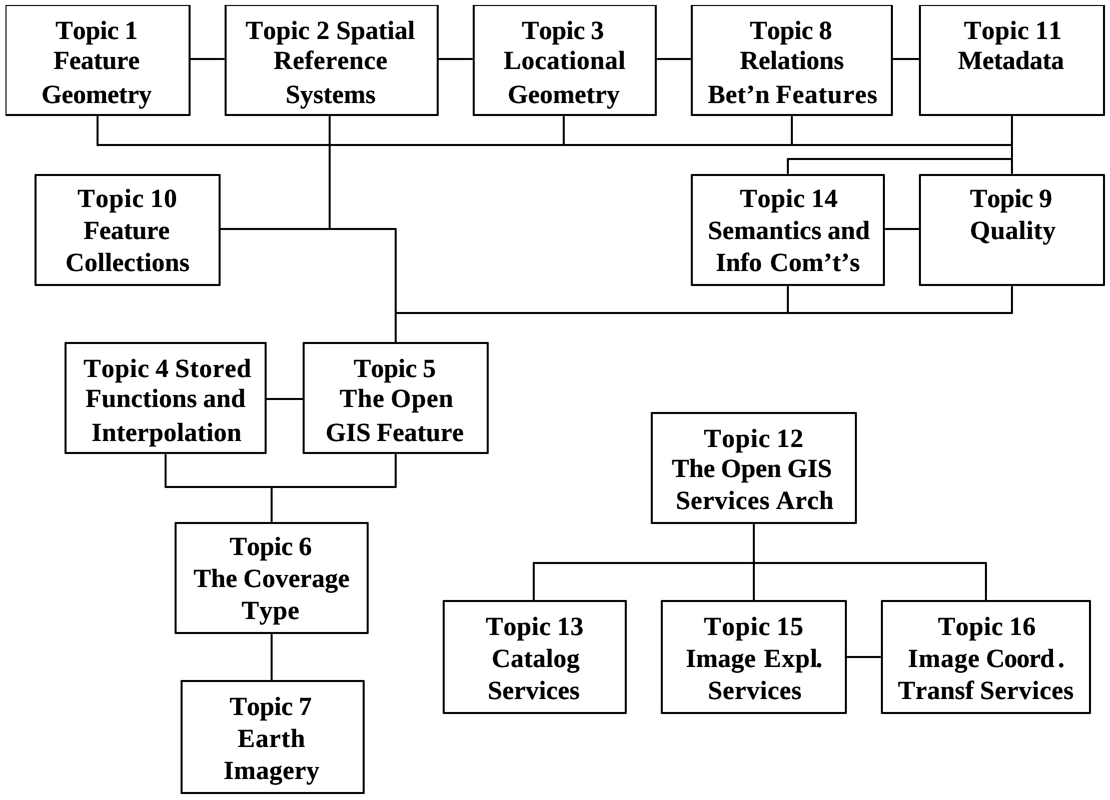

Topic 1 Feature Geometry
Topic 2 Spatial Reference Systems
Topic 3 Locational Geometry
Topic 8 Relations Bet'n Features
Topic 11 Metadata
Topic 10 Feature Collections
Topic 14 Semantics and Info Com't's
Topic 9 Quality
Topic 4 Stored Functions and Interpolation
Topic 5 The Open GIS Feature
Topic 12 The Open GIS Services Arch
Topic 6 The Coverage Type
Topic 13 Catalog Services
Topic 15 Image Expl. Services
Topic 16 Image Coord. Transf Services
Topic 7 Earth Imagery

Figure 4.2. Dependencies between Abstract Specification topics

Table 4.3 provides an overview of the OGC's Abstract Specification as of July 2001 (adapted from [Öhrström 2001]). For further information please refer to the following OpenGIS® Web page: http://www.opengis.org/techno/specs.htm

Specification	**Purpose**
Overview	Provides an overview of the OpenGIS® Abstract Specifications
Feature Geometry	Describes an abstract model for the geometric representation of GIS-objects (i.e. features)
Spatial Reference Systems	Contains definitions of classes for reference systems, data types, units and operations
Locational Geometry	Functions for mapping Features from one locational system to another
Stored Functions and Interpolation	Calculating functions, interpolation, and extrapolation
The OpenGIS® Feature	Modelling real world and abstract entities
The Coverage Type	The formulation and calculus of the Coverage Type and its subtypes
Earth Imagery	Image geometry models, and models for computing the real world-model connection
Relationships Between Features	How to model relationships between Features
Quality	Defines various position accuracy terms and concepts
Feature Collections	Models for handling Feature collections
Metadata	Models for handling Feature and Feature collection metadata
The OpenGIS® Service Architecture	A framework of services required for the development and execution of geospatially oriented applications

Catalog Services	OpenGIS® services for data discovery and data access
Semantics and Information	Communities sharing data between communities
Image Exploitation Services	Functions for image exploitation, such as Feature extraction
Image Coordinate Transformation Services	Services for transforming image position coordinates, to and from ground position coordinates

Table 4.3. Abstract Specification overview as of July 2001 [Öhrström 2001]

4.2.2. The Implementation Specifications

The Implementation Specifications documents, are a set of specifications, based on the Abstract Specifications (see *section 4.2.1*), that contain guidelines for implementing OpenGIS® applications or components. *Table 4.4* provides a very brief description of each of the Implementation Specifications. For further information refer to the following OpenGIS® Web page: http://www.opengis.org/techno/specs.htm

A "Testing Program" has been developed by the OGC to test for conformance of products to the OpenGIS® Implementation Specifications, and at a later stage, to test for interoperability between products. The conformance test is used to determine if a product implementation of a particular Implementation Specification fulfils all the mandatory elements. However, it does not ensure, or even test for, the interoperability of software products. Instead, the OGC hopes that as the specifications mature, the likelihood of interoperability will become higher.

Specification	**Purpose**
Simple Features Specification	Specification for the handling of simple geometric representations of GIS-objects, such as polygons (excludes 3D), and reference systems
Catalog Services Interface Implementation Specification	Specifies how geospatial handling over networks should be implemented

Grid Coverage Implementation Specification	Specification for all types of raster based images. Interfaces for analysis and calculation, such as histogram, covariance etc
Coordinate Transformation Services Implementation Specification	Strategies for coordinate systems and transformation between them
Web Map Server Interfaces Implementation Specification	Defines services necessary for Web-based access to geo-data and processing
Geography Markup Language (GML) Implementation Specification	XML encoding of the Simple Features Specification

Table 4.4. Implementation specifications as of July 2001 [Öhrström 2001]

4.2.3. Simple Features Specification

A Feature, as defined by the OGC, is the encapsulation of measurable or describable phenomena about real world or abstract entities. It is the fundamental unit of geospatial information and consists of both spatial and attribute data. Simple Features are a subset of Features, that only support linear interpolation between coordinates and do not consist of other features, i.e. they are atomic [Miller and Schirnick 1999]. Simple Features represent vector data, such as roads, land-use zones, and watersheds, as points, lines, arcs and polygons. They do not, however, support the representation of raster data[1].

The Simple Features specification does not provide details of how to map features to real world objects. Instead, it provides a specification for the implementation of mechanisms to work with Features and Spatial Reference Systems [Öhrström 2001]. This allows the communication of simple geometry, spatial reference system and attribute information between applications or components that conform to the Simple Features Implementation Specification.

[1] Vector data consists of a series of points (coordinates), some of which are joined by lines (i.e. sets of related points), and some line segments (arcs) are joined to form polygons. Raster data is composed of a grid of cells that represent geographic features, i.e. a georeferenced bit-mapped image.

The Open GIS Consortium has also produced Simple Features specifications for OLE/COM, CORBA and SQL for working with simple geospatial features in a distributed or component-based computing environment, or an SQL database, respectively. Due to the fact that Java has become the dominant software language for developing distributed enterprise-level applications, and since an estimated 80% of corporate data has a spatial component, Sun Microsystems realised that there was a need to add geoprocessing capabilities. Thus, there is now also an informal workgroup working on creating a Simple Features specification for Java [Daisey 2000].

4.3. Exposing data efficiently – The future of geospatial data access?

As mentioned in *section 4.1.1.1*, most Web Map Servers simply generate and then serve GIF or JPG map images to the user's Web browser. This approach is straightforward and works well when the user simply wants to view a low-resolution map image, but does not allow the user to process the result further in a client-side GIS application, or to make use of it as part of a service-chaining request.

In addition, most Web mapping applications are inseparably tied to a specific server implementation, i.e. the client is hard-coded to interact with a particular vendor's proprietary map server implementation. Thus a user must run different client applications in order to access the data and functionality provided by different server implementations. This lack of interoperability or reuse of client and server implementations, severely limits the ability of a user to transparently access data from multiple disparate data sources. [Wang *et al*. 2001]

This section introduces two technologies that have the potential to overcome the above-mentioned problems, providing much needed capabilities for the efficient indexing of searchable, georeferenced metadata as well as the delivery of geospatial data in a vendor independent/open format.

4.3.1. The .geo proposal

Information about a particular location is of more interest to people close to that location than people who are far away [Lake 2000]. Therefore, it makes sense to have most of the information regarding a particular location, stored and maintained in that vicinity. However, one must also remember that remote users may still be interested in that data, and therefore one must also ensure that such data is readily accessible, and may be easily integrated, on a regional and global scale.

The *.geo* proposal, submitted to The Internet Corporation for Assigned Names and Numbers (ICANN) by SRI International, for an open infrastructure for registering and discovering georeferenced information on the Internet, attempted to do just that.

The *geo* proposal was submitted to ICANN in response to an initiative of the ICANN Board to select a limited number of diverse proposals that could be used as a proof-of-concept on which to base decisions about introducing future Top-Level Domains[1] (TLDs). It motivated the creation of a TLD, called *.geo*, which could be used to index searchable, georeferenced metadata, using a modified version of DNS to encode latitude/longitude bounded cells as domain names in the form of minutes.degrees.tendegrees.geo

For example:

?? The geographic domain name *10e40n.geo* identifies the 10-degree x 10-degree cell whose southwest corner is located at 10 degrees east, 40 degrees north.

?? The geographic domain name *2e4n.10e40n.geo* identifies the 1-degree x 1-degree cell whose southwest corner is located at 12 degrees east, 44 degrees north.

?? The geographic domain name *11e21n.2e4n.10e40n.geo* identifies the 1-minute x 1-minute cell whose southwest corner is located at 12 degrees, 11 minutes east and 44 degrees, 21 minutes north.

---

[1] The *.geo* proposal was not accepted as one of the seven proposals selected for the proof-of-concept phase by ICANN, and a reconsideration request was rejected by ICANN on 16 March 2001. The reasons provided by ICANN include the complexity of the proposal, and ICANN's cautious approach to introducing new top-level domains. This does not, however, prevent the .geo proposal from being accepted in the future [ICANN Committee 2001].

The beauty of the *.geo* proposal is in its simplicity, as it provides an easy to use mechanism for finding information for a particular area. For example, if one were interested in obtaining information about Grahamstown, one would simply need to determine its longitude and latitude co-ordinates (possibly from a geo-referencing service). From this information, one could easily infer the domain name of the geospatial data (and service) server for that area. If a Web Map/Feature Server was running on that machine, one could very easily make use of the standardised request format to obtain the desired information (possibly as GML-encoded data). Additional benefits of the *.geo* approach are that it limits the amount of metadata per server, and it drastically reduces server bandwidth because clients can directly query the relevant cell server(s) to find metadata for a given area [Leclerc *et al.* 2001].

Regardless of whether or not the *.geo* proposal is accepted at a future date, it has made an invaluable contribution in this field, and highlights the need for an efficient mechanism to index spatial information based on location. For a full description the .geo proposal, please refer to [Reddy *et al.* 2000a][SRI Internet Initiative 2000].

4.3.2. A standardised spatial data transfer format - GML

As part of the OGC's development of specifications for sharing geospatial information and providing geospatial services, it developed the Simple Features Specification (see *section 4.2.3*) based on the Feature and Geometry models of the OpenGIS® Abstract Specification. It has subsequently developed an XML encoding of the Simple Features Specification, called the Geographic Markup Language (GML 1.0). GML is an open standard for marking up geospatial information, including both properties and the geometry of geographic features. It allows one to deliver geospatial information as distinct features, as well as specifying how the features are to be displayed (using a particular stylesheet) [Galdos Systems Inc. 2001].

GML is not a presentation format, but must be styled for presentation e.g. to Scalable Vector Graphics (SVG) or X3D (see *section 5.1.1.3*), using an appropriate style sheet. It is therefore possible for users to view the resulting maps using a standard browser (once the relevant plug-in has been installed, e.g. SVG plug-in from Adobe), negating the need for a proprietary client-side GIS application to visualise the geospatial data.

Figure 4.3 illustrates how GML data can be displayed in a standard, XML-enabled Web browser using an XSLT stylesheet that maps the GML data into an appropriately represented SVG image. The manner in which GML data is displayed, is determined by the creator of the GML to SVG stylesheet. It is therefore possible to have a number of different stylesheets for the same type of feature, e.g. one stylesheet may represent roads as a thin black line, whereas another stylesheet may represent roads using a thicker, red line. It is also possible for stylesheets to map the same GML feature to different SVG-based symbols, depending on the purpose of the map. *Figure 4.4* shows the intended use of GML in the RADGIS architecture.

GML is intended to enable the transport and storage of geographic information in XML and is anticipated to provide greater interoperability between GIS applications, to enable linked geographic datasets, and make a significant impact on the ability of organizations to share geographic information with one another. Although a relatively new technology, GML has already been hailed as a success, and according to [Wang *et al.* 2001] "*GML represents one of the most visible steps taken by the geospatial community towards the vision of widespread spatial interoperability*".

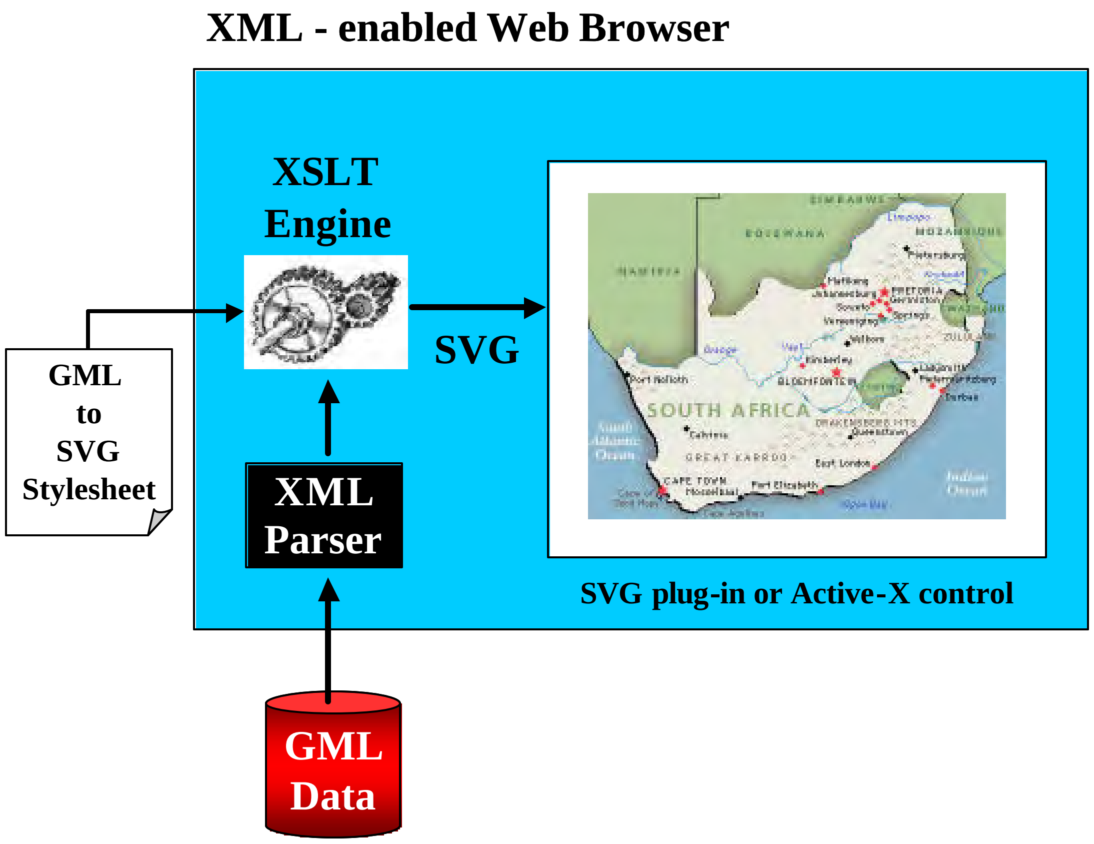


Figure 4.3. Displaying GML data in an XML-enabled Web Browser

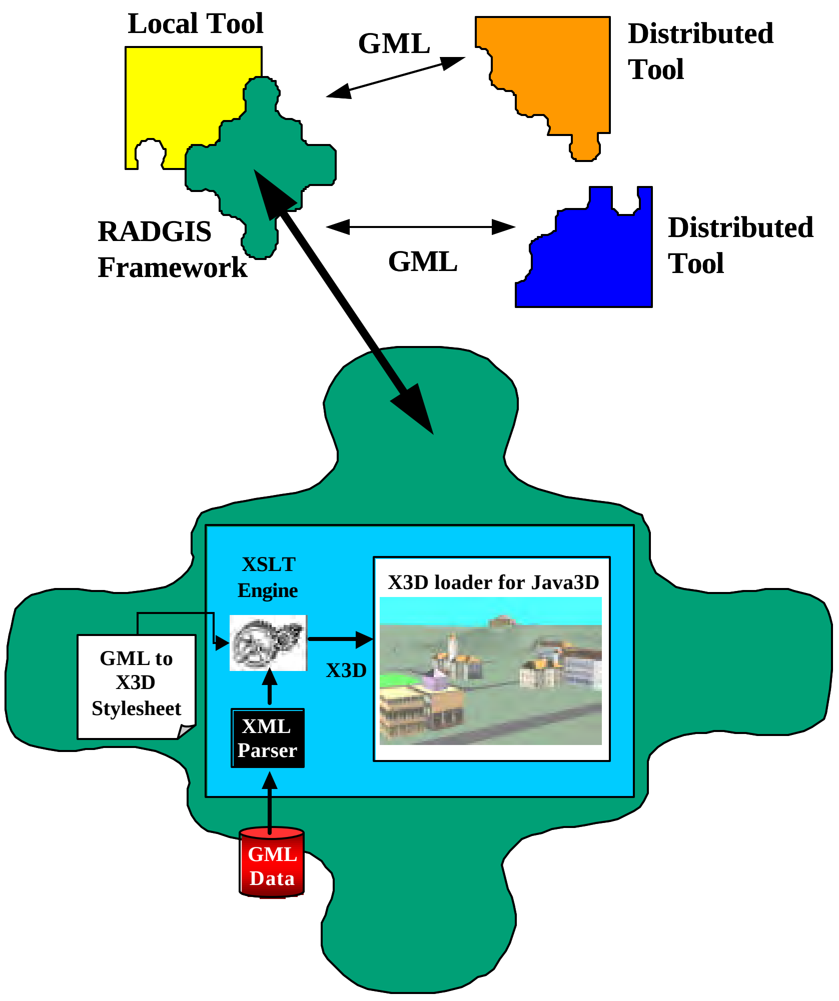


Figure 4.4. The intended use of GML in the RADGIS architecture

The use of GML rather than static GIF/JPEG images (which contain no geographic encoding) has many advantages including: better quality maps; no need to only target Web browsers although it can be rendered by most current XML-enabled Web browsers, without the need to purchase client-side software; custom map styling (using appropriate stylesheets); ability to create editable maps; more sophisticated linking capabilities; better query capability; control over content (filtering); animated features; and service chaining [Galdos Systems Inc. 2001]. Many of these advantages are due to the fact that GML is based on XML technologies.

The current version, GML 2.0, now based entirely on XML Schema (XSD), was released on 20 February 2001 and significantly expands GML 1.0 to include the encoding of complex features and feature associations. GML 3.0, which is to be finalised towards the end of 2001, will provide many useful extensions including topographic support, events, coverages as well as histories and feature timestamps [Lake 2001a]. These will allow more complex analytic GIS processing of GML-encoded spatial and temporal data, which in turn will facilitate the development of more sophisticated GIS tools and location services, based on the open GML standard.

4.4. Interoperable Geospatial Services

Within the OGC's specification of geospatial services, it has defined geospatial domain-specific business objects to ensure that the OpenGIS® Services Architecture can be realised with standards-based, Commercial-Off-The-Shelf (COTS) products available from multiple vendors [Open GIS Consortium 2001a]. The OGC also envisages that as developers implement products with OpenGIS® interfaces, interoperable geographic applications will be composed of components from the OpenGIS® Services Model and other supporting and compatible information services [Buehler and McKee 1996].

Many of the components will be implemented to run locally on the client machine as core services (e.g. the OpenGIS®'s GeoSpatial Display Services), as add-on modules, or stand-alone utilities. However, there is a strong business case for developing distributed implementations of geospatial and image manipulation services - specifically, but not limited to, those identified in the OpenGIS®'s Geospatial Coordinate Transformation Services, Geospatial Analysis Services, Image Geometry Model Services, Image Synthesis Services and Image Understanding Services. For a full list of Geospatial Domain Services as classified by the Open GIS Consortium, please refer to [Open GIS Consortium 1999b]. This view has recently been confirmed when it was announced that the Open GIS Consortium's Interoperability Program for 2001 was focusing on defining a Web Services architecture that would support the deployment of spatial services using WWW Protocols [Doyle 2001].

4.4.1. Motivation for the development of distributed implementations of interoperable Geospatial Services

In addition to the benefits derived from the ability to invoke location-transparent interoperable geospatial services, there are two main areas in which the use of distributed objects has the potential to optimise the execution of geospatial services: one is speeding up the processing of a particular service by using parallel processing or executing the service on a faster (remote) platform; and the other is reducing bandwidth requirements when one is making use of data from the internet, by performing as much of the processing as possible "close to" the data source.

4.4.1.1. Faster processing

Consider the following scenario, which would provide an extremely useful service to organizations that cannot afford to purchase expensive GIS software or powerful processing platforms. A fairly common geospatial operation would be to perform viewshed analysis on a particular spatial dataset, i.e. determine all possible locations that are visible from a particular point. However, depending on the type of work done by the user, it may not be a tool that is used often, and would almost certainly not be included in most "lite" GIS applications. It would also probably not warrant purchasing a large GIS application if this were the only "advanced" feature required.

There are two possible implementation solutions that are both catered for using our approach to the runtime integration of new services into our proof-of-concept system. The first is to allow the user to download and integrate a local version of this tool, and the second is to enable the user to download and integrate a front-end interface to a remote implementation of this tool.

It is the latter option that we will consider now, as it is the more complex solution, and has the potential to provide the additional benefits, not available to the version that would run on the client machine. Thus we will discuss the ability to perform the viewshed analysis remotely, on a faster machine located across the Internet, without having to purchase a local copy of the viewshed tool, or an entirely new GIS application that supported this operation.

Viewshed analysis, or line-of-sight mapping, is a computationally expensive operation that draws a line from the point of interest (e.g. **Point A** in *figure 4.5*) through each location, or cell, in a raster image to determine the slope between the two points (e.g. from **Point A** to **Point B** in *figure 4.5*). If any of the cells between these two points has a height value greater than the interpolated height value from the slope at that location, then the point being examined (**Point B**) is not visible from the point for which the viewshed is being performed (**Point A**). In *figure 4.5*, **Point B** is not visible from **Point A** because the height value of an intermediary cell, at **Point C** is greater than the interpolated height value.

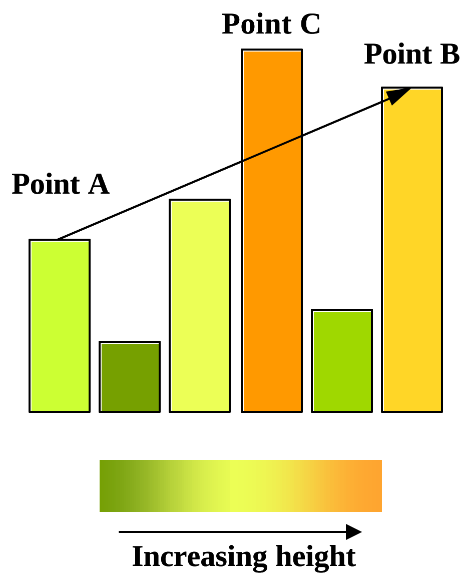


Figure 4.5. Viewshed Analysis Example

Even the most efficient algorithms are still at O(n^2 log n) [Kreveld 1996]. A recent viewshed analysis of a line with 47 points on a 20.2mb uncompressed geotiff image on a Pentium II 400Mhz PC with 64mb of RAM (our standard laboratory machine) took roughly 12 hours to complete. The same viewshed operation took less than 8 hours to complete on a Pentium III 450 Mhz with 256mb of RAM (considered a server-type machine for the purposes of this discussion).

It is not always possible to provide server-like processing power on the standard desktop machine, and therefore, for processor intensive operations, one should consider the possibility of performing the same operation on a remote machine with more powerful resources. However, the main consideration would be the time taken to upload the data from the client machine to the machine where it is to be processed, and the time taken to download the result once the calculation has been completed.

The data source used by the viewshed tool was a 20.2mb GEOTIFF image that could have been compressed to 2.8mb, for transmission over the Internet, using the ZIP compression algorithm. The resultant image would be roughly the same size. If one assumes a modest transfer speed of 5kb/sec, the time taken to download the 2.8mb input and upload the 2.8mb result is less than 20 minutes in total, including the time taken to compress and decompress the GEOTIFF data. Even without compressing the original dataset, if one calculates the time taken to transfer the data to the server, and the resultant image back to the client, it would still have been worthwhile sending the data to the remote machine for processing (8 hours versus 12 hours). In addition, the client's machine is free for further processing during this time.

Apart from the potential performance benefits illustrated above, this approach also provides accessibility to different vendor's implementations, possibly using different algorithms, dynamic upgrading, finer licensing granularity as well as different pricing models that may even include guarantees of Quality of Service in terms of speed or accuracy with which the algorithm is executed. For an in-depth discussion on the implications of developing (distributed) component-based applications on licensing and pricing models, refer to *section 5.5* .

4.4.1.2. Reduced bandwidth requirement

While rapid advances are being made in creating faster processors, it is ultimately the legacy Internet networking structure that will create the bottleneck to high-bandwidth multimedia applications [Preston *et al.* 1998a].

A potential solution for reducing the high demands made on Internet bandwidth may lie in processing more information on the server side, to reduce the amount of extraneous data that gets downloaded across the Internet to the client. This optimisation technique has tremendous potential in Internet-based GIS applications because GIS data sets are often extremely large, and the user is only interested in a small subset of this information.

It is also extremely likely that in the near future Data Providers will implement, or at least offer, distributed processing of GIS services that could be used in conjunction with their datasets. The benefits from this coupling include increased customer loyalty, and a decrease in the overall processing time, by negating the need to first transfer the data from the Data Provider, across the Internet, to a separate GIS Service provider before the geoprocessing can begin.

This approach also has the following inherent security benefits associated with it:

?? The user cannot gain unwanted access to the underlying data, i.e. only the GIS tools executed on Server machines have access to the data sets.

?? The user cannot download, pirate and/or reverse engineer a particular GIS tool because all code for executing the tool remains on the server at all times.

It is totally reasonable for a data provider to offer the ability to perform GIS operations on a dataset without ever allowing direct access to the data itself. For example, one may use an Internet-based GIS to select a suburb of a city, add a coverage detailing the road structure together with names, and download the resultant street map in the form of a GIF/JPEG image. The image was created using the data and services provided by a map service provider, but at no stage did the client have access to the underlying GIS data. This allows a data provider to provide inexpensive maps to clients without divulging the underlying dataset.

The sale of GIS data is often accompanied by copyright restrictions that prevent the purchaser from redistributing the dataset in its original form. Therefore, instead of not allowing any type of access to a dataset, highly controlled access could be provided through specific GIS services that do not provide access to the underlying dataset. This approach would allow service providers to offer value-added GIS services that make use of the purchased data set, without contravening the licensing agreement.

A data provider might sell a dataset to other data vendors/GIS consultants for large sums of money, or retain exclusive access to the dataset to maintain competitive advantage. Consider the following scenario: Company A, has a detailed Digital Elevation Model (DEM) of the Eastern Cape that they have invested a lot of time and money in developing so that they can offer consultancy services for the Coega project[1]. They do not wish to make the dataset available for fear of it being pirated, and someone else undercutting their bid for consultancy contracts. However, they also realize that there is money to be made from allowing other GIS experts, (hopefully) in different fields, to make use of their dataset. Therefore they develop and deploy a suite of CORBA services that allow clients to make use of the underlying dataset without providing direct access to the data.

Now consider the case where another GIS consultant, needs to perform a viewshed analysis to minimise the visual impact of erecting electricity pylons, but does not have his/her own DEM of the particular region in the Eastern Cape. He/she could either obtain the necessary terrain maps, digitise the area of interest and then use the data in his/her own GIS application (that supported the viewshed analysis operation), or simply use the viewshed service exposed by company A. The exposed service need only allow him/her to specify the bounding co-ordinates of the area of interest, and the envisaged positions for each of the pylons. It would then return the result of the viewshed operation as a GIF/JPEG illustrating the visual impact. Thus company A is able to share their data and generate income without relinquishing exclusive access to their DEM.

An alternative to the implementation of suites of services by data providers, is for data providers to facilitate the use of mobile agents by customers. The mobile agents would be allowed to move the code of a GIS service to the data provider's machine for execution. This reduces the responsibilities of the data provider, e.g. not having to license GIS service software, or ensure that the software provided produces accurate results.

[1] The Coega project is a major industrial development that involves the building of a new deepwater port on the Coega River, near Port Elizabeth, South Africa. For more information refer to the Coega Development Corporation, http://www.coega.co.za

Thus a range of possibilities are available to data providers, depending on their size and their willingness to diversify their core business to include the provision of GIS services. It is highly likely that large data providers might decide to invest in software that will facilitate client access to GIS services in addition to the data that they provide. However, smaller data providers might decide to allow (registered) clients to send mobile agents, which implement specific GIS services, to their intranet to perform a particular operation on their datasets locally.

Appendix A contains further information regarding the use of Mobile Agents in GIS that formed part of our research, but is not directly relevant to the focus of this thesis.

4.5. GIS and Location Services

The recent media focus on mobile devices, and in particular mobile phones, has sparked tremendous interest within industry over the business potential for (Mobile) Location Services. The goal of Location Services[1] [Koeppel, *current*], a combination of Web, wireless communication and GIS technologies, is to allow one to exploit location information anywhere, anytime, and on any device. In its broadest sense, a location service may be thought of as any application or service that extends spatial information processing, or GIS capabilities, to end users via the Internet and/or wireless network.

In contrast to GIS applications, location services are particular applications of spatial and analytic functions found in GIS applications, that filter their content or change their behaviour, based on the user's (specified) location. Thus, location services build on the existing underlying GIS functionality found bundled in current GIS applications.

[1] For more information regarding Location-Based Services please refer to the Location Interoperability Forum at http://www.locationforum.org, and the Open Location Services Initiative at http://www.openls.org

Location services hide the complexity of GIS tools from the user, by providing an easy-to-use interface for a specific service. This interface makes use of one or more GIS tools behind the user's back in order to provide the location-based service, and thus no longer requires that the user be knowledgeable in geography or cartography. In fact, if the user interface of a Location Service is simple enough, very little computer literacy is required for a user to be able to make use of complex GIS operations.

While the development of location services is a big step forward in its own right, it also plays a very important role in highlighting the intrinsic value of location in data, as well as the need to develop re-usable GIS components based on open systems.

Our research focuses primarily on developing a framework that allows the user to create their own customised, scaled-down GIS application, which contains only the tools required for their particular need. However, it could just as easily facilitate the development of small, specialised GIS-type applications that provide location services.

Thus, our RADGIS architecture could also used to create a framework in which one could combine location services to create more sophisticated tools, as well as allowing further processing of the results obtained from location services. For example, one could access a Real Estate Location Service that provided a map showing a number of houses that were for sale, and then overlay a map provided by a traffic routing location service to determine the level of congestion on the roads between each of the houses for sale and one's work premises.

There are many issues that need to be addressed in order for location-based services to be successfully implemented and widely utilised, especially via the Mobile Internet. These include generic hardware issues such as the ability to display 2D and 3D images, limited bandwidth, and the relatively low processing power and small memory capacities of most mobile phones. However, there are also three main back-end issues that hold the key to the successful implementation of location-based services, namely:

1. exposing location-based data and GIS operations,
2. providing a simple mechanism for content developers to create and deliver new location services, as well as to integrate location into existing applications, and

3. creating searching mechanisms based on semantics, rather than simply syntax or text-based searches.

Ideally, location service developers should be able to develop new location services without the unnecessary duplication of existing GIS functionality. Thus the development of location services requires that the GIS tools and data sets be "exposed" efficiently to ensure that the creation of location services is simplified. Some potential solutions to the issue of exposing location-based data and services are covered implicitly in *section 4.3.1* (.geo proposal) and *section 4.6* (Catalog and Registry services). The second issue is touched on briefly through discussions of how DGCML can be used to develop and deploy location services easily (see *section 5.2.5*), and how our RADGIS architecture may be applied to the creation of a client-side framework for integrating location services. However, the notion of searching location-based data using semantics (or location synonyms) is not dealt with elsewhere in this thesis and therefore warrants discussion now.

4.5.1. Searching location-based data

When accessing location services, mobile device users may have their location encoded automatically by the device or the service provider, using a standardised format. However, a location service user may wish to specify a location explicitly. This could be done in a number of different ways, such as by specifying a longitude/latitude pair, town/city name, suburb name, postal code, or telephone area code, and with varying degrees of precision, i.e. city versus suburb. For searches based on location, it may be advantageous to allow a user to match all these references to, or synonyms for, the same geographical location.

In addition, much of the data currently available on the Web contains a relatively high level of intrinsic location-based information. It therefore makes sense to make this existing information available via "intelligent" searches, rather than requiring that all the information be reformatted to include an explicit spatial reference before it is made available for use by location services.

The creation of "intelligent" searches, based on semantics rather than simply on keywords, removes the requirements that:

?? users format their location-based queries using a standardised geocoding, and

?? existing data, which contains implicit location information, be reformatted to include an explicit spatial reference.

An "intelligent" search, based on location semantics, could be implemented by simply searching a database of equivalent location encodings for a given location. The "intelligent" search would therefore be split up into multiple smaller searches based on location synonyms, enabling the use of existing location-based information that might not otherwise have been included in the result set. For example, given a search containing a postal code and a name, the search would also return information about that name based on the location inferred by the postal code, rather than simply returning a list of all documents that contain the name and the postal code explicitly as text.

More sophisticated searches that provide GIS-type "near" or "within" functionality could also be used in conjunction with existing intrinsic location-based data, but these searches would require additional pre-processing and could potentially create a large number of sub-queries.

4.6. Exposing GIS Services efficiently

The inability to discover and access geospatial data and geoprocessing tools in a simple manner, drastically reduces the effectiveness of GIS applications. The next generation GIS model will provide a distributed environment in which data and services are added, updated or removed dynamically. It is therefore necessary to provide a mechanism to reduce the complexity of keeping track of these events, and to allow users to access data and services transparently.

CORBA, DCOM, RMI and EJB provide naming/registry services that allow client applications to locate distributed objects by name. However, other than the use of the CORBA Trader Service, which is a directory service, these technologies do not provide a mechanism to register objects with attributes or meta-information. Such information is useful for distinguishing objects with the same name, or for performing queries based on distributed objects' properties.

Catalog and Registry services allow the registration of geospatial data and services by data and service providers. The use of a Catalog or Registry service to discover data or services, makes it is possible to change the location of where the data is stored or where a service is executing by simply editing the appropriate entry in the Catalog or Registry. It is not necessary to update all the clients that make use of the data-source or service. This change in the location of where the data is stored or where the geoprocessing is taking place is therefore transparent to the end-user.

Much work has been done to allow users to index and search spatial data, such as Catolog services. However, very little work has been done to provide a unified mechanism for registering GIS services, which have been developed using different distributed object technologies, with their associated meta-information. Such a mechanism is urgently needed, as it is imperative that distributed GIS services are advertised effectively and efficiently. We believe that there are two mainstream technologies that have the potential to fulfil this role, namely LDAP and UDDI.

4.6.1. LDAP

The Lightweight Directory Access Protocol (LDAP) is an extensible client-server protocol and information model that allows one to access and manage information in a tree-structured database [Roman 2000]. Each entry in an LDAP server has a distinguished name that allows easy identification, and stores associated information as attributes. Each attribute has an associated type and one or more values.

One of the tremendous benefits of using LDAP is that one can create one's own object types and attributes. This allows you to use LDAP directories for a wide variety of tasks, including the ability to register distributed components and services which will be extremely useful for advertising our GIS services developed using DGCML. For more information on LDAP, please refer to [Wahl *et al.* 1997]

4.6.2. UDDI

The Universal Description, Discovery, and Integration (UDDI) standard is a repository-based directory service for sharing business information, which includes the ability to find and access the applications or Web Services they expose. The services registered with a UDDI server can be deployed in a private (internal) or public (external) manner depending on whether one wishes to share the services within an organisation only, or throughout the Web [IBM 2001]. For more information on UDDI, please refer to [UDDI.org 2000].

However, according to Cimetiere [2001] the ambitions of UDDI might be too broad for it to succeed, and because a Web Service is simply another resource, it could just as easily be registered with existing technologies such as an LDAP server. Therefore, while we view UDDI as a promising technology for exposing Web Services, the simplicity, efficiency and proven track record of LDAP currently make it a more suitable candidate for advertising GIS services in our RADGIS architecture.

4.7. RADGIS - Our Approach

In order to overcome many of the problems associated with current GIS architectures, outlined in *section 1.1* and *section 4.1*, we have developed a framework that allows interoperable GIS services to be incorporated into a highly-configurable client GIS application at runtime. The proof-of-concept distributed GIS application we have developed to demonstrate the feasibility of such an approach has been named RADGIS, due to its ability to facilitate the Runtime Application Development of Geographic Information Systems[1].

One of the objectives of the project was to avoid vendor specific solutions, and to promote interoperability through use of open standards wherever possible. The RADGIS architecture therefore integrates interoperable services and geodata models based on OpenGIS® standards developed by the Open GIS Consortium, that have been presented earlier in this chapter. In particular, we decided to adopt the use of GML as the geospatial data transfer format, and the OGC's Services Architecture to develop runtime-extensible client-side GIS and Location Service applications that are based on vendor-independent, interoperable GIS components.

---

[1] Chapter 5 details how the RADGIS application has been implemented.

In the past there have been many efforts to facilitate the accessing of distributed ge ospatial data from different vendors. However, our RADGIS client architecture is an attempt to allow users to not only make use of geospatial data from different data providers, but also distributed GIS services. Because RADGIS is a runtime-extensible architecture that can invoke methods on distributed GIS tools, it is a hybrid of the client-side and server-side GIS architectures discussed in *section 4.1.1* and *section 4.1.2*, which provides a high level of location-transparency for accessing both data and services. Using this approach, the RADGIS architecture is able to draw on the advantages provided by both client-side and server-side GIS architectures, as well as overcome many of their respective disadvantages[1]. The resultant advantages and disadvantages of the RADGIS architecture are listed in *table 4.5*, and more comprehensive summary of benefits is presented in *section 6.6*.

Advantages to RADGIS Architecture

?? Significantly reduces application bloat by providing a small, extensible client framework.

?? Adherence to OpenGIS® standards to ensure interoperability with future OpenGIS®-compliant components.

?? Platform independence using Java, CORBA and EJBs (CORBA also provides language independence).

?? High level of location transparency when accessing distributed data and services.

?? Significant GIS functionality can reside on distributed servers.

?? Data sets can be obtained from multiple distributed data servers.

?? Georeferenced raster and vector data can be used, i.e. not restricted to using static images, e.g GIF and JPEG.

?? Highly customisable GUIs for individual services.

?? Not restricted to single-click operations.

[1] See tables 4.1 and 4.2 for the advantages and disadvantages of server-side and client-side architectures.

- ?? Centralised administration of data and GIS services is possible, reducing total cost of ownership.
- ?? Excellent performance for operations that occur locally.
- ?? Ability to perform processing remotely can eliminate need to download large data sets across the Internet.

Disadvantages to RADGIS architecture

- ?? Cannot be accessed with standard Web browser
- ?? Requires users to obtain additional software
- ?? Users must wait for software to download. However, since most of the services may be accessed remotely, often only the small DGCML descriptors for a service need to be downloaded.
- ?? Overall performance can be low with large data sets, but the RADGIS architecture promotes a distributed model that would allow the RADGIS client distribute the workload for certain operations.

Table 4.5. RADGIS Architecture advantages and disadvantages

A similar vision is shared by the Defence Science and Technology Organisation (DSTO). As part of the development of their Geospatial Service Segment Architecture (see *figure 4.6*), they are also exploring ways of using software components to build large systems, instead of designing monolithic applications that are obsolete before they are deployed. They have decided to make use of CORBA and have adopted the OpenGIS® Simple Features for CORBA implementation specification as their interface for accessing Geospatial Information [Davis 2000].

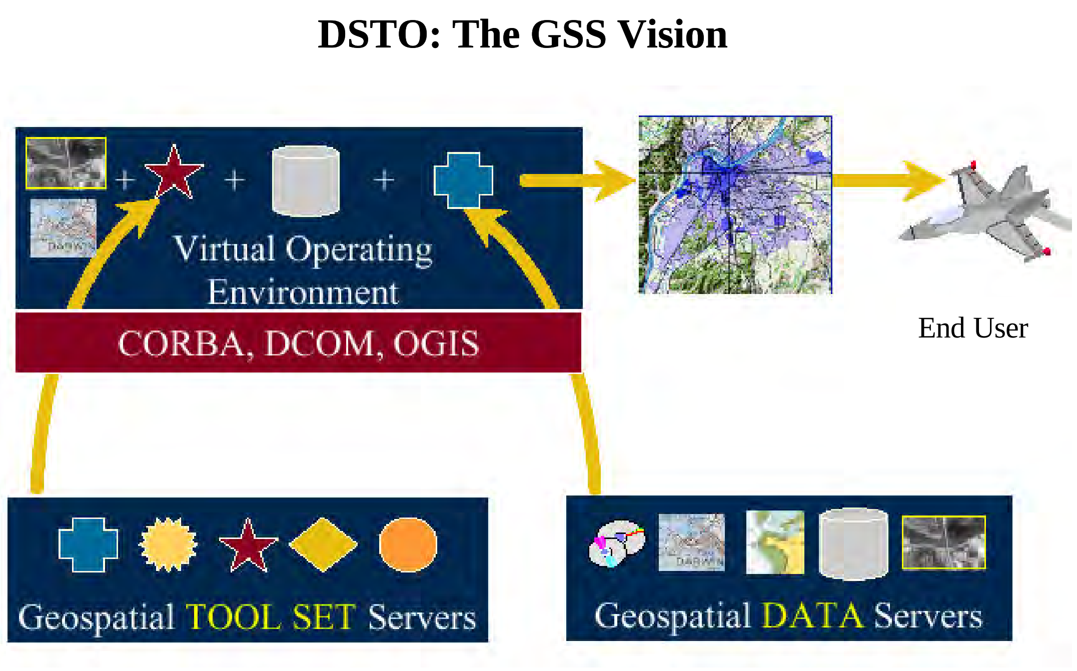


Figure 4.6. The Geospatial Services Segment Vision [Ekins and Davis 1999]

While traditional GIS applications are well-suited to GIS experts, there is a growing realisation that there are a number of users, in a variety of fields, who require spatial analysis tools similar to those found in GIS applications. These users generally require only a small subset of the functionality provided by traditional GIS applications, and do not necessarily wish to become experts in GIS in order to make use of GIS applications. Our RADGIS architecture therefore allows the development of small, client-side GIS applications that can be customised to suite different user domains, e.g. hydrology, geology, forestry, and according to different user's level of expertise. However, as the user's requirements change, or their level of expertise increases, it is possible for the user to add new functionality to the RADGIS client as well as to customise the services according to his/her needs or preferences.

We have developed a novel approach to the integration of interoperable GIS components at runtime, using client-configurable XML descriptors for deploying GIS services, based on our Distributed GIS Component Markup Language (DGCML). The GIS components may be implemented as Java objects that are packaged in JARs, which are downloaded and run as client-side services, or as distributed objects that reside on server machines, e.g. Common Object Request Broker Architecture (CORBA) Objects or EJB's. DGCML is used to wire-together local and remote GIS components to create services that can be added and removed by the GIS application at runtime. In addition, these services can be edited using a simple text editor, at runtime, if customisation is desired.

Therefore, the RADGIS architecture addresses the following problems found in most traditional GIS applications:

?? Application bloat – It is possible for the RADGIS client to initially implement only the very basic GIS services needed for data visualisation. However, its extensible nature allows users to add and remove tools, as and when required. Thus the user does not have to purchase a very large (and generally expensive) GIS package, which contains a great deal of functionality that will never be used.

?? Usability – The ability to add tools is as simple as selecting the required tools from a menu or web-page. The novice user can make use of a service without any knowledge of how the service has been created from distributed components, while the expert user has the ability to customise the service by simply editing the DGCML descriptor file.

?? Interoperability – RADGIS provides the ability to add services composed of interoperable distributed components developed independently by different vendors.

?? Location Transparency – RADGIS allows the processing of data to be performed wherever it makes most sense, without the user necessarily being aware of the physical location of the tool that is performing the processing.

The ability to specify GIS services using DGCML allows method-calls invoked by a particular service to be made on objects executing locally or on objects implemented as CORBA objects or EJBs residing on remote machines. If alternate codebases are provided for a particular service, it is possible for the RADGIS system to elect which codebase to use, based on the location of the data to be processed. This is advantageous because, even if certain GIS functionality has been installed locally as part of the RADGIS application, should it make sense to perform that operation on data that resides at a remote location, it is possible for the user to invoke a remote implementation of that service transparently. An example of where this may be mandatory, rather than simply for convenience or a matter of optimisation, is when one wishes to obtain data from a data provider who does not want to provide direct access to the data, but only allows access to the data through certain GIS operations that return static images or data at a particular resolution. Therefore RADGIS affords the user a high level of location transparency when performing geoprocessing operations.

Figure 4.7 below details the use of our proof-of-concept GIS application (RADGIS) based on highly interoperable (distributed) components developed using our Distributed GIS Component Markup Language (DGCML), the OpenGIS® Services Architecture, and the Geographic Markup Language (GML).

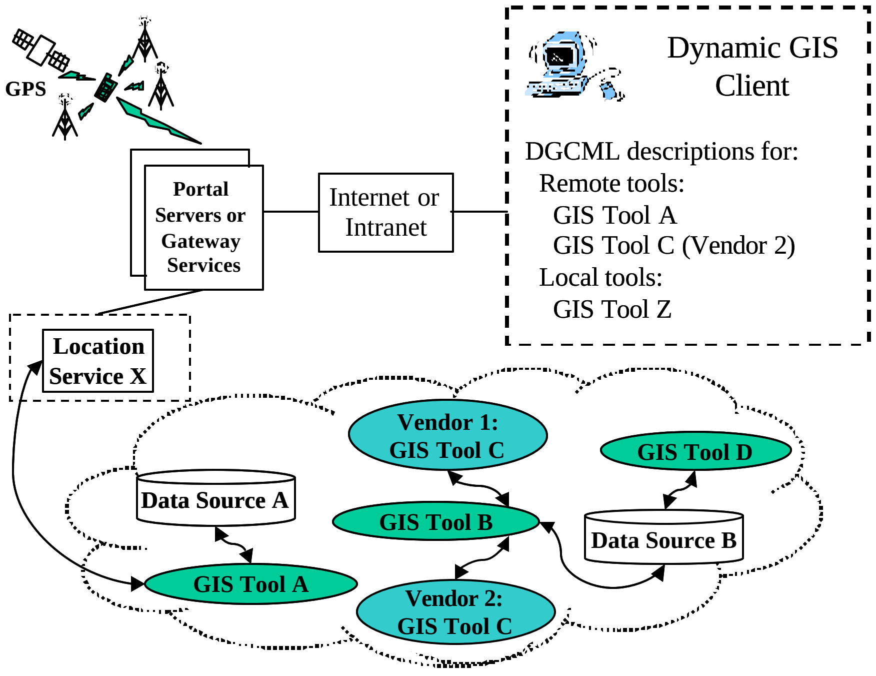


Figure 4.7. RADGIS: A highly distributed component-based GIS

It is generally accepted (see *section 1.2* and *section 1.3.3*) that future GIS applications will become distributed, disaggregated, decoupled and interoperable. The implementation of distributed interoperable components will ensure healthy competition between vendors and will allow end users to easily replace the implementation of a particular service with another superior or cost-effective service. The use of interoperable components also facilitates the development of highly scalable systems, and the packaging of particular services, resulting in lower costs and products tailored for specific end-users' requirements. Clearly the RADGIS architecture embraces these ideals, and goes further by providing a runtime-extensible framework that allows users to add services based on open standards to the client application, which were not known about at compile-time.

4.7.1. Use-case scenarios

With the aid of *figure 4.7*, we will now sketch a few typical use-case scenarios to illustrate the flexibility and usefulness of the RADGIS architecture, over traditional GIS applications.

For the first scenario, let us assume that during the course of performing a particular GIS operation, a user discovers that they do not have a particular GIS tool necessary to complete the operation. If the user was using a traditional GIS application that did not support that particular operation, his/her options might include outsourcing that operation, obtaining a different GIS package, or downloading and installing an additional module, which implements that GIS tool, for their existing GIS. However, using the RADGIS architecture, the user would be able to search a GIS tool repository, and depending on whether the tool was required to execute locally or remotely:

?? add that tool to the local client-side GIS (e.g. GIS Tool Z). This is similar to downloading an additional module except that the vendor of the tool need not be the same as the developer of the client-side GIS framework, and that RADGIS allows for greater levels of customisation and integration than might otherwise be possible when installing additional modules.

?? add a reference to a remote tool (e.g. GIS Tool A or C) which would be invoked transparently through the user interface which would be generated from the DGCML descriptor for that tool.

For the second scenario, consider once again the viewshed analysis example, given in *section 4.4.1.1*, which illustrated the ability to make use of distributed processing on a server machine to increase the speed of performing a particular operation. If, for example, one extended the scenario to assume that the user wished to perform the viewshed analysis of 47 points (representing pylon positions) for three different positions, it would generally be necessary to perform three sequential viewshed analysis operations. Using a standard laboratory machine, as described in the example in *section 4.4.1.1*, this would take approximately 36 hours of processing (3 x 12 hours per viewshed analysis) to complete. However, because each viewshed analysis is independent of each other, the ability to make use of 3 different distributed viewshed analysis services simultaneously, i.e. parallel processing, would allow the user to compare the three resultant images in approximately 8 hours (assuming each analysis is performed on a server that provides the same performance as obtained in the example in *section 4.4.1.1*).

For the final scenario, consider the ability of the RADGIS architecture to allow a user to make use of a distributed GIS tool implemented by a Data Provider that does not wish to allow direct access to the underlying data set. For example, a user wishes to make use of data (e.g. stored in Data Source A), but the only access to it is using GIS tools (e.g. GIS Tool A) implemented by the Data Provider. The ability to download a DGCML descriptor that makes use of that GIS tool allows the close integration of that tool with the RADGIS client, and provides the ability to invoke the necessary functionality via a GUI generated by the RADGIS client, according to the DCGML descriptor for using that tool.

4.8. Summary

This chapter has presented work being done by major standardisation bodies and large corporations in order to ensure interoperability of GIS components and applications, as well as a brief overview of GIS architectures, and products currently available, together with their associated advantages and disadvantages.

The relatively new research topic of Location Services was introduced, as it will have a major influence on the architecture of future GIS applications due to its heavy reliance on re-usable GIS components. We also highlighted some of the backend issues that will have a significant influence on the future success of location services. In particular, we stressed the need to make geodata and geoprocessing components easily accessible through initiatives such as the *.geo* proposal, and the use of Catalog and Registry services.

A number of problems with current GIS architectures were highlighted that illustrated the need to develop a flexible GIS architecture that can be adapted to the requirements of users who work in different application domains, with varying levels of competency. These factors were the motivation for the development of our RADGIS application, which facilitates the use of distributed data and GIS services in a location transparent manner. The RADGIS architecture was described, together with its associated benefits and how it overcomes many of the problems associated with traditional GIS architectures.

In the next chapter we discuss how the RADGIS architecture was implemented in more detail. In particular we focus on the development of our Virtual Grahamstown data set using VRML and the visualisation of this 3D geospatial data, which formed the basis of our initial research into Virtual GIS, as well as the development of GIS services using DGCML, which can be integrated into the RADGIS client at runtime.

Chapter 5
Implementation of the RADGIS Application

"The best way to predict the future is to invent it."
Alan Kay

A GIS must support both computational and display facilities as it allows the user to compute and display information about geographic features. Therefore, in the development of our RADGIS application, our research has focussed on both the visualisation of 3D geospatial data as well as the runtime integration of GIS services, for the analysis of that data, which may be executed locally or on a remote GIS server.

This chapter will therefore present a detailed description of how we implemented our RADGIS proof-of-concept application, and will provide insight into:

?? the development of our "Virtual Grahamstown" 3D data set using VRML;
?? the design considerations that resulted in our choice of Java3D for rendering the 3D data;
?? the implementation-specific details of our DGCML meta-language;
?? how one could use DGCML to create a location service; and
?? how we have made use of Java's Reflection API to allow the runtime addition and execution of GIS services that are developed and deployed using DGCML.

The factors that influenced the design of RADGIS as a client-side application, rather than a Java applet, have much to do with problems associated with the visualisation of 3D geospatial data. The use of VRML as a mechanism for facilitating the creation of Web-based Virtual GIS was the focus of much of our early research. Therefore this chapter presents two major areas of research that we have undertaken. The first section presents the issues associated with the visualisation of 3D geospatial data, while the second section focuses more on the implementation details of the runtime extensible framework provided by RADGIS for the integration of GIS services that have been developed and deployed using DGCML.

5.1. 3D Visualisation of Geospatial information

Most current Geographic Information Systems are static 2D, map-based systems with non-interactive response rates when displaying high-resolution maps. More recently, however, there has been a trend towards implementing interactive 3D GIS applications with the aid of improvements in 3D graphics software and hardware, efficient new terrain visualisation algorithms and, possibly most importantly, the tremendous interest in using VRML to display geospatial information on the Web.

This move towards greater use of 3D spatial data, and the inclusion of temporal data in the quest for the development of a spatiotemporal Geographic Information Systems, is a logical step in the evolution of Geographic Information Systems, providing a graphical insight into, and graphical analysis tools for analysing, large volumes of spatiotemporal data.

5.1.1. Rendering of 3D spatial information

Within the GIS domain there are two main binary formats for representing 3D geospatial information, namely Digital Elevation Models (DEM) and Triangular Irregular Networks (TIN). Since the mid 1990s, three-dimensional or Virtual GIS on the Web [Rhyne 1997] has been regarded as a promising alternative to traditional GIS applications. One of the obvious requirements for implementing such systems is the need for a simple mechanism for viewing 3D geospatial data without requiring a GIS client application.

The use of VRML for distributing and visualising geospatial data has received a lot of attention due to its simplicity, cost-effectiveness and wide accessibility [Kim *et al.* 1998] since it is an open standard (ISO/IEC 14772). This can be seen by the large number of research papers that have been published on this topic, such as [Fairbairn and Parsley 1997] [Dykes *et al.* 1999] [Rhyne 1997]. In addition, due to the popularity of VRML for modelling geospatial data, there are now a number of GIS products and conversion utilities available that are able to convert information stored as DEMs or TINs to VRML.

5.1.1.1. VRML and Cartography

The Virtual Reality Modelling Language (VRML[1]) is a vocabulary for the animation and modelling of 3D geometric shapes. It has become a widely accepted standard for interactive 3D information interchange on the WWW. VRML allows one to incorporate many different types of data, including text, diagrams, graphs, audio and video, together with 3D models, seamlessly within the 3D world. This, together with the ability to communicate 3D worlds across the Web, provides significant flexibility for the sharing of three-dimensional data sets, and enables VRML to provide an open standards alternative for displaying geospatial information. (For more information about VRML, refer to [VRML97] [Nadeau 1997] [Marrin and Cambell 1997])

A number of projects have been undertaken that successfully make use of VRML to visualise geographic data, such as [Fairbairn and Parsley 1997] [Martin and Higgs 1997] [Rhyne and Fowler 1998]. One such example is that of Buziek and Hatger [1998], who developed an interactive spatiotemporal 3D animation, using depth and tide information for the Elbe estuary over a 12 hour period, in order to investigate the cartographic potential of VRML for geo-referenced cartographic applications. According to Buziek and Hatger [1998], VRML is suitable for modelling geo-referenced 3D worlds, but also has some limitations. The most serious of these limitations is that VRML currently only supports 32 bit float values, which limits the precision to 7 digits. This accuracy is not adequate for geodetic coordinates, and thus world coordinates have to be shortened.

[1] All references to VRML in this dissertation refer to the ISO/IEC 14772 or VRML97 specification.

Another major limitation of using VRML for cartographic representation on the Web is the large amount of data to be transferred. However, VRML provides a number of optimisation techniques that can be employed to help overcome or at least reduce the effective bandwidth required for visualising data sets over the Web, including compression, inlining, level of detail (LOD) management, and ShapeHints[1]. In addition, a number of nodes[2] have been developed by the GeoVRML task group, which looks specifically at the representation of geographical data in VRML, to facilitate the efficient visualisation of large terrain models in 3D.

5.1.1.2. GeoVRML

Like VRML, GeoVRML is an official working group of the Web3D Consortium. However, its focus is on extending VRML as well as developing methods and tools for the representation of geographical data from disparate servers across the web, possibly generated from different sources, at different resolutions, and specified in different coordinate systems [Reddy *et al.* 2000c].

According to Reddy *et al.* [2000b], GeoVRML addresses most of the concerns raised by Dykes *et al.* [1999] regarding support for cartographic applications in VRML. It does this by overcoming the shortcomings of VRML, and providing additional functionality with respect to representing geospatial information, such as [Reddy *et al.* 1999a] [Reddy *et al.* 2000b]:

?? Support for data in various geospatial coordinate systems;

?? Scalability – facilitating the integration and use of data from large geospatial databases that are distributed over the web.

?? Providing the capability of representing large quantities of terrain and other related data;

?? Preservation of the original geographic data;

?? Management of multiple levels of detail of geospatial data; and

[1] See *Appendix B* for a more in-depth description of these optimisation techniques.

[2] See *Appendix C* for brief descriptions of these GeoVRML nodes.

?? Accuracy – overcoming the limitations of VRML's single-precision floating-point support.

This is achieved through extensions to the VRML syntax (using the VRML PROTO node), which implement nodes that support the efficient and accurate representation, as well as rendering of large terrain models. More information about the GeoElevationGrid, GeoCoordinate, GeoLocation, GeoOrigin, GeoLOD, GeoInline, GeoPositionInterpolator, GeoTouchSensor, GeoViewpoint and GeoMetadata nodes can be found in *appendix C*.

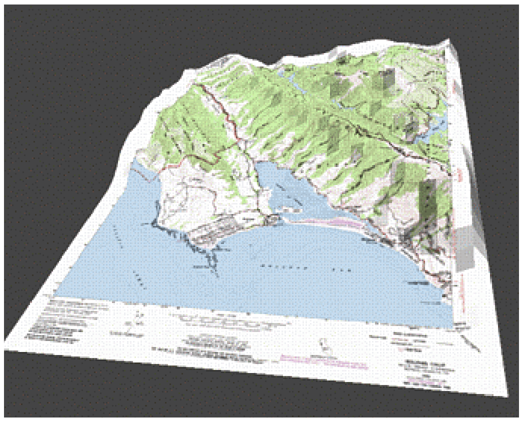
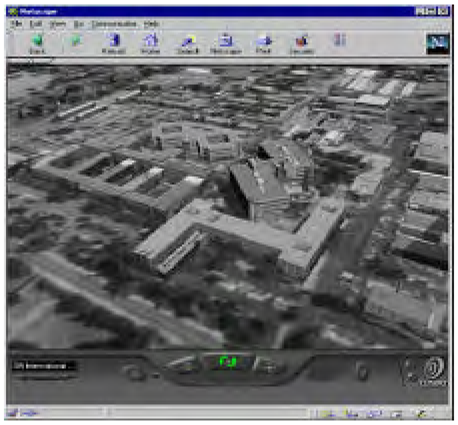



Figure 5.1. Screenshot of scenes developed using GeoVRML [Reddy 2000]

The work that has been undertaken by the GeoVRML workgroup in the development of GeoVRML, and its acceptance as a Web3D Consortium "Recommended Practice" goes a long way towards being able to accurately represent 3D geographic information that may be visualized across the web using a standard VRML browser (with Java support).

5.1.1.3. X3D

The X3D (Extensible 3D) format, formally known as VRML-NG (VRML Next Generation), is an open 3D graphics specification for the Web that extends the functionality of VRML97. The main objectives of the X3D working group include ensuring backwards compatibility with VRML97, and the integration of XML. X3D will enable the creation and deployment of visually rich 3D graphics that can be viewed using small, lightweight web clients with advanced 3D capabilities. In addition, due to the X3D working-group's close interaction with the MPEG-4 group's ongoing 3D integration activities, X3D will add high-performance 3D to broadcast media [Web3D Press Release 2001][X3D FAQ 2001].

X3D adopts a component-based architecture that supports the extension of the X3D vocabulary. This enables the development of extremely compact 3D clients that can be extended with plug-in components/profiles [Web3D Press Release 2001]. There are currently a number of profiles that have been developed, including [X3D FAQ 2001]:

?? The X3D Core profile (X3D-1) – contains a reduced set of VRML nodes that only support simple non-interactive animation and is intended for the widest-possible adoption of X3D support.

?? The X3D-2 profile – is a larger profile that covers the full VRML specification in order to provide support for fully interactive worlds and existing "rich" VRML content.

?? The GeoVRML profile – which contains support for the GeoVRML nodes listed in *section 5.1.1.2*.

Once GML (see *section 4.3.2*) supports 3D geospatial data, it is inevitable that a GML to X3D stylesheet will be written that will allow GML content to be converted to the X3D format for presentation in an X3D browser. Thus, utilising the necessary XML stylesheets, X3D profiles and supporting Java classes, it should become possible to view GML, VRML and GeoVRML geospatial data in an X3D browser. This will be particularly useful for the 3D data visualisation approach we have adopted in the RADGIS architecture (see *section 5.1.2*), which currently makes use of a VRML loader for Java3D, but which can easily be extended to make use of an X3D loader for Java3D when one becomes available.

5.1.2. RADGIS: Data sets and Visualisation

The general framework for our Runtime Application Development GIS (RADGIS) architecture was outlined in *section 4.6*. This section, however, provides specific implementation details regarding the development of our data set in VRML and the use of Java3D for rendering the data, as well as explaining the factors that have contributed towards these choices.

When we originally started our research, our focus was on the development of a Virtual GIS application that was easy to use, and could be used by anyone who had access to a Web browser, i.e. no proprietary software. At that stage VRML was the only open-standards 3D visualisation platform for the Web. Thus, when we created our model of Virtual Grahamstown (see *section 5.1.2.1*), we made use of VRML.

However, VRML itself is fairly limited in the types of operations that it can perform. This is because VRML is not a general-purpose programming language. It is simply a vocabulary for marking-up scene descriptions, which runs entirely within its plug-in environment. In order to make VRML more powerful, it is necessary to make use of the Java programming language to program custom application logic that can interact with the VRML scene.

There are two specified methods for using Java with VRML that are supported by a number of VRML plug-ins. One method is through the use of Script nodes, and the other is though the use of the External Authoring Interface (EAI). It was the latter that was of primary interest to us, as it provided the desired mechanism for creating custom visualisation Java applets that were able to manipulate the VRML Scene Graph in the VRML plug-in, and provide customised user interaction and increased functionality.

The interaction between Java code and the VRML plug-in provides a powerful mechanism for overcoming some of the shortcomings of using VRML by itself. An applet can be used to effect changes in a VRML world, by providing control over the contents of a VRML browser, embedded in a web page. It does this through the Web browser's plug-in interface, such as Netscape's LiveConnect or Microsoft's ActiveX/COM, which allows objects embedded in a web page to communicate with each other. While VRML plug-ins are not required to implement the EAI to achieve VRML 2.0 compliance, several plug-ins have implemented it.

Unfortunately, there are a number of problems with using the EAI, which include implementation differences between Web browsers, and different versions of a particular Web browser, as well as whether or not the VRML browser plug-in supports the EAI. The most serious issue for the development that we originally planned to undertake, which implemented the GIS client as an applet, was the inability for the applet to connect to the VRML scene graph if the applet was being run using the Java plug-in.

As we expanded our goals from simply providing a mechanism to visualise 3D geospatial data, to include the ability to integrate distributed GIS services implemented using CORBA Objects and EJBs, we soon ran into implementation issues due to the lack of support for "new" features in the JVM's of Web browsers. The JVMs supplied with Web browsers are very seldom up-to-date with the latest JVM from Sun, and most only support version 1.1.x of the JVM. This means that some browsers do not support RMI over IIOP, Java Foundation Classes, e.g. Swing, or "advanced" GUI features such as Java's Drag-and-Drop functionality, and the Accessibility API. The use of the Java plug-in therefore became necessary to utilise the latest features of an SDK release, rather than wait for them to be incorporated in an upgrade to the browser JVM. The plug-in approach was also attractive, as it did not dictate what Web browser or particular version of the browser the user had to install in order for the GIS client applet to run correctly.

However, it soon became apparent that what we were trying to achieve was possibly best implemented as a client-side application, which incorporated a dedicated 3D browsing environment, rather than an applet that accessed an external VRML browser. This approach overcame the security restrictions placed on applets, and the inability for an applet executing using the Java plug-in to access the Scene Graph in the VRML browser. It also simplified the integration of local and remote services implemented as DGCML descriptor files that require the ability to read and write configuration and helper files, and make network connections to multiple machines on which the distributed services were running.

Our current proof-of-concept distributed spatiotemporal GIS *client application*, RADGIS, has therefore been written in Java, for platform independence, and uses Java3D for the visualisation of the 3D VRML worlds, thanks to the availability of a VRML loader for Java3D. This decision has proved very useful, as it has also ensured that support for GML, or X3D scenes generated from GML using a GML to X3D style sheet, could be added by simply adding the appropriate GML or X3D file-format loader for Java3D.

Currently, neither the GML nor the X3D specifications are complete. While an early version of an X3D file-format loader for Java3D does exist [Brutzman *et al.* 2001], it is not possible to test the overall system, based on GML encoded spatial data, as GML does not currently support 3D features and therefore neither a complete GML file-format loader for Java3D, nor GML mapping to X3D currently exists.

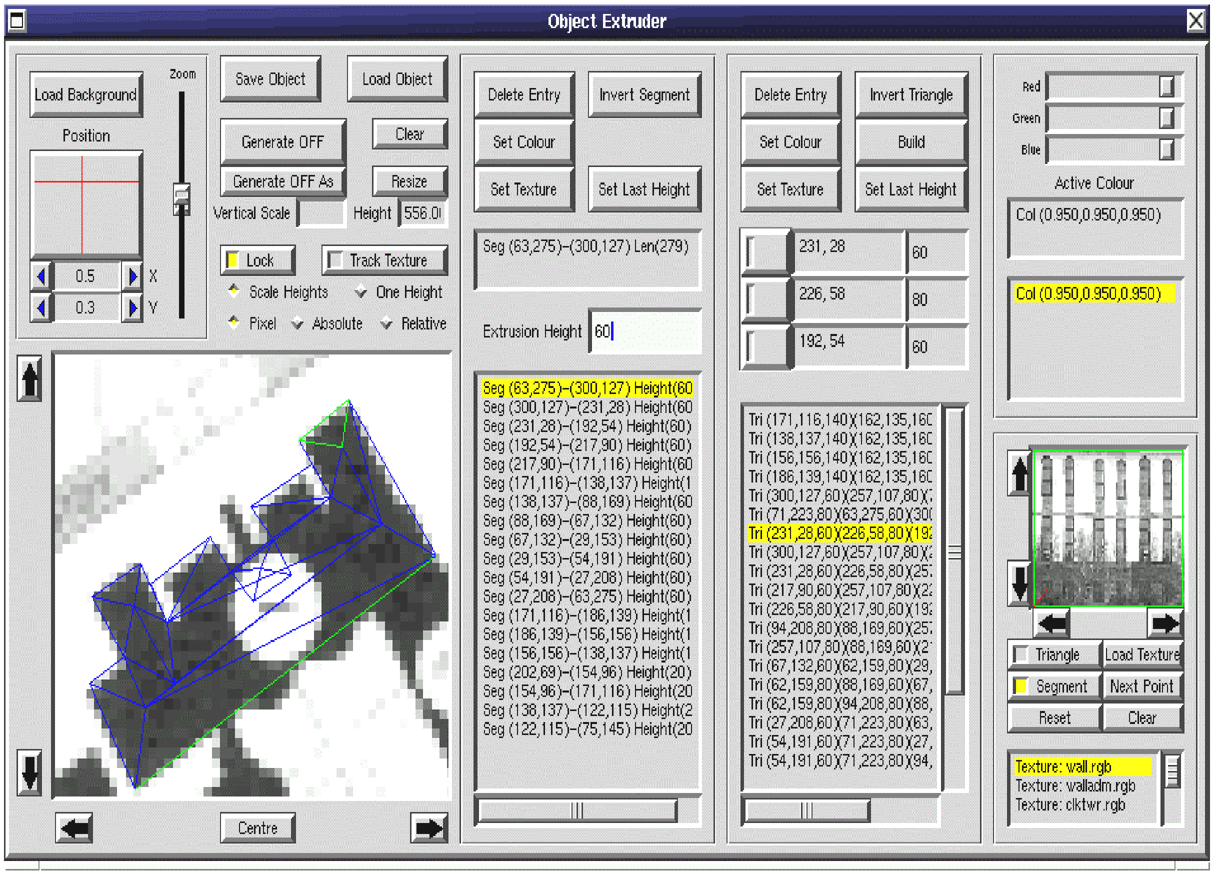
Since the focus of this research is not on the rendering of 3D geospatial data, we have decided not to change file formats until a fully implemented version of the X3D loader becomes available. However, we recognise the short-comings of VRML with respect to geo-referencing geospatial data and the limitation of single-precision floating-point support, and wish to highlight the extensive work being done by the GeoVRML working group to support geospatial data rendering based on the VRML and upcoming X3D specifications.

5.1.2.1. Creation of the Spatial Data Set: Virtual Grahamstown

In order to create the VRML data sets, we made use of an extrude utility written by Bangay [1997]. This allowed us to scan in maps of Grahamstown and then "digitise" features such as roads and buildings. Contour maps with roads and outlines of the floor plans of buildings, from the Town Planning Office, were used for most of the layout and geo-referencing of the roads and buildings. However, some smaller, more detailed maps of the Rhodes University campus were also used for creating more accurate representations of buildings on the university campus. Thus a number of maps covering adjacent areas were used in the creation of this model since no one map was capable of providing the view of the town in the required detail.

Once the maps had been scanned in, the outlines of the buildings were used as a template for specifying the arrangement of the walls. Using the specialised extrude tool (see *figure 5.1*), developed specifically for the purpose of creating the objects in the virtual world, these walls were raised to the appropriate levels, and then a roof was added. The resultant three-dimensional volume represents the outside of the building, and is stored as a set of polygons that make up the walls and roof of the virtual building.

The two polygon primitives that were used to create the buildings were vertical rectangles, used to form most of the walls, and triangles that were used to construct the roads and roofs, and any other specialised feature. Since the outline of the building is captured as a sequence of line segments whose end points normally overlap to produce a closed horizontal polygon, it is useful to ensure that the end points of overlapping segments are not needlessly duplicated. Thus points within a small distance from each other are identified and merged to form a single common vertex.



```
Height 556.0
X 0.5
Y 0.3

Seg (63,275)-(300,127) Len(279)

Extrusion Height 60

Seg (63,275)-(300,127) Height(60
Seg (300,127)-(231,28) Height(60
Seg (231,28)-(192,54) Height(60)
Seg (192,54)-(217,90) Height(60)
Seg (217,90)-(171,116) Height(60
Seg (171,116)-(138,137) Height(1
Seg (138,137)-(88,169) Height(60
Seg (88,169)-(67,132) Height(60)
Seg (67,132)-(29,153) Height(60)
Seg (29,153)-(54,191) Height(60)
Seg (54,191)-(27,208) Height(60)
Seg (27,208)-(63,275) Height(60)
Seg (171,116)-(186,139) Height(1
Seg (186,139)-(156,156) Height(1
Seg (156,156)-(138,137) Height(1
Seg (202,69)-(154,96) Height(20)
Seg (154,96)-(171,116) Height(20
Seg (138,137)-(122,115) Height(2
Seg (122,115)-(75,145) Height(20

231, 28    60
226, 58    80
192, 54    60

Tri (171,116,140)(162,135,16
Tri (138,137,140)(162,135,16
Tri (156,156,140)(162,135,16
Tri (186,139,140)(162,135,16
Tri (300,127,60)(257,107,80)(
Tri (71,223,80)(63,275,60)(30
Tri (231,28,60)(226,58,80)(19
Tri (300,127,60)(257,107,80)(
Tri (231,28,60)(226,58,80)(25
Tri (217,90,60)(257,107,80)(2
Tri (226,58,80)(217,90,60)(19
Tri (94,208,80)(88,169,60)(25
Tri (257,107,80)(88,169,60)(2
Tri (67,132,60)(62,159,80)(29,
Tri (62,159,80)(88,169,60)(67,
Tri (62,159,80)(94,208,80)(88,
Tri (27,208,60)(71,223,80)(63,
Tri (54,191,60)(71,223,80)(27,
Tri (54,191,60)(71,223,80)(94,

Col (0.950,0.950,0.950)
Col (0.950,0.950,0.950)

Texture: wall.rgb
Texture: walladm.rgb
Texture: clktwr.rgb
```

Figure 5.2. Screenshot of Extrude utility

Each line segment in the outline of the horizontal polygon forms a vertical rectangular wall in the final virtual building. An additional parameter must therefore be specified to represent the height of the wall, and the absolute height of the building as a whole (relative to sea-level or any other convenient reference point) can then be specified as a base offset for the entire building.

Since each object will be used together with many other objects in a virtual world where the speed of interaction is important, there is a trade-off between the level of detail (LOD) of the structures and the final rendering speed. It is therefore assumed that a simplified outline for the building is sufficient for most objects. However, should more detail be required for a particular object, that object can also be created at a greater LOD and used together with VRML LOD and InLine statements to provide a higher level of detail for that object (see *appendix B*).

Once the structures had been created, it was then possible to add colour and to map textures to the buildings to add realism to them. See *figures 5.3* and *5.4* for sample snapshots from Virtual Grahamstown, which illustrate the realism achieved by adding colour and textures.

Colour, as opposed to texture mapping, was used where the significance of the building was low, or the building was being drawn at a lower level of detail, or where the texture of an area would have been relatively plain. This reduced the scene complexity and increased the speed of rendering the scene.

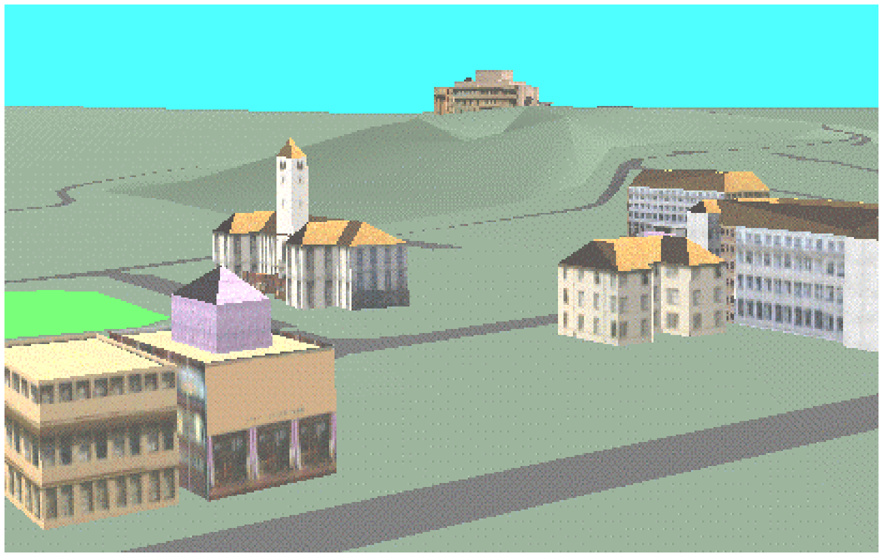


Figure 5.3. A snapshot of selected buildings on the Rhodes University campus (looking towards the Grahamstown Monument)

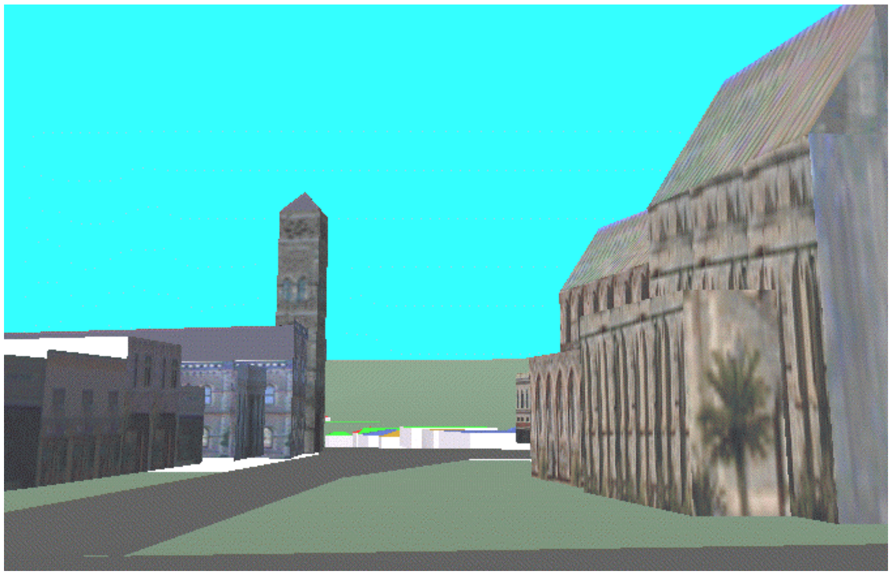


Figure 5.4. A snapshot of the Grahamstown City Hall(Left) and Cathedral(Right)

The use of textures for important landmarks, however, is almost mandatory, and the resulting realism is extremely high in comparison with simply using colour. The texture maps for these buildings were obtained by taking still-shots of the actual buildings using a video recorder. While the use of texture mapping is relatively resource intensive, there are ways to minimise the effect of using textures within a scene, and to reduce memory usage at rendering time. These include:

?? Repeating the texture both horizontally and vertically, allowing areas with repeated features to be efficiently generated from only a single instance of the texture of that feature.

?? Selecting active areas of the texture allows reuse of the texture maps for cases where only smaller portions of the texture are required. For example a texture used for an entrance arch with window above, could equally well be used for a wall with a window of the same shape.

5.2. Location-transparent GIS Services

The second part of this chapter shifts the focus to our more recent and innovative work. As outlined in the Introduction (see *chapter 1*), we have identified a number of problems with traditional GIS applications, and have therefore developed RADGIS in an effort to provide an alternative GIS client architecture that overcomes many of these problems (*see section 4.7*).

The runtime-extensible RADGIS architecture enables the development of highly-customisable and scalable GIS clients, that can be tailored by end-users according to their domain-specific requirements and level of expertise. It allows users to make use of both local and remote implementations of GIS services that have been developed and deployed using our DGCML meta-language. This facilitates the use of GIS services that may be invoked with location-transparency on the local machine or on a remote server, depending on where the data that is to be processed is stored. This is an important capability as data and services become more closely integrated, and data providers add value to their underlying datasets by providing value-added services, in order to maintain their competitive advantage.

5.2.1. Standardised Metadata for Efficient Integration

Metadata is extremely important for providing context. Just as we require information to base our decisions on, the more metadata one provides about the GIS components, the easier it becomes to integrate them efficiently with the client GIS application.

Software is generally shipped with documentation on how to use it, together with an online help system. Individual software components (both local and remote implementations) should be created in the same manner, i.e. each component should have associated documentation describing how it should be used, including expected input, output, boundary conditions and possibly what algorithm was used in its implementation, as well as having a comprehensive online help system. However, because these components are to be discovered, and added to the dynamic application at runtime, as opposed to at development-time, there is the additional requirement that this information be stored in a standardized manner, to facilitate automated retrieval.

Two methods for providing information about the GUI and on-line help for each tool, implemented as a CORBA object or EJB, were initially considered. The first was to create a standard set of methods that must be implemented by each CORBA object or EJB, e.g. getHelp and getGUI. These methods could in turn query a database and return the relevant XML encoded metadata. The second method, which we decided to adopt, does not require any changes to existing CORBA objects or EJBs. Instead, it provides the metadata about the intended use of the CORBA objects and EJBs in a readily customisable XML document, using the Distributed Component GIS Markup Language.

5.2.2. The Distributed GIS Component Markup Language (DGCML)

The DGCML meta-language was developed to provide deployment, GUI and help system meta-information about locally and remotely implemented GIS services that could be integrated by the RADGIS client at runtime. The factors that influenced the design of DGCML were covered in *chapter 3*. This section will now provide implementation-specific details of the DGCML meta-language.

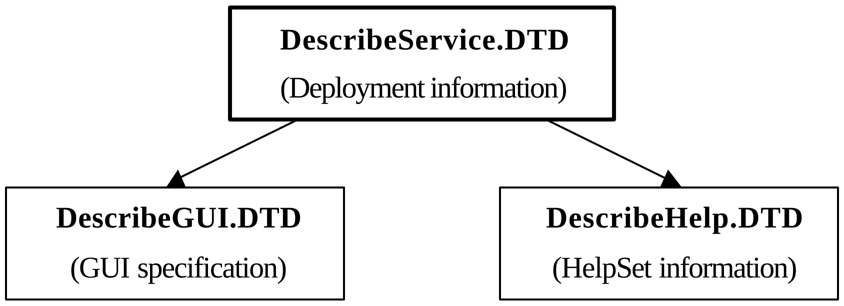


Figure 5.5. The DGCML DTD hierarchy

Figure 5.5 illustrates how the DGCML language has been defined in three separate DTDs that specify the format of the deployment, GUI, and help information stored in a DGCML descriptor file. The describeService.DTD file, shown in *table 5.1*, ensures that a GIS service descriptor based on DGCML contains the following information:

?? The name, vendor and version of the GIS service, as well as an optional icon that can be used to identify the servic e;

?? Links to a description of the service, and the license agreement;

?? The codebase which indicates what type of tool it is, i.e. whether the tool resides locally or is a remote implementation, and its location;

?? A complete description of the GUI required by the tool to operate correctly; and

?? The links to the HelpSet files.

```
<!ENTITY % helpSystem SYSTEM "file://localhost/C:/describeHelp.dtd">
%helpSystem;

<!ENTITY % guiDisplay SYSTEM "file://localhost/C:/describeGUI.dtd">
%guiDisplay;

<!ELEMENT GISService (CodeBase*, GUI, Help)>
  <!-- Menu is optional in case the tool is a stand-alone application -->
  <!ATTLIST GISService name CDATA #REQUIRED vendor CDATA #REQUIRED
                              version CDATA #REQUIRED>
  <!-- href is an optional tag that indicates where to find a web page describing the
          (distributed) tool -->

<!ELEMENT CodeBase (License?)>
  <!ATTLIST CodeBase url CDATA #REQUIRED icon CDATA #IMPLIED>
  <!-- if the icon field is empty, the name of the tool (as bound in NS) is used -->

<!ELEMENT License EMPTY>
  <!ATTLIST License href CDATA #REQUIRED>
```

Table 5.1. DTD for GIS Service (describeService)

5.2.2.1. Deployment metadata

The XML descriptor file for a particular GIS service contains the unique name of the GIS service, the vendor’s name and the version of the service. It also includes an optional link to a description of the GIS service. The description of what the GIS service does, generally also marked-up using XML, serves as online documentation.

The XML encoded description can be processed by an XML-to-HTML style sheet, using the XML Style-sheet Language (XSL) [Adler 2000], to create an HTML document for presentation in a Web browser. Alternatively, it is also feasible that the XML description could be used by a Registry service when differentiating between multiple tools that fulfil the same function.

The DGCML descriptor allows for the specification of alternate codebases, for the provision of more than one remote runtime instance of a particular vendor's implementation of a GIS tool. This provides fault tolerance, and the ability to switch remote service providers based on service levels or cost, in the event that component-based or per-usage charging models are implemented at a later stage (see *section 6.6*).

The specification of more than one codebase for an implementation of a particular tool also provides the ability to choose a server "close" to the data source. This is very useful if, for example, a geospatial data vendor also provides remote access to instances of particular GIS services. (We are assuming that processing the data on the remote site is more efficient than first downloading the whole dataset and then performing the operation locally, or that the user does not have a local implementation of that service.)

5.2.2.2. GUI specification

The DGCML GUI specification allows the client GIS application to build the GUI required by the GIS Service at runtime, using Java's Reflection API. This means that changes to the GUI may be made by simply editing the DGCML GUI description. There is no compile-cycle required, and the changes to the GUI are reflected the next time the GUI is generated, i.e. changes to the GUI do not require the user to close down and restart the entire client GIS application.

The describeGUI DTD, which describes how a GUI for a GIS service can be specified, is given in *table 5.2*.

```
<!--  All methodcall names starting with the string "remote:" are reserved for internal use by
      DGCML -->

<!ELEMENT GUI (Component | MethodCall)*>
  <!-- If there are no components, there is no visible GUI and the tool is simply "executed"
        using method-calls -->


<!ELEMENT Component (Component | Property | MethodCall | Event)*>
  <!-- MethodCall is required for invoking methods on Components that are not
        getting/setting properties of beans, such as pack() on a container -->
  <!ATTLIST Component name ID #REQUIRED type CDATA #REQUIRED
                      position CDATA #IMPLIED>

<!ELEMENT Property (MethodCall*)>
  <!ATTLIST Property comp NMTOKEN #IMPLIED name NMTOKEN #REQUIRED
                     value CDATA #REQUIRED >

<!ELEMENT Event (MethodCall | Property)*>
  <!ATTLIST Event type NMTOKEN #REQUIRED filter NMTOKEN #IMPLIED>

<!ELEMENT MethodCall (Param*)>
  <!ATTLIST MethodCall name CDATA #REQUIRED ReturnType CDATA #REQUIRED
                      ReturnValueDest CDATA #REQUIRED>

<!ELEMENT Param (Property?)>
  <!ATTLIST Param Source CDATA #REQUIRED DataType CDATA #REQUIRED
                  CallType CDATA "IN">
```

Table 5.2. DTD for GUI specification (describeGUI)

The describeGUI DTD specifies the following:

?? Zero or more components or method calls, where the absence of any components infers that the GIS Service does not require its own GUI, i.e. it is probably a batch process.

?? Unique naming of all components such that they may be referenced as sources and/or destinations of arguments/return types for method calls.

?? The getting/setting of JavaBean properties.

?? The events generated by the relevant components.

?? The method calls that should be invoked.

?? The parameter types with which a method call should be invoked, including a callType flag to signal whether the argument in the object's IDL file was defined as an "IN", "OUT" or "INOUT" parameter.

The focus of DGCML is to provide a highly customisable front-end for users that can support tight integration with the client-application. Therefore, it has been kept relatively simple, and has not attempted to become a Java-like XML programming language. This means that certain complex operations may not be easily achieved using DGCML. However, these complex operations would normally encapsulate some programming concept that would be best implemented as a Java class. That Java class could then be referenced in the DGCML descriptor to provide the desired functionality simply and efficiently.

Thus there is a trade-off between the ability to easily customise the functionality of the GUI by editing the DGCML GUI specification, and reducing the complexity and amount of Reflection that is required to build the GUI, by writing certain complex operations as "helper" Java classes.

Complex GUI's may therefore be developed either through the use of Java's Swing components, or custom-built GUI/helper classes. If one wanted to add a custom GUI component not (easily) programmable via the DGCML meta-language, it is possible for that functionality to be created in a standard Java class, and for an object of that type to be instantiated and added to, or used by, the GUI.

5.2.2.3. Help Metadata

Some GIS packages are so large that their documentation is seldom up to date with the new features, and often they have trouble just keeping up with their standard features. For a large GIS package that tries to provide as many tools as possible, the volume of documentation becomes almost unmanageable to maintain, and this in turn makes it difficult for users to find relevant information within the help system.

There is a greater likelihood, however, that if specialists wrote individual services, that the documentation (help system or user manual) would be up to date, and could provide more detailed use-cases, explanations of the algorithms used for certain processes, such as the interpolation methods used for the creation of digital elevation/terrain models, explanations of boundary conditions, or possible problem data sets.

We are using JavaHelp 1.1 [Sun Microsystems 2001c] in our RADGIS client to provide access to remote help data for our GIS services. JavaHelp uses XML documents to specify the structure of each help system, and HTML formatted text for the presentation of the help. It also defines simple mechanisms for the merging of help data from different components that may be stored at different locations, as well as the indexing and searching of these help files. For a comprehensive description of how JavaHelp provides location-transparent access to help data, and the ability to merge help data from multiple components, refer to the JavaHelp User's Guide [Sun Microsystems 2001c].

```
<!ELEMENT Help (HelpSet*)>
  <!ELEMENT HelpSet EMPTY>
  <!ATTLIST HelpSet href CDATA #REQUIRED>
```

Table 5.3. DTD for Help specification (describeHelp)

The describeHelp DTD allows the user to specify one or more HelpSets for the GIS service, i.e. the help system for a particular GIS service can be created from a number of smaller help topics if desired. These HelpSets are merged (in a hierarchical format) with the RADGIS client's help system at runtime when the tool is invoked. Each HelpSet element simply contains a reference to the JavaHelp HelpSet file, which in turn contains references to the JavaHelp map file, and the files necessary for providing the table-of-contents, index, and search views.

Figure 5.6 shows how the HelpSets for two GIS tools (GIS Tool A and GIS Tool B) have been merged with the main help system of the client application. It is possible for the help systems of these tools to be downloaded to reside locally, or to be accessed at runtime across the Internet. Therefore use of JavaHelp, which facilitates the implementation of a distributed help system, provides a high level of location transparency to the end-user when accessing help information.

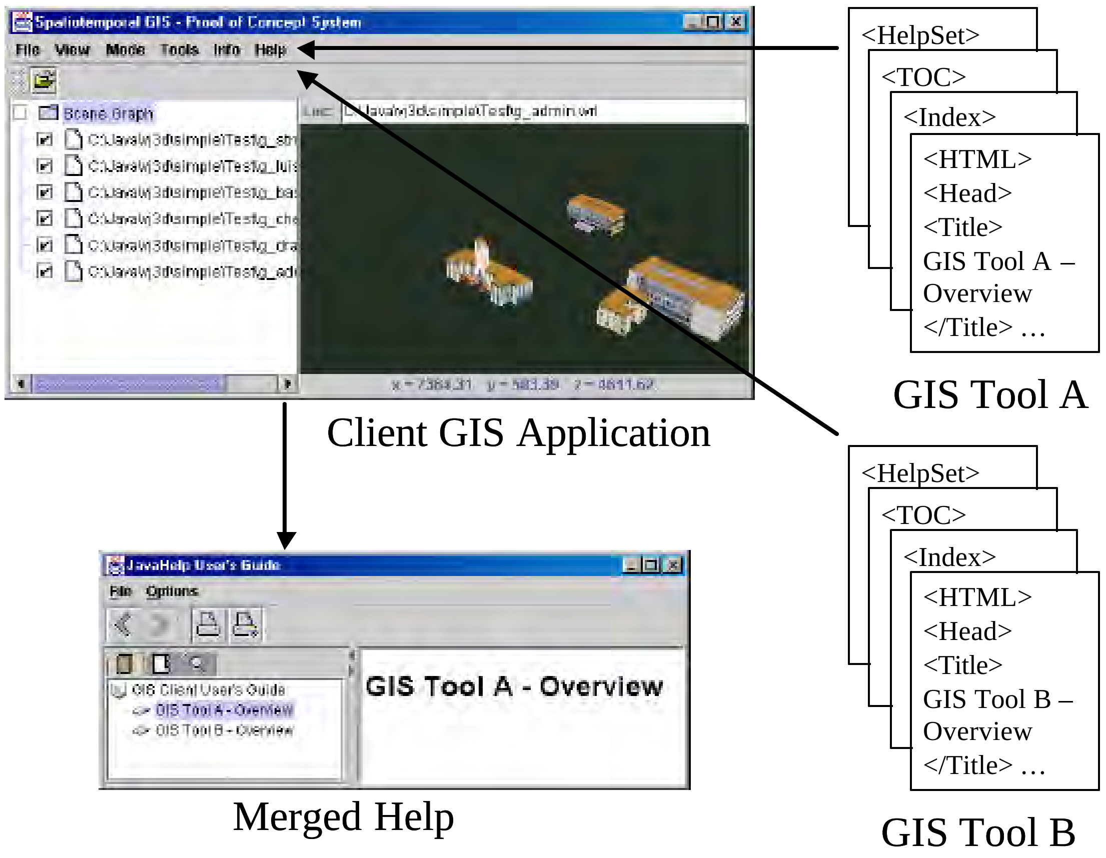
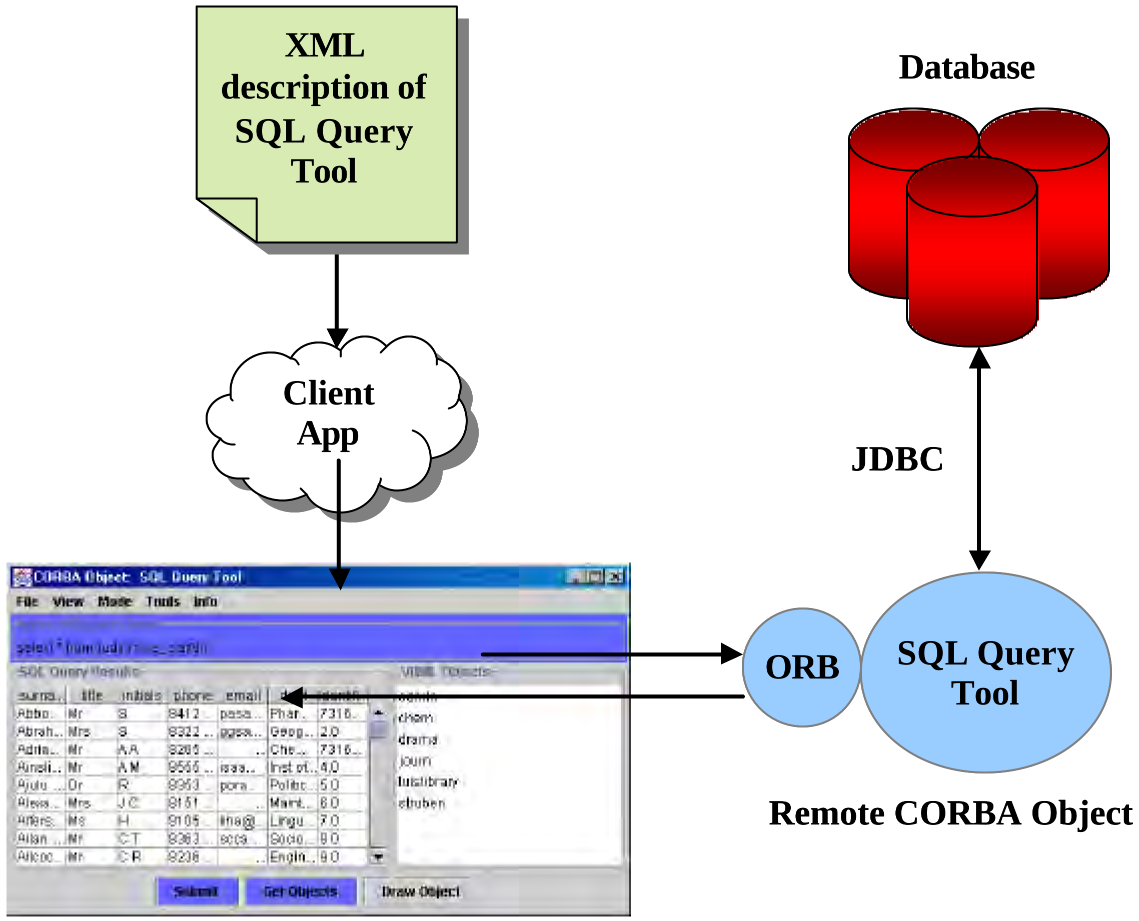


Figure 5.6. Use of JavaHelp to display help data from different GIS Services

5.2.3. Adding and Invoking a Service – A Use-case Example

In order to illustrate how the functionality of the RADGIS client can be extended at runtime through the addition of a new service, developed and deployed using DGCML, we will now briefly describe a simple use-case scenario for a remote tool that has been implemented using a CORBA object.

Below, *figure 5.7* illustrates how a simple SQL query tool, implemented remotely as a CORBA object, could be added at runtime and invoked by the client application.



surna..	title	initials	phone	email	dept	[illegible]
Abbo..	Mr	S	8412 ..	pasa..	Phar..	7316..
Abrah..	Mrs	S	8322 ..	ggsa..	Geog..	2.0
Adria..	Mr	A A	8285 ..	..	Che...	7316..
Ansli..	Mr	A M	9566 ..	isaa..	Inst of..	4.0
Ajulu ..	Dr	R	8963 ..	pora..	Politic..	5.0
Alexa..	Mrs	J C	8151 ..	..	Mark..	6.0
Allers..	Ms	H	9105 ..	lina@..	Lingu..	7.0
Allan ..	Mr	C T	8363 ..	scca..	Socio..	8.0
Allcoc..	Mr	C R	8208 ..	..	Engin..	9.0

admin
chem
drama
journ
luslibrary
struben

Figure 5.7. CORBA SQL query tool example

First the user searches for the tool that is required for the task at hand, possibly by browsing the Web for links to DGCML descriptors or using specialised lookup services such Registry or Directory services (see *section 4.6*).

Once the user has found the desired tool, the DGCML descriptor for that tool is downloaded, together with any supporting Java "Helper" classes, and then the icon and tool name are added to the GIS client's menu and/or toolbar. This completes the "installation" of the new service.

When the user chooses to invoke the service by selecting it from the menu or toolbar, the DGCML descriptor file is parsed by the client application. The client application then looks up the object reference to the CORBA object, and assuming that there was no compile-time knowledge of the tool, would use CORBA's Dynamic Invocation Interface (DII) to invoke methods on the object.

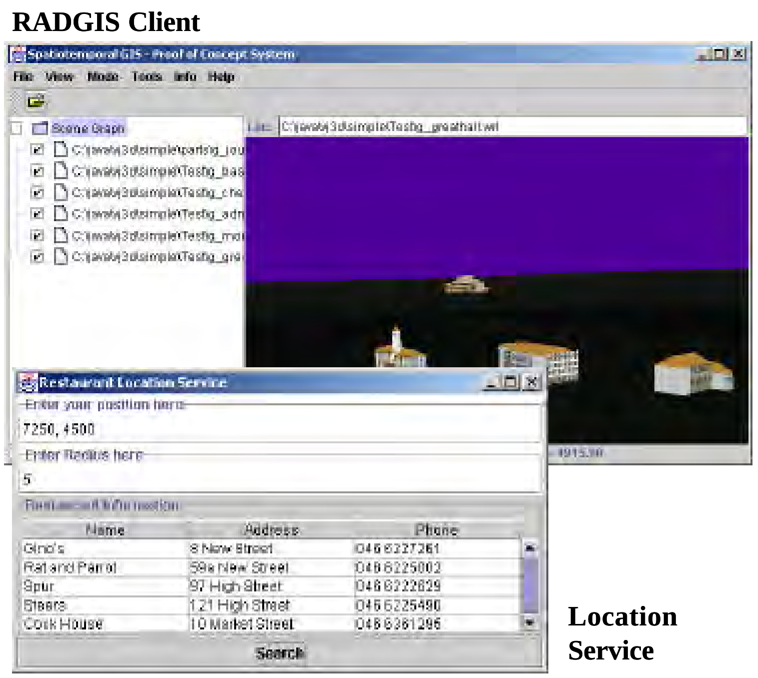
These method invocations would generally be the result of events generated by the user of the client application, when interacting with the GUI associated with the tool. This GUI would have been created by the client application in response to the XML description associated with the CORBA object, e.g. the SQL Query tool in *figure 5.7*.

5.2.4. Creation of a Location Service

In general, location services are specific applications of one or more generic GIS tools that simplify the interface and allow traditionally non-GIS users to make use of GIS-type functionality transparently.

For example, consider the interface required for implementing a simple yellow-pages location service for finding all restaurants within a particular distance of a user-specified location. This location service is essentially a "complex" spatial query that would probably not be correctly specified by a non-technical user. However, due to the nature of the location service to be implemented, the number of unknowns (such as the number of parameters, the names of these parameters, and the name and location of the database table holding the restaurants' location information) are reduced, allowing much of the query to be pre-generated by the developer of the service.

In addition, by creating a simple interface, such as the one implemented using DGCML shown in *figure 5.8* (the DGCML listing for this example has been included in *appendix E*), it is very easy for the user to make use of such a service, i.e. only the user's location and radius in which to search for restaurants are required. It is possible to simplify the user interface even more if the user accesses the location service from a cellular phone, which can be used to automatically determine the user's location using triangulation.


RADGIS Client
Spatiotemporal GIS - Proof of Concept System
File View Mode Tools Info Help
Scene Graph
Restaurant Location Service
Search
Location Service

Enter your position here: 7250, 4500

Enter Radius here: 5

Restaurant Information

Name	Address	Phone
Gino's	8 New Street	046 6227261
Rat and Parrot	59a New Street	046 6225002
Spur	97 High Street	046 6222629
Steers	121 High Street	046 6225490
Cock House	10 Market Street	046 6361295

Figure 5.8. Example of a location service for finding restaurants

The above example shows the tremendous flexibility of the RADGIS architecture. RADGIS is, in effect, a generic client architecture that is able to parse DGCML descriptors at runtime and, using Reflection, generate the GUI necessary to execute GIS services, location services or any other type of service described by a valid DGCML descriptor. Thus our approach allows one to create a client-side framework in which one could combine location services to create more sophisticated tools, as well as allowing further processing of the results obtained from invoking a particular location service.

5.3. Invocation of methods at runtime

Our DGCML goes further than simply wiring together components to create an application, but looks at the integration of tools for which no previous knowledge of other components is known. At the same time, a requirement of our RADGIS system is that it must be simple for the user to add and remove tools as well as modifying their installation and/or GUI for tight integration with the client application.

In our description of the DGCML grammar so far, we have not mentioned anything about how the services are implemented. This is because the DGCML is independent of whether the services are implemented as CORBA Objects, EJBs or simply in local class files (stored in JARs).

Methods of both local and remote objects may be invoked when executing a GIS service that has been developed using the DGCML. However, because these services have been developed separately to the RADGIS application, the RADGIS application has no compile-time knowledge of these objects. It is therefore necessary to make use of special mechanisms to accommodate this dynamic behaviour.

5.3.1. Invocation on local objects using Reflection

Java classes that implement GIS services, which have been downloaded to the client's machine to run locally, require a runtime mechanism to instantiate objects, set properties and invoke methods. Fortunately Java provides just such a means for the "introspection" of Java classes, namely Reflection.

Reflection is a runtime capability that facilitates late binding, and is an essential part of JavaBeans technology, although its uses stretch far beyond JavaBeans. Its power lies in providing an abstraction that frees software from having static references to target classes and objects when it is compiled [Portwood 2000]. Thus, designing with Reflection provides flexibility, extensibility and pluggability that are essential for the type of client architecture we have developed.

The java.lang.reflect package provides the ability to query a Java class about its properties, and to operate on methods and fields by name for a given object instance, within the basic security framework, e.g. based on access modifiers [Tremblett 1998]. For more information regarding Java's Reflection mechanism, refer to [Portwood 2000].

According to Portwood [2000], there are a number of myths surrounding the use of Reflection, including that Reflection is too complex for use in general-purpose applications and that Reflection reduces performance of applications. He maintains that, when properly applied, Reflection leads to improved performance and simplified maintenance, with greater reusability and extensibility of software.

Apart from the method invocations on the GIS tools, Reflection is used for building the GUI from Swing components. Swing components are JavaBeans, and as such allow one to use the Bean Introspector to interrogate a Swing component (using Reflection) to reveal important aspects of its behaviour, such as the types of events it will respond to, and the types of events it may generate [Spruit 1997].

5.3.2. Invocation on Remote Objects

Our RADGIS application makes use of both static (i.e. using compile-time knowledge of stubs) and dynamic (i.e. runtime discovery) method invocations on remote objects.

This means that, at runtime, it is possible to look up references to:

?? distributed objects that were known about at compile-time, and were "built-in" by the programmer. Here the only "unknown" at compile time may have been where the distributed objects would reside at runtime.

?? distributed objects that were not known about at compile-time.

The former is relatively straightforward because one is able to make the assumption that the programmer has detailed knowledge of how the remote object should be invoked, what parameters to pass, and what to do with the result. The latter, however, is far more complicated as one cannot assume that the user has any knowledge of how to invoke the tool that has been discovered at runtime, nor how to incorporate it into the current application. The level of expertise required by a user (or by the framework acting on behalf of the user) that wishes to discover new tools at runtime is therefore much greater than that of a user that only makes use of static invocations of "built-in" tools.

There are two possible models for accessing remote services:

?? The first assumes that only the discovery of tools is dynamic, and thus once a tool has been selected, the necessary stubs are downloaded and installed on the client machine.

?? The second requires the use of the Dynamic Invocation Interface (and the Interface Repository) to implement the dynamic invocation of method-calls on the remote CORBA Object(s), or the use of Java's Reflection mechanism for EJBs. It is unfortunately not possible to simply use Java's Reflection mechanism to invoke methods dynamically on both CORBA Objects and EJBs, as the CORBA Objects may have been implemented using a language other than Java (which may not support Reflection), and therefore the CORBA (proxy) Objects do not hold the necessary information for performing Reflection on the CORBA Objects' implementations. Both of these approaches have therefore been implemented in the RADGIS client to provide transparent access to CORBA objects and EJBs for which no compile-time knowledge exists.

5.3.2.1. CORBA Objects

CORBA's Dynamic Invocation Interface (DII) allows a client to choose any target CORBA Object at run time and then dynamically invoke its methods.

Dynamic invocation using the Interface Repository is invariably slower than using static stubs. However, much of the performance loss associated with DII in general, is attributed to looking up the interface name, getting the operation identifier/parameters, and creating the request (Duman, 1999). Therefore, using the information about the method-call stored in the XML descriptor, one is able to minimise the performance loss associated with using the dynamic invocation.

5.3.2.2. Enterprise JavaBeans (EJB)

It is possible to create the necessary proxy objects for the home and remote interfaces required for invoking methods on a particular EJB using Java's Reflection API (as used in *section 5.3.1* for invoking methods on local objects). This is extremely useful as it simplifies the invocation process and allows a standard mechanism for the runtime invocation of methods on both local and remote objects.

5.4. Summary

In this chapter we have presented the implementation details and some of the initial design considerations that were taken into account during the development of our RADGIS application and the DGCML grammar.

Specifically we detailed the use of VRML models for the creation of the data set, the problems associated with the use of VRML for cartographic representation and the work currently being undertaken to solve these problems together with the emerging X3D standard.

Once we had outlined how the RADGIS client visualised the 3D data, we gave a break-down of the DGCML descriptor file and provided a simple use-case scenario. We also provided an example of how a location service could be developed using DGCML. The ability to invoke methods on local and remote objects was then discussed to complete the overview of technologies used in the development of the core functionality of the RADGIS application.

With this knowledge it is now possible to look at some of the implications that the RADGIS architecture has on the development of GIS services for developers, as well as the implications of using and customising such an application for the end-user. This will provide us with an opportunity to discuss the benefits that are derived from using the RADGIS architecture, and to look at some of the issues would that need to be addressed if RADGIS were to be released commercially.

Chapter 6
Discussion

"Reality is merely an illusion, albeit a very persistent one."
Albert Einstein (1879-1955)

Thus far we have described the problems with current GIS architectures, which have highlighted the need for an extensible client architecture that facilitates the addition, and close integration, of interoperable GIS services at runtime. We have also outlined the implementation of our proof-of-concept system, called RADGIS, which aims to fulfil these requirements. In this chapter we will now present some of the implications that arise from the use of the RADGIS architecture, such as the design considerations for object developers and component integrators. In particular, we will highlight the fact that very little, if any, modification to existing objects is required, and that component integrators are able to assemble GIS and location services with tremendous ease and flexibility.

We will also discuss what impact the use of the RADGIS architecture has on the usability of client-side GIS applications, i.e. the level of expertise required by the user to successfully make use of the RADGIS client. This allows us to present the qualitative benefits of using the RADGIS architecture to access GIS tools and services distributed across the Web.

During the course of our research, and the implementation of our RADGIS architecture, a number of issues arose from the use of distributed components and services that were not part of the original research mandate. These include the necessity to provide a directory service for registering and locating GIS services, as well as the adoption of new charging models for distributed components and services, the possibility of employing parallel processing, and the ability to modify the user interface at runtime based on what tools are currently being used (adaptive or intelligent user interfaces).

While we were unable to address these supplementary issues in the limited time and scope of this investigation, we have devoted the latter part of this chapter to these issues in order to highlight further work that would have to be undertaken if one were to implement a full version of the RADGIS architecture for commercial release.

6.1. Design considerations for the object developers

Due to the design of DGCML, which allows component integrators to wire together both GUI objects, and local and remote objects that provide GIS functionality, there is no need to modify existing GIS tools that have been developed as components. The flexibility of the DGCML language, together with the ability to make use of "helper" Java classes ensures that it is possible to adapt an existing GIS tool for use in the RADGIS architecture. The amount of "adapting" that must be performed by the component integrator to successfully make use of that component can be considered an indication of its level of re-usability.

This is a large benefit of our design, since it does not require that the objects that implement the desired GIS functionality be rewritten to adhere to a particular standard or format. Instead, it is possible to adapt existing components. However, there are definite benefits to be derived from creating GIS services based on objects that have been implemented using standards, e.g. the Open GIS Consortium's Implementation specifications, including simplifying compatibility and substitutability decisions (see *section 2.1.4*), and reducing the need to create complex component adapters.

Therefore, while there are no additional requirements imposed by the RADGIS architecture on object developers, or the need to rewrite existing objects, it would be beneficial for object developers to create GIS tools that conform to standards.

6.2. Design considerations for the component integrators

It is the component integrator who will most likely write the DGCML descriptors that provide customised GIS services, which can be integrated by the RADGIS client at runtime. The goals of component-based software development are to maximise code reuse and simplify application development. However, many unforeseen circumstances may arise when trying to make use of such a generalised approach to application development, particularly with respect to integrating components that were developed by different vendors, at different times and possibly across application domains.

Thus, from time to time, it may be necessary for a component integrator to also write converter or "helper" objects to facilitate particular operations that fall outside the scope for which the component was originally intended. These helper classes also allow component integrators to simplify the DGCML code necessary for specifying complex GUIs and component interactions, as well as for "hiding" certain operations that should not be modifiable by the end-user. The customisability and the flexibility provided by the DGCML meta-language, together with the ability to create and utilise "helper" Java classes, therefore ensures that a component integrator is able to develop easily-customisable, yet sophisticated, GIS services based on diverse GIS components.

However, without standardisation amongst GIS object developers and component integrators, on issues such as naming conventions and documentation, the power to replace implementations and to facilitate tight integration of services with the client application will be severely hampered. Therefore the greatest benefits will be derived from our approach, and CBSD in general, when used in combination with agreed upon standards.

6.3. Implications for the client

The application of our approach to dynamic runtime systems' development using distributed objects, to the field of GIS, opens up the scope of GIS applications for traditionally non-GIS communities that require only a subset of the GIS functionality available. Thus users would no longer have to download and install large GIS packages if they were only ever going to use a fraction of the GIS functionality that is typically bundled with current GIS applications. Instead, using RADGIS, they would have the ability to add GIS services to the client framework as, and when, required.

Rudimentary GIS applications can be created by relatively inexperienced users, simply by selecting the tools required, and electing to add them to the client application. This course of action makes use of the default DGCML descriptor file, developed by a component integrator, which defines the mechanics of how the tool is to be located, and integrated into the dynamic application at runtime. No further customisation or user intervention is required in order to make use of these tools. The RADGIS client uses the self-describing DGCML descriptor to display the default GUI for invoking the tool, as well as integrating the help associated with the tool, into the client application's help system.

However, the real power of this approach lies with the expert user who can modify a DGCML descriptor, so as to allow tighter coupling and integration with other local and distributed tools, as well as the core client application. This includes the development of new services that perform multi-stage processing by chaining tools together, or executing batch operations.

Thus, if the tools are used with their default DGCML descriptors, no further configuration is necessary and the end user is not required to perform any complex integration operations. However, expert users are able to benefit from the large degree of customisation available, and are therefore no longer restricted to use a GIS designed for "most" users, but instead can customise their client GIS to suit their individual requirements.

6.4. Combining GIS tools and data to create location services

Our research has primarily focussed on developing an architecture that allows the user to create their own customised, scaled-down GIS application, which contains only the tools required for his/her particular tasks. However, we have also demonstrated how our approach would allow one to create a client-side framework in which one could combine location services to create more sophisticated tools, as well as allowing further processing of the results obtained from location services (see *section 5.2.4*). The RADGIS architecture could, therefore, just as easily facilitate the development of small, specialised applications that integrate one or more location services.

This is an important capability of the RADGIS architecture considering the significant industry interest and financial backing that is moving the Location Services industry forward as new markets for location services are rapidly being uncovered. Location Services' strong reliance on GIS functionality means that many of the OpenGIS® standards developed by the OCG, particularly GML, will also be used in the development of future location services.

The relatively recent development of location-based services, which share much in common with distributed GIS services, has also spurred renewed interest in providing Web-based GIS functionality. Location services will therefore foster the development of open implementations of GIS tools that may be accessed remotely, as well as the exposure of valuable intrinsic location-based information. This is extremely important because it is only as more data and tools are made available online, that the relating and organising of this location-based information will allow hidden meanings and relationships to be revealed, and the true potential of GIS and location services to be unlocked.

6.5. Runtime discovery of remote implementations of GIS services

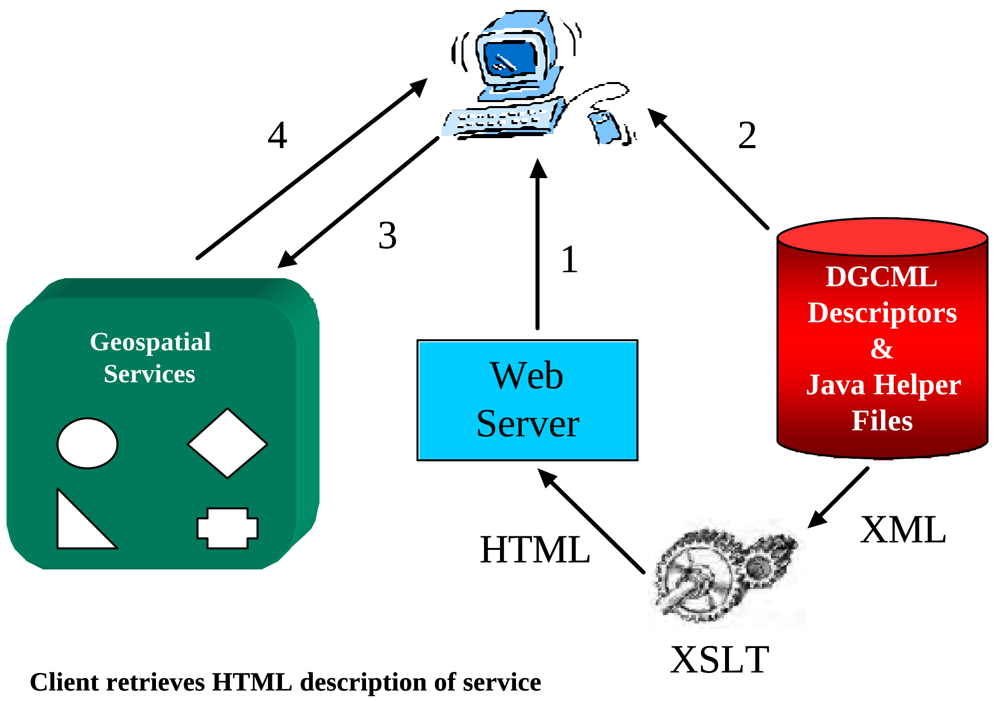
The envisaged future of distributed geospatial data and services will not become a reality without the ability to efficiently expose these data and services. In the same manner that web search engines have enabled us to search huge quantities of data, so the ability to search for geospatial data and services will require providers of these data and services to register them with software that provides a searchable repository of features and metadata.

Due to the focus of the research we have not implemented a catalog or directory service (see *section 4.6*) to allow the registration and discovery of GIS services implemented using DGCML. Instead, for demonstration purposes, we have simply generated Web pages "advertising" the services. These Web pages are based on the XML descriptions of the GIS services, and have links which allow the user to download the DGCML descriptor(s) and any supporting Java "Helper" classes (see *figure 6.1*), bundled as a JAR file with an extension .DGJAR.

The user can browse, download and install a new service in two different ways:

?? The client may use a standard Web browser to navigate to, and browse, a Web page which describes a GIS service that may be used by the RADGIS client. The user may download the JAR file using the Web browser to the local machine, and install the GIS service by selecting the .DGJAR descriptor using the "install" option in the RADGIS client at runtime.

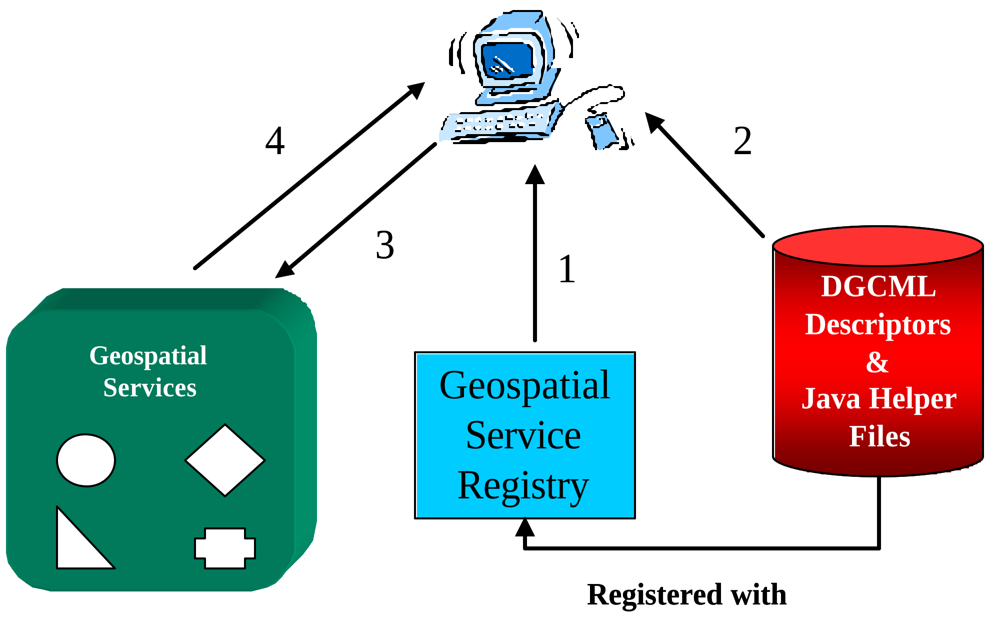
?? Alternatively, the RADGIS client can make use of a generic Web-browser service, implemented using a customised Java Swing JEditorPane component, that has been developed and deployed as a DGCML descriptor. Clicking on a link to a .DGJAR file in this Web-browser service invokes the RADGIS client's installService method, automating the download and install process, i.e. adding the icon and service name to the menu and/or toolbar.



1. **Client retrieves HTML description of service**
2. **Client uses HTML to locate and download DGCML and Java Helper Files**
3. **Client uses DGCML descriptor to invoke method on remote object**
4. **Client obtains result from remote object**

Figure 6.1. Advertising GIS services using Web pages

However, we realise that the simplicity of this approach also has a number of drawbacks, and does not take into account the research being undertaken with respect to directory services that are being designed to simplify object/service registration and discovery. Therefore we refer the reader back to *section 4.6*, in which we discussed some of the prospective technologies that may be used to develop registries that advertise GIS services. *Figure 6.2* shows the role that a registry service would play in the RADGIS architecture to allow the client to discover a particular GIS service.


4
3
1
2
Geospatial Services
Geospatial Service Registry
DGCML Descriptors & Java Helper Files
Registered with

1. **Client searches geospatial service registry for particular service, based on name or properties**
2. **Client uses link to locate and download DGCML and Java Helper Files**
3. **Client uses DGCML descriptor to invoke method on remote object**
4. **Client obtains result from remote object**

Figure 6.2. Advertising GIS services using a searchable registry

6.6. Benefits

The runtime application development paradigm of the RADGIS client architecture, which facilitates the addition of GIS services, developed and deployed using DGCML, has a number of advantages. These include:

?? the creation of small GIS applications with specific functionality, aimed at reducing:
  - o the per-seat licensing fee;
  - o the complexity of the overall application; and
  - o the learning curve for non-GIS users.

?? finer licensing granularity, which facilitates the use of alternative pricing models (see *section 6.7.1*). Thus the user may no longer be required to pay for functionality that they do not require.

?? reducing application bloat, which in turn reduces the in-memory and secondary storage requirements of the software.

?? loading functionality only when required which amortizes the load time through the entire run of the application. Unused functions are never loaded, which reduces the demands made on system resources.

?? the flexibility to support both novice and expert users within the same architecture.

?? the ability to make use of different underlying implementations of a particular GIS service while retaining the use of a single "familiar" user interface to which the user may be accustomed. This is particularly useful for eliminating the "limited transfer of knowledge" when a user changes from using one software-package to another.

?? providing the user with increased flexibility when choosing application components and services, depending on the user's processing requirements and level of expertise.

?? improved robustness due to reuse and constant retesting of components in different environments.

?? the ability to make use of dynamic invocation, which ensures that users always have access to the latest version of a particular object/service. Due to the centralised nature of server-side processing, a vendor can easily retract an old version and rebind a new version of that tool to the old naming context, thereby dynamically upgrading the tool transparently and ensuring that users always work with the most up-to-date versions of objects.

?? the ability for a single distributed object to be used by many different service implementations that are distributed across the Internet. This reuse of "live" software components or services has the potential to be more useful than making use of "static" component code repositories.

?? increased interoperability and customisability of components via a runtime extensible client framework.

?? possible bandwidth savings if the geospatial data to be processed resides "close" to the remote service that is being invoked, e.g. if both the data and service reside on the same Intranet.

Our approach has the added benefit that within the field of GIS, the Open GIS Consortium has started standardising interfaces and specifying IDLs for the spatial and attribute datatypes [Open GIS Consortium, 1998b]. It is therefore possible to create a client that knows how to interact with standardised GIS datatypes and services, increasing the ability to incorporate new tools at runtime with high levels of integration.

Alternatively, if one considers the use of more general-purpose components, it also becomes possible to make use of a single component in more than one application domain. Hence it may be possible to generalise our RADGIS architecture to the point that one no longer requires the use of different applications for different tasks. Instead it would become possible to make use of a set of loosely coupled components that allow tight integration to process information across traditional application boundaries.

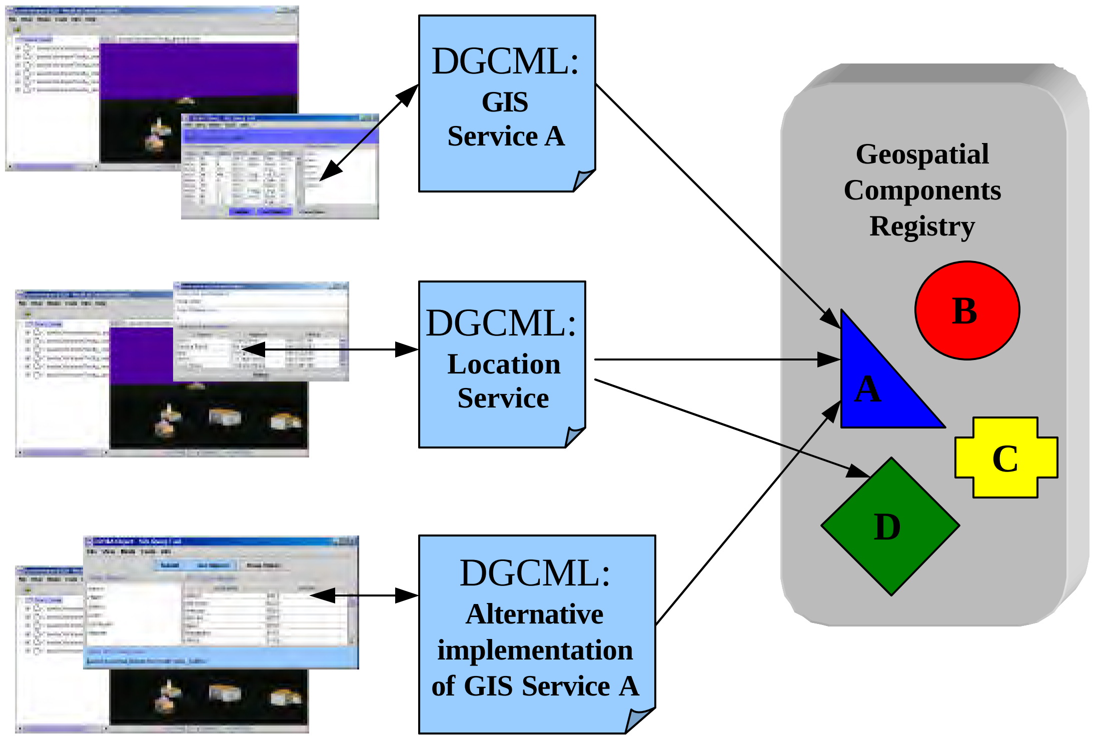

DGCML:
GIS Service A
DGCML:
Location Service
DGCML:
Alternative implementation of GIS Service A
Geospatial Components Registry
A
B
C
D

Figure 6.3. Example showing how more than one DGCML descriptor could make use of a distributed GIS tool

While the implementation of such an architecture may currently be too ambitious, the ability to utilise the same component in multiple applications, using DGCML descriptors that adapt it for use in a particular application, is possible. Each DGCML descriptor could simply reference the same component(s), but provide application-specific GUIs and help (see *figure 6.3*).

6.7. Implications of distributed CBSD for a Commercial implementation of RADGIS

We have covered a number of component-based software development technologies that have shown tremendous potential for application in the domain of GIS. However, there are three areas that warrant further examination with respect to implementing a full commercial version of our RADGIS client architecture, and the effective utilisation of distributed GIS and location-based services. These are:

?? the ability to charge for the use of distributed GIS services;

?? design considerations associated with the ability to invoke distributed components concurrently; and

?? the need to increase the "intelligence" of the user interface to cope with the complexities arising from the addition of new services at runtime.

6.7.1. Charging Mechanisms for distributed objects

The implementation of distributed GIS tools that are invoked remotely means a user would no longer be required to buy, install and maintain large, costly GIS packages locally. Instead it would be possible for a user to purchase individual services that run locally, and to "rent" distributed services on a per usage basis. Therefore, a number of heterogeneous pricing models may be used to determine the cost involved in accessing and processing geospatial data.

The adoption of the RADGIS architecture commercially would require the ability for GIS service providers to charge for the use of their distributed services. While charging models for distributed software tools are not the focus of this thesis, we would like to point out some of the pertinent issues:

?? The use of a distributed tool may be charged on a per-usage fee, and/or a lookup table of registered users may be consulted to determine payment method, i.e. it may be possible to pay a once-off fee for using a service, or pay a per-usage fee, or a combination of an initial fee plus a per-usage fee.

?? Assuming the distributed tool processes a particular dataset, which is not provided by the user, the client may be charged for the use of/access to the dataset independently of the use of the distributed tool. Alternatively, if the dataset is only accessible using that particular tool, the cost of accessing the data may be incorporated into the cost of using the distributed tool.

?? The user may simply be charged for the use of the software, or may also be charged for processing time. It may be useful to charge the client different amounts depending on the size of the data set involved. This also allows quality of service charging mechanisms to be introduced whereby the client may choose how quickly and/or when to process their data, i.e. PC vs. Mainframe, and peak vs. off-peak times.

It would also be essential to define what service is being provided, what cost is involved, and what boundaries are defined for where the service starts and stops. Additionally, the user may also be given the ability to negotiate a service contract for the use of a particular tool that cannot change without the user's knowledge and consent/authorisation. For example, if a software vendor upgrades their software, and then decides to charge more for the new service, the user must be notified of the change, but may, by law, also have the right to retain the use of the old version at the original price.

The ability to negotiate a fixed cost for invoking a particular version of a tool would also reduce the need to negotiate the use of that tool every time it is used, and would allow for charging to be performed transparently. This would be particularly useful for tools that are used frequently, and that have a relatively low cost associated with them.

According to Gabriel and Wagner [2001], neither the ISO nor the OGC define a price model in sufficient detail for electronic commerce (e-commerce) applications. They also believe that while general e-commerce developments like UDDI, Electronic Business XML (ebXML) and RosettaNet[1] are suitable for commercial-off-the-shelf products, they do not provide enough flexibility in their pricing mechanisms to adequately deal with configurable

[1] RosettaNet is an organisation that was set up to define and implement a common set of standards for e-business, supporting business processes between supply chain partners.

products. For a more in-depth discussion on pricing models, please refer to [Gabriel and Wagner 2001].

6.7.2. Potential parallelism issues

Coppit and Sullivan [2000] point out that even though a program that is composed of multiple executable components is inherently concurrent, modern components do not provide much functionality for concurrency control. Therefore it is necessary to be aware of the potential danger of invoking distributed components concurrently.

Within the CORBA and EJB specifications there are different ways of communicating with remote objects, including synchronous (invocation causes the client to block, waiting for a result before continuing), deferred synchronous (invocation returns immediately but application must poll for the result) and asynchronous (message is sent to the local message queue and execution continues without waiting for a result – the result is returned to the local message queue triggering a callback method) requests.

The CORBA 2.0 specification supported only synchronous and deferred synchronous method invocations, while CORBA 3.0 supports synchronous, deferred synchronous and asynchronous messaging. The EJB 2.0 specification introduced message-driven beans, in addition to the existing entity and session beans, and therefore now supports both synchronous and asynchronous communication.

If a client application makes use of multi-threading to invoke distributed objects, or if distributed objects' methods are invoked using deferred synchronous or asynchronous requests, then the implications of executing two or more tools in parallel must be considered.

This parallelism can be used to execute two or more *different* GIS tools in parallel, or to subdivide a single task into many smaller subtasks, and then execute each of these in parallel using two or more versions of the *same* tool.

An example of the use of parallel processing to increase efficiency is shown in *figure 6.4*. In *Process A*, all components are executed in sequential order. *Process B* illustrates a semantically equivalent set of operations that will achieve the same end-result. However, in

Process B, it is possible for the GIS tools, which simply convert an ARC file to a polygon list and a DXF file to polygon list, to be executed in parallel. Therefore, in *Process B*, the time taken before the results can be combined is the greater of the times taken by the two GIS tools to complete execution, i.e. the greater of Time D or Time E.

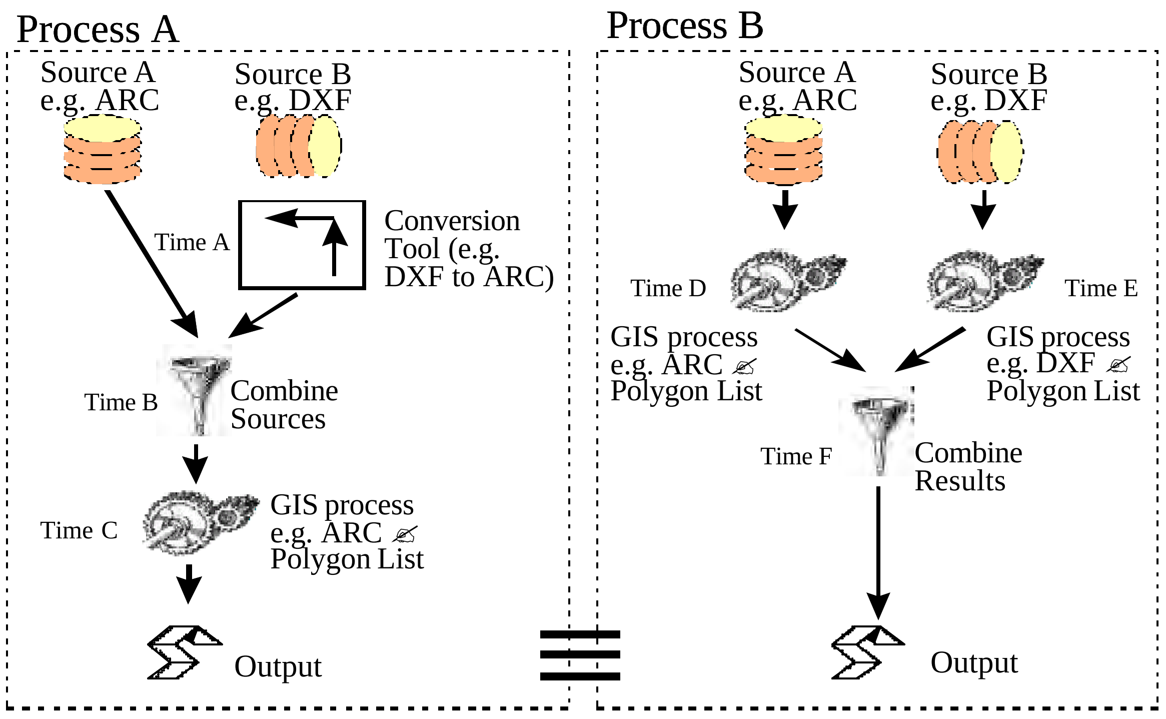


Figure 6.4. Contrived Example of Process Equivalence

If the time taken to convert a DXF file to a polygon list is the same as converting an equivalent ARC file to a polygon list, and the ARC and DXF files in Process B have an equal number of polygons, then optimistically, it is possible for Process B to be twice as efficient as Process A. In addition, due to the restructuring of *Process A*, the time taken initially to convert the data sources into a common data format (Time A) is removed, as this operation is no longer required.

If one further assumes that the output of the GIS tool is at least an order of magnitude smaller than the input, then in general, the time taken to combine the results in *Process B* should be at least an order of magnitude smaller than the time taken to combine the inputs in *Process A*. Thus, the ability to restructure the way in which a process is performed, together with the ability to make use of parallel processing, means that *Process B* has the potential to be dramatically more efficient than *Process A*.

However, if there are dependencies between different tools, i.e. they are required to run in sequential order, or modify the same data, it is imperative that a transparent mechanism to prevent the client from being able to fire off these processes in parallel is implemented to ensure data integrity.

6.7.3. Dynamic Interface Development

Our research focuses on the creation of a runtime extensible client architecture that enables the addition of distributed services. Dynamic component integration allows components to interact in ways that might not have been predicted by the original designers. However, this adds additional complexity to user interfaces that are currently too inflexible, are not able to change according to the user's needs, and do not interoperate. Therefore, in our research, we also briefly explored the possibility of providing an adaptive user interface.

The implementation of an adaptive user interface would simplify the potentially complex user interface that might result from the addition of numerous GIS services during the lifetime of the client application. In addition, we have implemented a basic mechanism that provides the ability to send data stored in a GUI component of one GIS service, to a GUI component residing in another GIS service without explicitly coding the relationship between them. This is done at runtime, based on the GIS services that are currently being used.

One of the goals of the RADGIS architecture was simplicity of operation. Therefore, while we have not fully implemented or explored the implications of developing an adaptive interface for the RADGIS client, further research in the fields of Adaptive and Intelligent User Interfaces (see *appendix D*) is required to ensure that the benefits of the RADGIS architecture can be fully realised without adding undue complexity to the user interface.

6.8. Summary

In this chapter we have brought together the qualitative results of the research that has been presented in this dissertation. It has demonstrated the benefits of using the RADGIS architecture and discussed the design considerations and implications for object developers, component integrators and the end-user.

We have also highlighted some of the technical issues that have arisen from the research that has been undertaken, which require further examination. The most important of these is the registration of services and distributed objects with catalog and registry services so that they can be easily discovered and integrated into applications.

While directory and catalogue services are currently mainly concerned with the passive self-registration of objects, it is not unlikely that in the future, these services will be able to actively seek out services in a manner similar to that of current Web search engines. This will facilitate the dynamic registration of services in up-to-date directories and catalogues.

A number of issues arising from use of distributed CBSD, and which require further research, were also raised, including:

?? the ability to make use of an innovative charging mechanisms;

?? the ability to perform parallel processing, and deal with some of the associated risks; and

?? the possibility of creating an intelligent adaptive user interface to facilitate the automatic runtime customisation of the user interface.

Chapter 7
Concluding Remarks

Vanessa: That's you in a nutshell.
Austin Powers: No, this is me in a nutshell: “Help! I'm in a nutshell! How did I get into this bloody great big nutshell? What kind of shell has a nut like this?”

Austin Powers International Man of Mystery.

This thesis was motivated by the current change in paradigm of application development in general, and particularly within the field of GIS, towards a network centric approach that facilitates the integration of distributed resources. The focus of our approach, therefore, was the provision of a *runtime-extensible* and *customisable* GIS client architecture that provides the user with location-transparent access to independently-provided, yet interoperable, distributed data and services.

During the course of this thesis, we have described some of the major problems with current GIS architectures, and we have highlighted the move towards a highly component-based, distributed software architecture. We have also provided an insight into the research initiatives that are currently underway to provide open standards for the implementation of GIS services and data formats. These standards are being developed to facilitate the implementation of vendor-independent *interoperable* GIS services, and will ensure the future success of the distributed geoprocessing model.

While GIS has contributed a tremendous amount to our understanding of spatial relationships, its greatest contribution yet may be as a number of loosely connected distributed, but interoperable, services and data sources. As users become more adept at working with location-based data, so they will chain services together to perform more complex tasks. In turn developers will be able to gain better insight into the requirements of users in particular domains, enabling them to focus their efforts on providing composite services and customised front-ends, while reusing the underlying GIS functionality.

This final chapter brings together the work that has been presented in this dissertation on runtime-extensible, distributed GIS applications, by offering a critical assessment of our RADGIS client architecture, including its limitations. We then conclude by highlighting our contribution towards the study of distributed GIS applications.

7.1. Assessment of the RADGIS client architecture

There are currently two major GIS architectures: the traditional GIS client and the Web-browser front-end to a Web Map/Feature Server for visualising GIS data. The latter is a very constrained architecture that does not provide much processing functionality, but was developed as a mechanism for allowing widespread visualisation of geospatial data across the Web. Web-browser based GIS applications are limited by the GUI functionality of Web browsers in general, and by the fact that basic GIS visualisation operations, including simple Pan and Zoom operations, require processing of data on a server machine, which incurs performance penalties because large amounts of data need to be transferred across the network. Therefore, there is a growing trend away from Web-browser based GIS towards small, customised clients with rich GUI functionality that are able to access distributed services.

The two major problems with traditional GIS applications are that they suffer from application bloat, and provide very limited interoperability with software from other vendors. These factors have a negative impact on the overall usability of GIS applications, which are generally considered difficult to operate and limit the transfer of knowledge when moving from one GIS application to another, because knowledge gained while using a GIS application from one vendor cannot be directly applied to using a GIS application from another vendor due to differences in fundamental conceptual approaches.

The RADGIS architecture has been shown to overcome the problem of application bloat by allowing application developers to rapidly develop small, customised GIS applications for novice users, or highly domain-specific systems for expert users. The development of small, customised GIS-type applications, which can be extended at runtime if necessary, increases the usability of the GIS application by reducing the number of extraneous features that are not required to provide the basic functionality required by novice users, in particular application domains. The development of smaller, customised GIS applications that are tailored to better fit the conceptual model of a specific application domain, e.g. environmental, geological, municipal, also increases the usability of GIS applications. However, the real power of the RADGIS architecture lies in its ability to allow the end-user to customise the application based on his/her requirements, at runtime. This ensures that the client application has the flexibility to withstand changing levels of expertise or user requirements.

As the level of expertise of end users increases, the knowledge gap between novice users and expert users increases. It is, therefore, extremely important to provide an extensible architecture that can cater for users with very different levels of competence, and their progression from novice to expert user. The RADGIS architecture, together with DGCML, provides a single extensible client that is able to accommodate the needs of both novice and expert users through:

?? the simplicity of its approach with respect to adding, removing and customising services. This allows unnecessarily complicated features that are not immediately required by the novice user to be left out of the initial GIS client, while retaining the option to add them later (as opposed to having to obtain a different client that provides additional functionality); as well as

?? providing increased flexibility and customisation for expert users who wish to get more out of an application than was originally intended by the developer. The level of customisation currently afforded by most applications typically only allows the end-user to make use of built-in functionality through scripting languages or to customize the overall look-and-feel of an application. RADGIS allows the user to add and replace services within the client application, as well as to decide on the level of integration of these new services with other services. This, in turn, facilitates the development of new services that the original developer of the system may not have envisaged.

In general, the core client application would have a fairly small software footprint that possibly only included the basic Geospatial Display Services. However, it could be extended with as much functionality as the user required, when the user required it. The adoption of a distributed GIS architecture also has the potential to reduce the software footprint on the client's machine, because a number of the tools would be accessed remotely.

The RADGIS client is therefore a hybrid client, which is neither a traditional "thin" client (simply a user interface), nor a "fat" client (user interface and all the application logic). Instead the client may be extended or trimmed down at various stages of its life, based on the user's requirements. It is also possible to load templates based on the task the user wishes to perform, i.e. a single DGCML descriptor could describe a whole menu of options which, when installed, would provide access to a customised set of tools that are required for a particular task.

The issue of interoperability has been addressed in this thesis by the adoption of open standards developed by the OGC and ISO/TC 211. In particular, the use of XML encoded geospatial data, e.g. using GML, will provide a simplified method of exchanging data between interoperable components, as well as simplify the visualisation of data on the Web, e.g. using SVG in a standard XML-enabled Web browser.

The adoption of standard interfaces when implementing components will also dramatically improve the ability to determine the substitutability and compatibility of components. The adoption of a single open standard, such as OpenGIS, will mean that any future components developed according to this standard would be interoperable with existing components, irrespective of who implemented them.

Another major problem with GIS applications, which relates to both traditional and Web-browser based client architectures, is the inability to provide location transparent access to distributed geospatial services. The RADGIS architecture makes use of DGCML descriptor files, which specify the use of local and/or distributed objects, to implement GIS services. The user is able to invoke these GIS services in the same manner regardless of whether they make use of local objects or remote objects, providing a high level of location transparency. While there may be some noticeable performance benefits from the ability to explicitly specify the best location for processing to occur, this option would generally only be made available to the expert user.

There is currently much research into distributed GIS (see *figure 7.1*) [Alameh 2001]. However, the RADGIS architecture improves on current models by providing a *runtime-extensible* distributed-GIS client, based on OpenGIS® standards. This client is easily extended through the addition of interoperable GIS services that are developed and deployed using DGCML. The DGCML descriptor files are straightforward to edit and allow the user or component integrator to customise GIS services. These changes do not require the RADGIS client to be restarted, but take effect the next time the tool is chosen from the menu options, providing *runtime* extensibility.

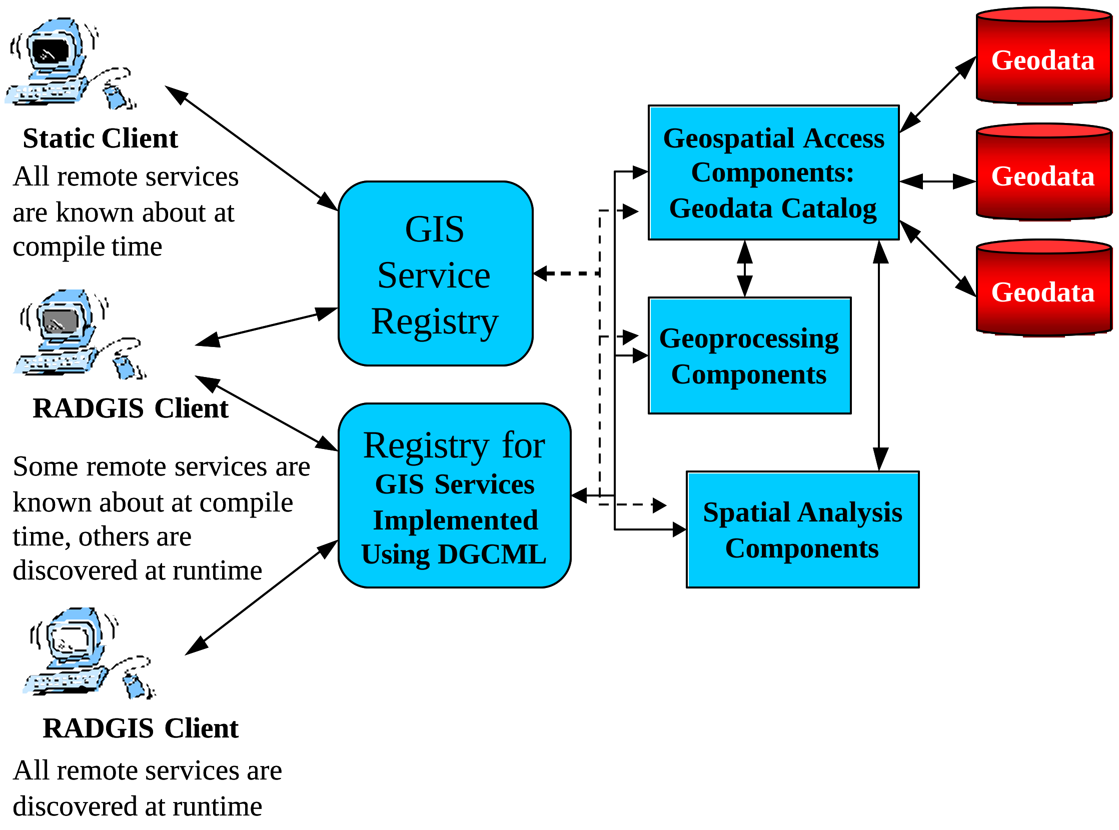

Static Client
All remote services are known about at compile time
RADGIS Client
Some remote services are known about at compile time, others are discovered at runtime
RADGIS Client
All remote services are discovered at runtime
GIS Service Registry
Registry for GIS Services Implemented Using DGCML
Geospatial Access Components: Geodata Catalog
Geoprocessing Components
Spatial Analysis Components
Geodata
Geodata
Geodata

Figure 7.1. Example of a distributed GIS architecture

Our research focused primarily on the ability to add GIS services, implemented as local classes that execute locally, or as distributed CORBA objects and EJBs, to an application at runtime. One of RADGIS's strengths is the ability of the client application to access remote objects at runtime for which no compile-time knowledge exists, i.e. the client application does not have access to the stubs for the remote objects. In this case one must assume that the client may have very little knowledge, if any, of how the methods of that object are to be invoked in order to provide the particular service required by the user, and whether or not the service requires its own Graphical User Interface (GUI).

We have looked at the implications this has on the design of the RADGIS client application and the distributed objects used to implement the GIS services. We have also shown that, while it is not practical to create an entire application at runtime, the benefits of being able to tailor an application at runtime, or to add functionality provided by different software vendors is both appealing and feasible.

The use of a distributed GIS model, as implemented in our RADGIS application, means that one can delegate a particular task to a platform or location that best fits the task at hand, based on processing and data requirements. Therefore, although particular optimisations will have to be user driven, as opposed to being location transparent, it will be possible for more than one implementation of a particular tool to be invoked. Such optimisations will, for example, depend on where the data source is located, or what type of operation is being performed, i.e. it may be more cost effective to transfer the data set to a more powerful machine for processing, than process the data locally.

Apart from the potential performance benefits to be derived from processing data on faster machines, or using the ability to perform parallel processing, the implementation of a distributed GIS architecture composed of interoperable components also provides accessibility to different vendors' implementations of a particular GIS service. This allows the application developer or end-user to choose a particular implementation based on the algorithms used, as well as factors such as whether or not the vendor supports dynamic upgrading, licensing restrictions, pricing models, and possibly even guarantees of Quality of Service in terms of speed or accuracy with which the algorithm is executed. It is also worth noting that some users may be willing to accept reduced efficiency, as a result of executing a

GIS service remotely, when offset against benefits such as reducing the total cost of ownership of GIS software, or eliminating the need to store, update and maintain local copies of geospatial data.

The ability to make use of the RADGIS architecture to create a client-side framework in which one could combine location services to create more sophisticated tools, as well as allowing further processing of the results obtained from invoking a particular location service, shows the tremendous flexibility of the RADGIS architecture. RADGIS is, in effect, a generic client architecture that is able to parse DGCML descriptors at runtime and, using Reflection, generate the GUI necessary to execute GIS services, location services or any other type of service described by a valid DGCML descriptor.

Therefore, we conclude that the original research objective stated in *chapter 1*, i.e. "**To develop a runtime extensible, highly-customisable, distributed-component based GIS and Location-based Service clients**", has been achieved.

7.1.1. Limitations of the research undertaken

Distributed GIS is an extremely broad field, which encompasses many different topics, and makes use of many different technologies, operating on a number of diverse platforms. However, due to limited time and resources, it was necessary to focus the research on specific issues. Therefore the following unresolved issues are considered outside the scope of the research that was undertaken:

?? A complete GIS application is an extremely complex system comprising many diverse functions. The RADGIS architecture, however, was implemented as a proof of concept system, comprising a generic framework and very specific tools that were developed to illustrate specific concepts. The RADGIS client is therefore not a fully-functional commercial GIS application.

?? Because the RADGIS client is not a complete GIS application, and we did not implement a wide range of tools for use with the client, it was not possible to test the RADGIS client under heavy load or perform meaningful usability studies.

?? The core of the RADGIS architecture, which allows the addition of GIS services to the client application at runtime, developed and deployed using DGCML, has been implemented and test as a whole. However, some of the components that facilitate the visualisation of geospatial data in formats other than VRML had to be tested separately, because the implementation of certain "bridging" components are not currently available, and certain standards have not been finalised. For example, while we have highlighted the benefits of using GML as a standardized data format for transferring geospatial data, we have not made use of it in the implementation of our RADGIS client. This is due to our focus on the visualization of 3D data, which is not currently supported in GML 2.0.

?? Although the specification of the DGCML meta-language using a DTD does not limit our RADGIS architecture, we acknowledge the potential benefits to be gained from converting it to an XML Schema-based specification.

?? We have not implemented, or made use of, a geoprocessing registry service for registering and discovering GIS services that have been implemented and deployed using DGCML.

?? We have not implemented mechanisms to deal with charging, transaction management, or security when dealing with distributed objects.

During our research we have identified two utilities that would be extremely useful for working with the DGCML meta-language, but which have remained unimplemented because they are not essential to the research that we have undertaken. They are:

?? a simple GUI editor that would allow the user to create GUIs by dragging and dropping Java Swing components, and then generate the equivalent DGCML description of the GUI. It would also allow the user to load existing DGCML GUI descriptions and modify them, thus simplifying the editing of DGCML descriptor files.

?? a code generation tool that allows a user to generate Java code from a DGCML descriptor, rather than interpreting the DGCML descriptor and then having to perform Reflection at runtime. Such a utility could be used to optimise the execution of frequently-used GIS services.

While we have not used the Simple Object Access Protocol (SOAP) or XML Remote Procedure Call (XML-RPC) to implement method-calls on remote objects, we acknowledge the tremendous impact these technologies are having on the implementation of distributed applications on the Web, and believe that the RADGIS client could easily be adapted to make use of these technologies. There is currently tremendous industry impetus behind the development of Web services, implemented using technologies such as the SOAP, Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI). There are also a number of research projects within the field of GIS that are starting to look at the benefits that can be derived from using such technologies and Web Service-based architectures.

7.2. Thesis contributions

Having summarised the findings of this thesis, it is now possible to explicitly highlight the contributions that our work makes to the fields of GIS and distributed component-based software development. Therefore, the following points are offered as the major contributions that have been presented in this thesis:

?? A thorough literature survey of current state-of-the-art GIS research and software architectures, including location-based services. In particular we have focussed on research undertaken by large standardisation bodies, such as the Open GIS Consortium, in the interests of promoting interoperability based on open standards.

?? In the development of our RADGIS architecture, we have highlighted the value of using XML as a transfer format for geospatial data (i.e. GML and X3D), and as a wiring language for combining (distributed) components to implement GIS services, in the evolution of GIS applications.

?? We have highlighted the current status and problems surrounding the visualisation of 3D geospatial data using GML, VRML, GeoVRML and X3D, as our RADGIS application aims to keep pace with developments in the field of 3D geospatial visualisation.

?? The development of a novel GIS client architecture, called RADGIS, which was designed to overcome the problems identified with current GIS applications. Its ability to allow the user to customise the GIS client at runtime, provides an extensible architecture that facilitates a high level of customisation, and allows the user to work with distributed geospatial data and services in a location-transparent manner.

?? The development of a markup language, called DGCML to facilitate the development and deployment of GIS services that can be integrated into the RADGIS client at runtime, including the ability to specify the GUI required by a GIS service as well as links to the necessary help files.

?? We have highlighted the implications of employing a distributed CBSD approach to developing client applications, as utilised in our RADGIS application, in order to draw attention to the need for further research on:

- distributed component-based charging mechanisms;
- the need to deal with issues of concurrency arising from the use of distributed components; and
- the ability to dynamically customise the user interface based on the types of tools that are available for processing the geospatial data.

The contributions of this thesis extend beyond distributed GIS architectures, and can be applied in the broader context of Web/distributed programming. They also takes cognisance of the changing trend in Web-based application development towards the implementation of distributed Web Services and "intelligent" clients, which is currently being realised through the development of Microsoft's .NET and Sun's Open Net Environment (Sun ONE) technologies. The increasing importance of being able to combine interoperable Web services, based on open standards, to facilitate the development and customisation of specialised client applications, signals a move away from the Web-browser as the client for Internet applications, towards more "intelligent" clients.

Appendix A – Mobile Agents

GIS applications are resource intensive by nature, i.e.:

?? GIS datasets consist of megabytes, possibly even terabytes of spatial and attribute data, necessitating the introduction of hardware-based data compression for storage optimisation.

?? GIS applications are computationally intensive due to the use of transcendental functions, complex transformations in map projection and high-level graphics rendering.

?? GIS applications require large-bandwidths between GIS users over Intranets and the Internet.

While rapid advances are being made in creating faster processors and secondary storage devices that have greater storage capacities, it is ultimately the legacy Internet networking structure that will create the bottleneck to high-bandwidth multimedia applications. This is particularly true of resource intensive GIS applications that require large spatiotemporal datasets to be transferred across the relatively limited bandwidth of much of the Internet (especially within South Africa). Thus, developing a distributed spatiotemporal GIS for use over the Internet provides constant challenges for optimisation, and necessitates a flexible architecture.

During the course of our research into distributed GIS architectures, we therefore also briefly looked at the use of Mobile Agents as a method for reducing Internet traffic. This appendix details some background material with respect to Mobile Agents, as well as some of the research that we undertook when investigating the integration of Mobile Agents into our Proof of Concept system.

One particularly useful scenario for which the use of Mobile Agents shows great promise, is for operations that require one to download a large data set in order to perform a relatively simple operation which returns a result that is generally an order of magnitude smaller than the data set. In such a scenario, it would be more efficient to transfer and execute the code necessary to perform the operation on the machine on which the data set resides (or possibly same intranet), rather than transferring the data set across the Internet, to the machine that was to perform the processing.

Background

Agents should be reactive, autonomous, goal-oriented and temporally continuous, i.e. agents should be continuously running processes that exercise control over their actions and are able to respond proactively to changes in the environment in order to achieve a particular goal. Agents can be classified according to the role they fulfil, for example different types of agent may exhibit communicative, learning, mobile, flexible and/or character qualities [Franklin and Graesser 1996].

Conde [1998] argues that the traditional distributed object paradigm is "a synchronous message-passing paradigm whereby all objects are distributed, but stationary, and interact with each other through message-passing". Even though the use of Java's Reflection mechanism for dynamically invoking EJBs, and CORBA's dynamic facilities, including Dynamic Invocation Interface (DII), and the Interface Repository, allow the creation of extremely flexible systems that allow runtime discovery and late-binding [Orfali and Harkley 1998], the objects themselves are still stationary.

While research into the use of agents is not new, according to Kurki [1998], mobile agents, i.e. agents that are able to migrate from one machine to another in a heterogeneous network, are an emerging technology that is attracting more and more interest from distributed systems researchers. Mobile agents are able to initiate their transfer to a different host and migrate the code, data and, in a system that supports strong migration, the execution state so that it can continue execution from where it stopped before the transfer.

In addition to mobility, agents also exhibit the following characteristics [Conde 1998] [Millman 1998]:

?? *Asynchronous*: a mobile agent can execute asynchronously as it has its own thread of execution.

?? *Discrete*: a mobile agent is invisible to the user and the system, providing location transparency.

?? *Flexibility*: a mobile agent can adapt to changing circumstances, e.g. it is able to work around broken links and downed servers. If the network connection is broken, and the agent needs to move, it can simply wait until the connection is restored.

?? *Local interaction*: a mobile agent generally moves to another location to interact with other mobile agents or stationary objects locally, rather than using remote message passing.

?? *Object-passing*: when a mobile agent moves, the whole object is passed, including its code, execution state, data, and travel itinerary.

?? *Persistence*: a mobile agent is autonomous and self-sustaining, i.e. it contains sufficient information to decide what to do, where to go, and when to go.

?? *Parallel execution*: it is possible to subdivide a task so that multiple agents can be dispatched to different sites to perform these sub-tasks in parallel, or even to perform multiple tasks in parallel.

?? *Secure*: when a mobile agent arrives at a host, it is subjected to the security restrictions of the context, a gateway between agents visiting the host and the host's resources, that provides an agent sandbox [Agosta 1998]. This ensures that mobile agents are resistant to interception and tampering, and that agents may only access particular resources subject to verification, e.g. digital signatures.

According to Conde [1998], there are many technical advantages of mobile agents, and there is no single alternative to all of the functionality they provide.

The Object Management Group (OMG) is currently working on the specification of an agent framework to support agent mobility via Mobile Agent System Interoperability Facilities Specification (MASIF) on top of the Common Object Request Broker Architecture (CORBA). GmbH Informations - und Kommunikationstechnologie [IKV++ 1998], are researching emerging telecommunications technologies such as the Telecommunications Information Networking Architecture (TINA), and have developed Grasshopper, which they claim to be the first mobile agent environment that is compliant to the OMG MASIF standard.

The use of Mobile Agents in GIS applications

A potential solution for reducing the high demands made on Internet bandwidth may lie in processing more information on the server side, to reduce the amount of extraneous data that is downloaded across the Internet to the client. This optimisation technique has tremendous potential in Web-based GIS applications because GIS data sets are often extremely large, and the user is only interested in a small subset of this information. Thus, if it is possible to process and refine the data required by the user on the server side, it could reduce the overall Internet bandwidth requirements.

Our research investigated means to reduce client-side setup and the demands for client-side processing power by distributing the services/GIS tools so as run on the most appropriate machine, e.g. on the same machine as the dataset, or a machine more appropriate to do intensive "number crunching".

Our approach identifies two major uses of mobile agents to reduce the amount of Internet bandwidth required by a user of a Web-based spatiotemporal GIS. The first is to move an agent to a server machine to perform a task on a data set stored on the server, and the second is to allow GIS tools to be moved to the client machine to execute locally. This is in contrast to the current CORBA paradigm where one would obtain an object reference to a GIS tool implemented as a remote object.

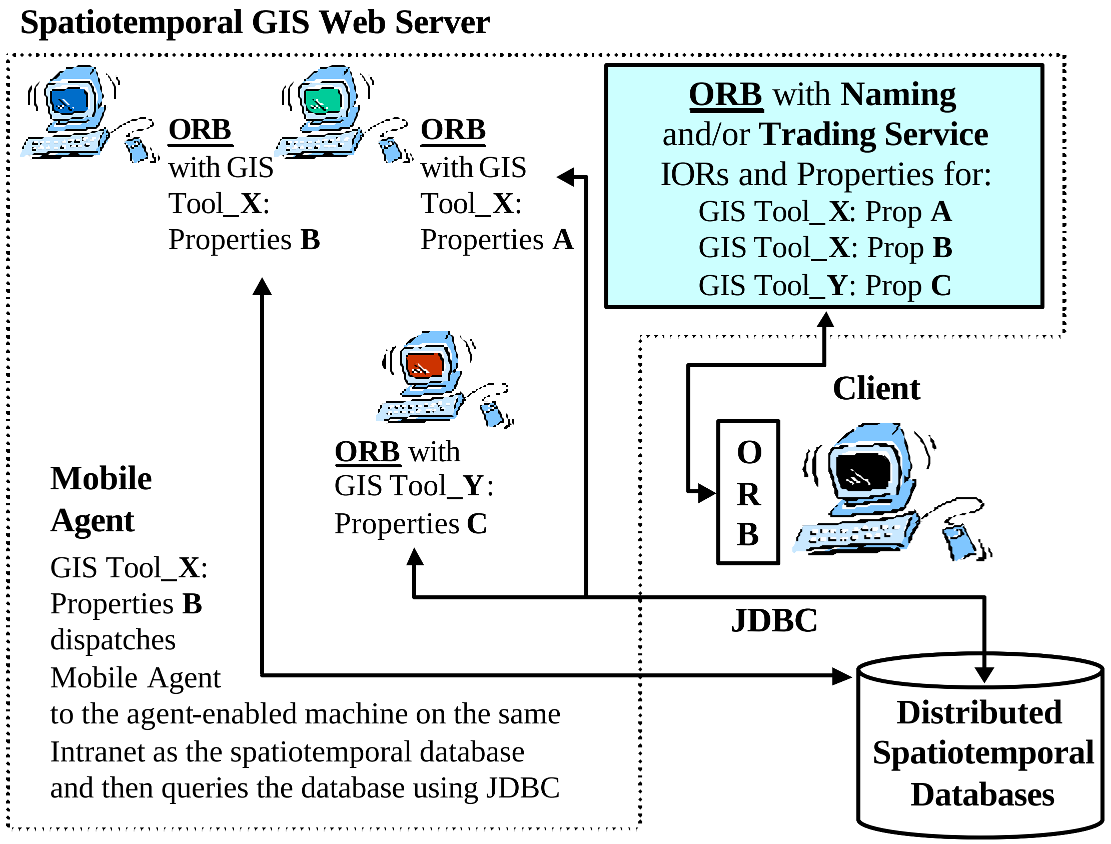
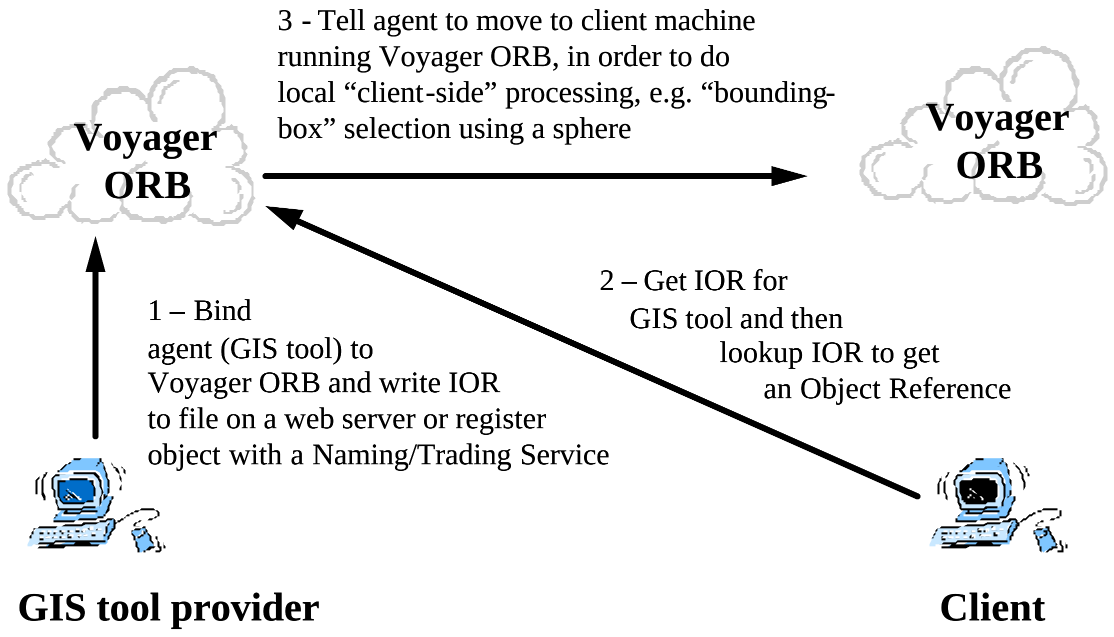

Spatiotemporal GIS Web Server
ORB with GIS Tool_X: Properties B
ORB with GIS Tool_X: Properties A
ORB with Naming and/or Trading Service IORs and Properties for:
GIS Tool_X: Prop A
GIS Tool_X: Prop B
GIS Tool_Y: Prop C
Client
ORB
ORB with GIS Tool_Y: Properties C
Mobile Agent
GIS Tool_X: Properties B dispatches Mobile Agent to the agent-enabled machine on the same Intranet as the spatiotemporal database and then queries the database using JDBC
JDBC
Distributed Spatiotemporal Databases

Figure A.1. Example of component interaction using a Mobile Agent

Figure A.1 shows the interaction of CORBA ORBs and Mobile Agents in our "Proof of Concept" Web-based spatiotemporal GIS. In particular it illustrates how the client can use the CORBA Naming and/or Trader Services to dynamically discover and invoke distributed GIS tools. Should one of these tools be implemented as a mobile agent, it is then also possible to dispatch the agent to a particular machine in order to perform the processing "locally" on that machine. This is done either to achieve parallel processing or to perform the processing of a dataset on the same machine as the dataset, or another agent-enabled machine on the same Intranet as the dataset, so as to minimise Internet bandwidth requirements.

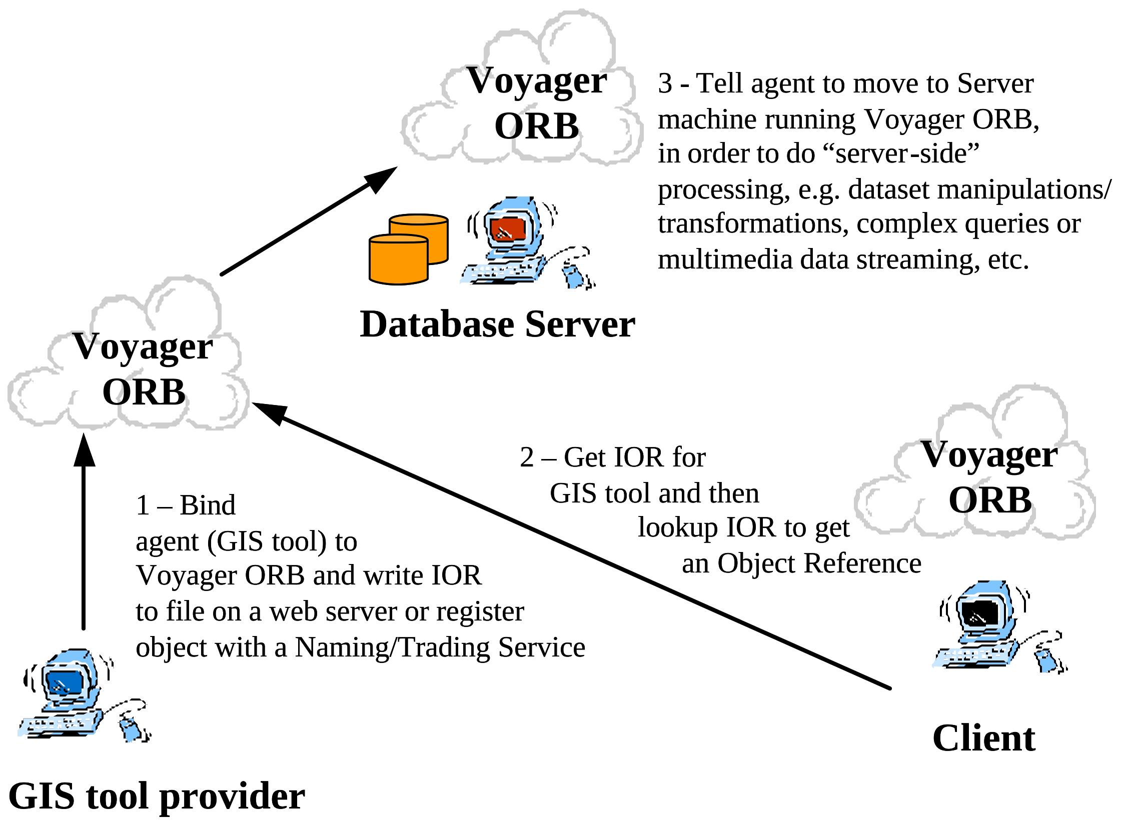


3 - Tell agent to move to client machine running Voyager ORB, in order to do local "client-side" processing, e.g. "bounding-box" selection using a sphere
Voyager ORB
Voyager ORB
1 – Bind agent (GIS tool) to Voyager ORB and write IOR to file on a web server or register object with a Naming/Trading Service
2 – Get IOR for GIS tool and then lookup IOR to get an Object Reference
GIS tool provider
Client

Figure A.2. Using Agents for "client-side" processing

Figure A.2 and figure A.3 show simplified examples of moving mobile agents to different machines in order to achieve different tasks, i.e. *figure A.2* illustrates the moving of an agent to the client machine in order to perform local processing, while *figure A.3* illustrates the moving of an agent to the machine on which the spatiotemporal dataset resides in order to perform "local" processing on the dataset.

It should be noted that the possibility exists for mobile agents to be used to process/collect data from multiple sources. Therefore, agents may travel to more than one machine in order to perform a specified task and then collate the results before returning home to deliver the final output.



Voyager ORB
3 - Tell agent to move to Server machine running Voyager ORB, in order to do “server-side” processing, e.g. dataset manipulations/ transformations, complex queries or multimedia data streaming, etc.
Database Server
Voyager ORB
2 – Get IOR for GIS tool and then lookup IOR to get an Object Reference
Voyager ORB
1 – Bind agent (GIS tool) to Voyager ORB and write IOR to file on a web server or register object with a Naming/Trading Service
Client
GIS tool provider

Figure A.3. Using Agents for “server-side” processing

According to the Voyager ORB Developer Guide [ObjectSpace Inc. 1998], one can expect a performance benefit of between 1 000 and 100 000 times when using local messaging as opposed to remote messaging. However, according to [Kurki 1998], even if the performance benefits become negligible, due to the communication overhead of transferring the agent being comparable to the overhead of transferring the data set across the network, it is still useful to have the code closer to the data.

The use of mobile agents can also improve fault tolerance or flexibility. For example when an agent reaches its destination, should the network go down, it can wait for the network to come up again and then send the result/move to its next destination. Thus any lengthy process that may be interrupted due to the network going down, and would have to be resumed at a later stage, could be better implemented using as agents to quickly traverse the network, as there is less chance of the network failing over a short period.

The ideal solution would, therefore, be a mix of stationary and mobile code that provided a single uniform paradigm for distributed object computing, including synchrony and asynchrony, message-passing and object-passing, for stationary objects and mobile objects within the framework of a web-based spatiotemporal Geographic Information System (GIS).

Appendix B - VRML Optimisations

VRML provides a number of mechanisms to increase downloading and rendering speed, such as proper decomposition of a scene, inlining, and streaming certain multimedia and graphics elements. In terms of efficiency, using compression schemes (such as gzip, binary format or geometry compression) or instancing (reusing parts of your scene, textures and multimedia elements) VRML can produce images three times smaller than GIFs. [Gorman 1997]

When discussing VRML optimisations, one must draw a distinction between optimising the time taken to download a virtual world (optimising bandwidth), and optimising the rendering of that virtual world.

Rendering Optimisations

It is important to identify where the bottlenecks for rendering a scene are, i.e. if the scene is co-ordinate (vertex) bound, texture-bound, or pixel-bound [Nadeau et al. 1996] [Silicon Graphics 1998].

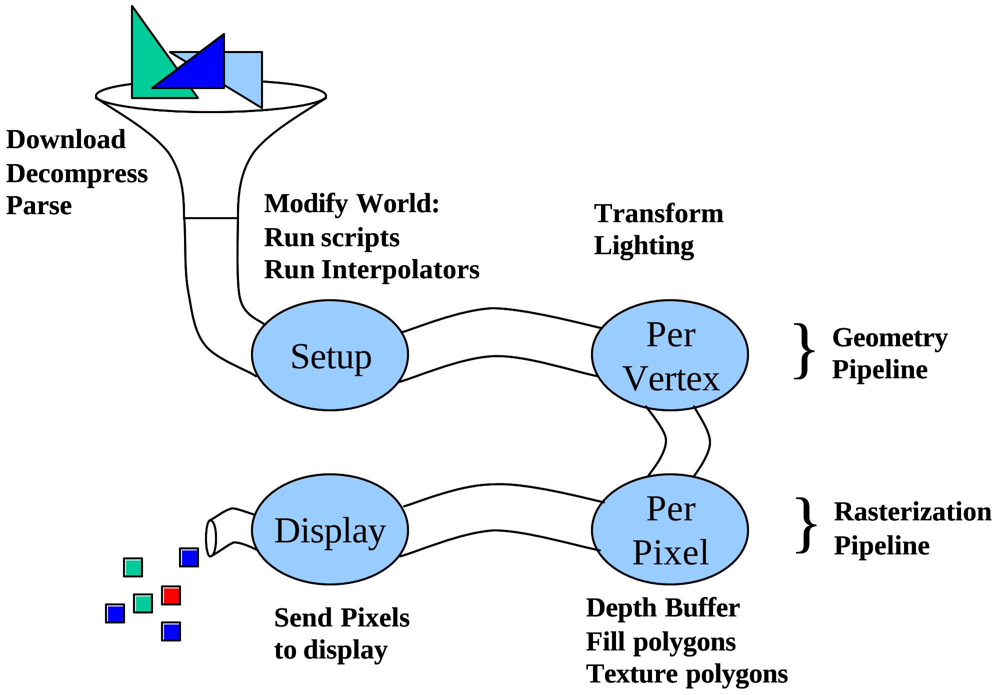


Figure B.1. VRML Rendering Pipeline [Silicon Graphics 1998]

Co-ordinate-bound

A scene is co-ordinate-bound if the computations involving co-ordinates are limiting the speed of the pipeline. Co-ordinates, and their associated texture co-ordinates and colours, are used in all pipeline stages. Therefore it is very important to try to avoid letting the scene become co-ordinate-bound.

Pixel-bound

The scene is pixel-bound if the computations needed fill triangles using triangle colours and textures in the Rasterize stage are limiting the pipeline speed.

Texture -bound

The scene is texture-bound if computations necessary to access textures in the Rasterize stage are limiting the performance of the pipeline.

It is also possible for the performance bottlenecks associated with rendering a particular scene to shift as the viewer moves through that scene. Consider, for instance, a detailed terrain on a virtual planet. When the planet is a distant dot, far from the viewer, the number of pixels drawn for the planet's geometry is very low, but the number of co-ordinates used to build that geometry is high. From such a distant vantage point, the planet will cause the rendering pipeline to be co-ordinate-bound. However, as the viewer moves closer, the planet's image on the screen grows in size, causing more pixels for the detailed terrain of the planet to be drawn, and the pipeline may become pixel-bound.

The tremendous range of viewing freedom offered to viewers of VRML worlds makes it difficult for VRML world authors to optimise their worlds so that they draw quickly regardless of the viewer's vantage point. Instead, VRML world authors strive to optimise for typical viewing situations. For instance, if the viewer is expected to walk from room to room in a virtual building, then the world's content can be optimised for this path. If the viewer unexpectedly dives through a wall and hovers in mid-air outside the building, drawing speed may decrease for this atypical vantage point.

Compression

The VRML 1.0 specification focuses on the specification of 3D graphics objects, while paying little attention to minimising download time. For example, the VRML files must be in ASCII format, which wastes bandwidth.

A number of current VRML browser plug-ins now support the downloading of gzipped world files. The use of gzip compresses the ASCII files, saving bandwidth and decreasing the download time. Should the plug-in be able to interpret the gzipped file rather than unzipping it first, it may also be feasible that the parsing time is decreased which ultimately leads to a decrease in the time taken to draw the scene initially.

According to Leung, they have been unable to find a browser implementation that can accept compressed VRML from a CGI program within a WWWInline node (see InLine below). When using a CGI program to create VRML data on the fly, this becomes a major drawback as it leads to long delays in the presentation of the data. These delays are due to both the CGI activation (latency) and the transmission delays (bandwidth) associated with the larger data size of the uncompressed VRML file.

Although compression of the ASCII files using gzip is supported, a compressed binary format would require far fewer bytes, and is an important consideration for accessing very large data models over the Web [Leung 1998].

Inline

The WWWInline node may be used for reducing the file size of a virtual world by allowing one to define a bounding box for the file it references. If the user never actually sees the bounding box (because of culling, or for any other reason), the inlined file might never be downloaded, which ultimately translates into a saving in bandwidth. The process of decomposing a scene into objects that can be inlined requires careful thought and planning, but the savings in bandwidth requirements and transfer time make it worth the additional effort [Matsuba and Roehl 1996].

Inlining also facilitates the re-use of objects if they are stored separately, as opposed to all being placed in one file, i.e. the same object (stored by itself) can be incorporated into more than one "virtual world" using the WWWInline node.

USE and DEF

The DEF keyword allows one to provide a node with a name that may be referenced later in the same file with USE or ROUTE statements. Instead of creating a copy of the node, the USE statement inserts the same node into the scene graph a second/third/etc. time, resulting in the node having multiple parents [VRML97].

Instancing (or the reuse of objects) is useful for traditional GIS because if symbols are used to represent features in a coverage, one can instance 3D symbols and then reuse them to save space and reduce processing overheads. However, this only works if you can find a generic symbol and are not too concerned about the actual dimensions of the object it is representing. For example you could use 3 different generic tree symbols to "represent" a forest. There are many objects in everyday use that are similar and can be defined once and used where needed. Thus, depending on the GIS application, the required Level of Detail (LOD), and the required accuracy of representation, a considerable saving in space and processing could be obtained by defining generic symbols and re-using them in the scene.

Shapehints

If you know that all the faces in an object have consistent vertex ordering, and/or that the shape only has an "outside" (i.e., the inside surfaces are not visible) then the ShapeHints node should be used to inform the browser of that information. This allows the rendering engine to avoid lighting surfaces that won't be seen, and possibly do backface culling. The result is a noticeable performance improvement [Matsuba and Roehl 1996].

Bounding boxes

Information about the size and location of the bounding box is used by the browser to decide if a WWWInline node is visible. In this way, it is possible that should the user never do a 360-degree turn, part of the scene need never be downloaded. Making "clever" use of bounding boxes is particularly useful for decreasing the amount of data that needs to be downloaded initially, and may result in savings of bandwidth should the user never get close enough, or look in the direction of the object represented by the bounding box.

Level of Detail (LOD)

An important feature of VRML, which increases the speed of visualisation, is based on using several different representations for a single object, i.e. VRML implements a mechanism to support level of detail (LOD) management. The level of detail defines how the object appears on display with respect of viewing distance. It means that the rendering software substitutes one model with less or more detail as user goes through the scene. Furthermore, objects can be defined as invisible from certain distance.

Note, that the number of faces in each scene with several LODs is higher than in a single-level scene description. Thus, a higher speed of interactive walking-through virtual worlds is achieved by increasing the amount of data. Although this last statement sounds confusing, it is correct. When using LOD, one increases the amount of data for that scene (multiple representations for the same object). However, at the same time, by allowing the browser to render objects at a lower LOD, one obtains an increase in performance. Thus, there is a trade-off between downloading additional LODs for a particular object, i.e. increased bandwidth, and increasing the rendering performance.

One can delay transferring the inlined file until the user gets close enough to see it, by placing a `WWWInline` statement inside a LOD node. One can even have multiple versions of an object at different levels of detail, and only transfer them as needed. Thus, the LOD node can be used to increase the power and efficiency of the Inline node. [Matsuba and Roehl 1996]

Progressive LOD

As an alternative to using VRML's built-in LOD, one can implement one's own LOD handling. This is useful if, for example, one wishes to implement a progressive LOD mechanism. Progressive LOD means that even if the user needs to view the next LOD, the whole object does not get downloaded again, only those parts that are different to the current LOD. This is similar in concept to video compression, where only the changes between two frames are stored/transmitted.

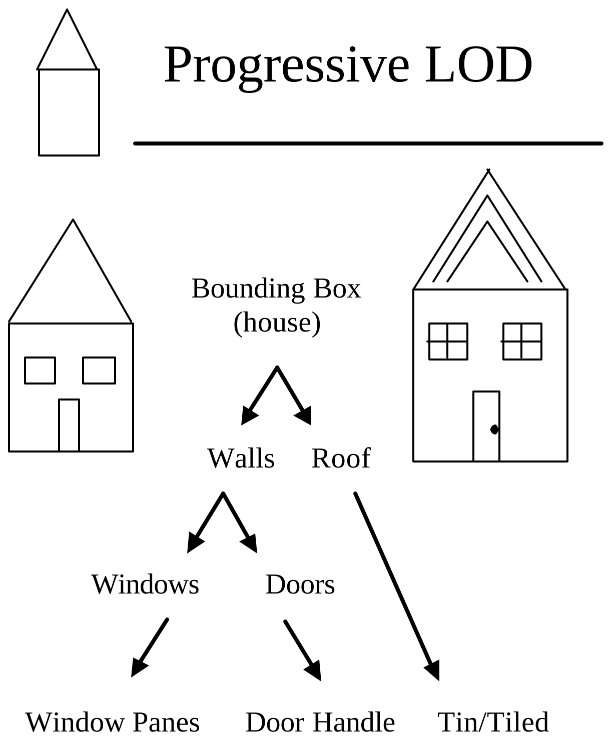


Figure B.2. Progressive LOD

Thus, there are two methods for implementing LOD:

?? Using VRML's LOD node: Redrawing of objects at varying LODs or specifying urls for each LOD. Each method ends up downloading the same base information each time.

?? Using VRML's createVRMLFromString/createVRMLFromURL: Specifying additional objects to be added to the original object in the scene graph as the LOD increases. This method decreases the bandwidth implications because only the changes need to be downloaded each time and not the entire object. It may also be less processor intensive because one is adding and removing portions of an object at different LODs as opposed to removing whole objects and then adding new objects. This is an issue that still needs investigation.

Linear interpolation

All VRML 2.0 interpolator nodes use linear interpolation to compute intermediate values between the key values you provide. A linear interpolator computes an intermediate position or orientation each time an output is needed. Any number of intermediate values can be computed between your key positions and orientations.

The use of interpolation is especially important when playing an animation at different speeds. For a quick animation, one's VRML browser may only have time to draw the world a few times between the time the animation starts and the time it stops. In this case, one's browser may only need to linearly interpolate values at a few fractional times between the key fractional times provided.

For a slow animation, one's VRML browser may have the time to draw the world many times and may need a large number of interpolated positions or orientations. In this case, one's browser may interpolate values at many fractional times between key fractional times.

Using keyframe animation and linear interpolation, one can describe an animation independent of the playback speed of the animation. During playback, an appropriate number of intermediate values are computed automatically.

Appendix C – GeoVRML Nodes

The following nodes are part of the GeoVRML 1.0 Recommended Practice document that has been submitted to ISO for inclusion as an amendment to the ISO VRML97 standard [Reddy *et al.* 2000b] [Reddy *et al.* 2000c]:

?? The **GeoElevationGrid** node - allows one to take the curvature of the earth into consideration when modelling large extents. It provides a height field representation for geospatial elevation data, offset from an ellipsoid model of the planet, in a number of geographic coordinate systems such as geodetic (latitude/ longitude) and Universal Transverse Mercator (UTM). In contrast, VRML's ElevationGrid node values are offset from a single flat plane [VRML97], and it is therefore unsuitable for representing geospatial data where the curvature of the earth needs to be taken into account.

?? The **GeoCoordinate** node - lets a modeller specify coordinates using geographic coordinate systems (e.g. geodetic, UTM) directly within a VRML file, as opposed to having to convert them to Cartesian coordinates first. This is useful for inserting output from devices such as GPS units, which normally output a location as a latitude/longitude coordinate, straight into VRML files. The GeoCoordinate node transparently converts the data into a Cartesian frame, and correctly positions the coordinates in the global model.

?? The **GeoLocation** node – allows the user to georeference an arbitrary VRML model, i.e. establish the relationship between the coordinates of the VRML model with a specific point on the earth. It also orients the model correctly, depending upon its position on the earth, to ensure that a model built using the standard VRML right-handed coordinate system will have its base correctly aligned with the surface of the earth.

?? The **GeoOrigin** node - As previously mentioned, VRML uses single-precision (32-bit floating-point numbers) to model and render all geometry [VRML97]. However this precision is not enough to accurately display geographic data at high resolutions. Thus one of the requirements for modelling geographic (e.g., geocentric) coordinates beyond a resolution of 10-100 m is the use of at least double-precision (64-bit floating-point numbers). The GeoOrigin node enables the accurate rendering of double-precision geographic coordinates by defining an absolute geographic origin (or GeoOrigin) in double-precision, and then for all double-precision geographic coordinates, the difference between each coordinate and the GeoOrigin is taken. The result is a single-precision offset that can be used for accurate rendering of that object. All GeoVRML nodes that deal with coordinates, e.g. GeoElevationGrid, GeoCoordinate, and GeoLocation, support the use of the GeoOrigin node.

?? The **GeoLOD** (formerly QuadLOD) node - provides the ability to browse multi-resolution, tiled terrain data, which is essential for memory management and scalability operations when browsing massive terrain datasets (e.g. Terravision II [Reddy *et al*. 1999b]. It automatically manages the progressive loading of higher-resolution data as the user approaches the terrain, and also unloads terrain data once the user has moved past.

?? The **GeoInline** node - is a grouping node that is used to decide when its children should be read from a location on the web. This is done either immediately when the node is first loaded, or at a later stage when for example, a VRML ProximitySensor triggers the required event.

In addition to these nodes, GeoVRML 1.0 also includes the **GeoPositionInterpolator** node to perform animations using geographic coordinates, the **GeoTouchSensor** node to return the geographic location at the current (mouse) pointer position, the **GeoViewpoint** node to specify a camera location in geographic coordinates, and the **GeoMetadata** node to provide a summary and links to full metadata descriptions of the geographic data [Reddy *et al*. 2000b]. For more information on the GeoVRML nodes, please refer to http://www.geovrml.org.

Appendix D - Dynamic Interface Development

According to Dey et al. [1997b], dynamic component integration shows the most promise for context-aware computing, which requires an infrastructure that permits intelligent mediation between software components. An example of such a system that has been developed is Cyberdesk [Dey et al. 1997b]. Cyberdesk is an adaptive interface that modifies the list of available tools at runtime, based on the user's current activity, using a dynamic mapping of user actions to possible user actions.

In order to illustrate the benefits and validity of such an approach, this appendix provides some background on adaptive and intelligent interfaces in order to make the reader aware of some of the research that has been performed in these fields. Much of this research was performed based on static, non-distributed applications, and thus the application of this research to the runtime extensible distributed applications involves additional complexity that must be taken into consideration.

Adaptive interfaces are a way of reducing the complexity of an application with respect to its usability [Browne *et al.* 1990] [Shneider-Hufschmidt et al. 1993]. Thus, research into adaptive and intelligent interfaces explores the following basic software usability issues [Encarnação 1997]:

?? Simplification of the design and implementation of "good" user interfaces,

?? The clearer and more efficient presentation of information,

?? Simplification of the interaction between the client and the application, and

?? The creation of interfaces that provide better support for a users' particular tasks.

On the other hand, intelligent user interfaces automatically adapt to the needs of different users, learn new concepts and techniques, anticipate the needs of users, accommodate the changing needs of users over time (for example, as the user's level of expertise progresses from novice to expert), take initiative and make suggestions to users, and provide explanations of their actions [Maes and Lieberman, *current*].

Research into adaptive and intelligent user interfaces also incorporates research from a number of other fields, particularly the use of agents and artificial intelligence. *Figure D.1* illustrates the relationship between adaptive and intelligent user interfaces as well as number of sub-disciples.

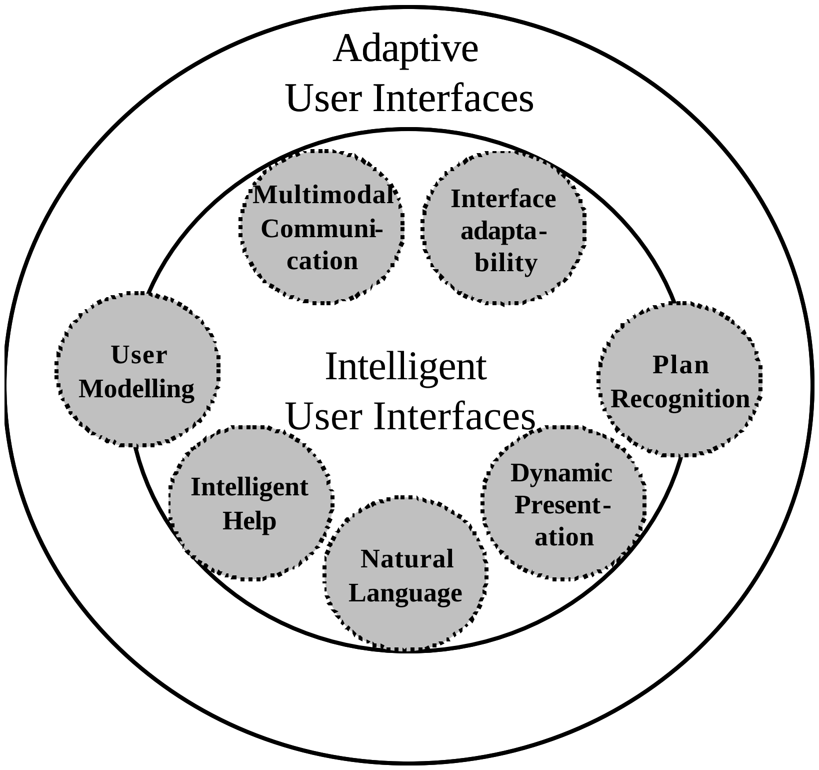


Figure D.1. Components of intelligent and adaptive user interfaces [Encarnação 1997]

The main benefits to be derived from these approaches are the simplification of the user interface, and customise the application based on the user's level of expertise and processing requirements. For example, an experienced user may be aware of his or her level of expertise and request full menus and brief prompts, whereas a novice may request short menus and lengthy prompts [Sukaviriya and Foley 1993].

More intelligent interfaces may also infer information about the user's level of expertise and customise the application based on how the user works with the application, and which tasks the user makes use of most frequently. This could be extremely useful for customising the GIS client based on the user's most frequently performed task, e.g. data capturing (digitising), or analysis, or presentation.

Appendix E – DGCML listing of a location service for finding restaurants

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE GISService SYSTEM "file://localhost/C:/describeService.dtd" >
<GISService name="Restaurant Location Service" vendor="RUDevGroup" version="2.0.2">
  <CodeBase url="iiop://localhost:900/RestaurantLocService" />
  <GUI>
    <Component name="MainFrame" type="Frame">
      <Property name="title" value="Restaurant Location Service"/>
     <Component name="ResultPanel" type="Panel" position="Center">
          <Property name="layout" value="java.awt.BorderLayout"/>
        <Component name="RLSResultsScrollPane" type="ScrollPane" position="Center" >
            <MethodCall ReturnValueDest="_tmpBorder2"
                        ReturnType="javax.swing.border.Border"
                        name="javax.swing.BorderFactory.createTitledBorder">
             <Param  DataType="java.lang.String" Source="Restaurant Information"/>
           </MethodCall>
          <Property name="border" value="_tmpBorder2" />
          <MethodCall ReturnValueDest="_tableModel"
                     ReturnType="RemoteResultSetTableModel" name="constructor">
            <Param DataType="java.lang.Integer.TYPE" Source="4" />
            <Param DataType="java.lang.Integer.TYPE" Source="3" />
          </MethodCall>
          <Component name="SQLResults" type="Table" position="Center" >
              <Property name="model" value="_tableModel" />
          </Component>
         </Component>
      </Component>
      <Component name="InputPanel" type="Panel" position="North">
        <Property name="layout" value="java.awt.BorderLayout"/>
        <Component name="UserLocation" type="TextField" position="North">
           <Property name="text" value="X-coordinate, Y-coordinate"/>
         <MethodCall ReturnValueDest="_tmpBorder1"
                    ReturnType="javax.swing.border.Border"
                    name="javax.swing.BorderFactory.createTitledBorder">
           <Param DataType="java.lang.String" Source="Enter your position here"/>
         </MethodCall>
         <Property name="border" value="_tmpBorder1" />
        </Component>
        <Component name="Radius" type="TextField" position="South">
           <Property name="text" value="distance in km"/>
          <MethodCall ReturnValueDest="_tmpBorder1"
                    ReturnType="javax.swing.border.Border"
                    name="javax.swing.BorderFactory.createTitledBorder">
            <Param DataType="java.lang.String" Source="Enter Radius here"/>
          </MethodCall>
          <Property name="border" value="_tmpBorder1" />
        </Component>
      </Component>
      <Component name="SubmitQuery" type="Button" position="South">
        <Property name="text" value="Search"/>
        <Event type="action">
```

```
         <MethodCall ReturnValueDest="_userLoc" ReturnType="java.lang.String"
                 name="constructor">
           <Param DataType="javax.swing.JTextField" Source="UserLocation">
             <Property name="text" value="java.lang.String"/>
           </Param>
        </MethodCall>
         <MethodCall ReturnValueDest="_userRadius" ReturnType="java.lang.String"
                 name="constructor">
           <Param DataType="javax.swing.JTextField" Source="Radius">
             <Property name="text" value="java.lang.String"/>
           </Param>
        </MethodCall>
        <MethodCall ReturnValueDest="_tmpRes" ReturnType="[Ljava.lang.String;"
                 name="remote:makeRestaurantQuery" >
          <Param DataType="java.lang.String" Source="_userLoc" />
          <Param DataType="java.lang.String" Source="_userRadius" />
        </MethodCall>
        <MethodCall ReturnValueDest="null" ReturnType="void" name="_tableModel.init">
          <Param DataType="[Ljava.lang.String;" Source="_tmpRes"/>
        </MethodCall>
        <MethodCall ReturnValueDest="null" ReturnType="void"
                 name="_tableModel.fireTableChanged">
          <Param DataType="javax.swing.event.TableModelEvent" Source="null"/>
        </MethodCall>
      </Event>
    </Component>
    <MethodCall ReturnValueDest="null" ReturnType="void" name="pack"/>
    <Property name="visible" value="true"/>
    <Event type="window" filter="windowClosing">
      <Property name="visible" value="false"/>
    </Event>
   </Component>
  </GUI>
  <Help>
   <HelpSet href="RestaurantLS.hs"/>
  </Help>
</GISService>
```

Glossary of Terms and Acronyms

"The beginning of wisdom is to call things by their right name."
Chinese Proverb

Accuracy - the quality of the result, or the degree of correctness of the measurement. It should be distinguished from precision, which relates to the quality of the process by which the result was obtained.

Algorithm - a statement of the steps to be followed to solve a problem.

API – Application Programming Interface

Applet – a Java program that is included in an HTML Web Page using the <applet> tag, and runs in a client's Web browser.

Attribute data - the a-spatial, descriptive information about features that is often used for analysis and the manipulation of the associated geospatial data.

BeanML – Bean Markup Language. A wiring language developed by IBM Alphaworks that allows one to describe an application composed of JavaBeans in XML.

Cartography - The art or technique of making maps or charts.

Catalog - A collection of entries, each of which points to a feature collection and describes its contents, coverages, and other metadata.

CBSD – Component-based Software Development. CBSD is the philosophy of building a system by assembling and integrating existing components rather than building the system from scratch. Also known as Component-Based Software Engineering (CBSE).

CCM – The CORBA Component Model is a core component of the CORBA 3 specification, developed by the OMG, which extends the basic architecture defined in the EJB Specification, and allows the creation of server-side scalable, language-neutral, transactional, multi-user and secure enterprise-level applications.

COM – Component Object Model. A specification developed by Microsoft for writing reusable software components that can be accessed and invoked in a Windows environment. Also see *DCOM*.

CORBA – The Common Object Request Broker Architecture, developed by the OMG, is an architecture and specification for creating, distributing, and managing distributed objects in a network.

COTS software – Commercial-off-the-Shelf software.

Datum – A point, line or surface that is used as a reference.

DCOM – Distributed Component Object Model. A distributed software architecture developed by Microsoft, based on COM, that provides the ability to perform remote procedure calls so that DCOM objects can run remotely over a network. Also see *COM*.

DEM – Digital Elevation Model. Digital elevation models are cartographic/geographic data in raster form that represent the elevation of a dry land surface, i.e. they are typically used to represent terrain relief, and are also referred to as Digital Terrain Models (DTM).

DGCML (Distributed GIS Component Markup Language) – a meta-language developed by the author to enable the creation and deployment of GIS services, based on local and distributed components, which can be incorporated into the RADGIS client.

DII – CORBA's Dynamic Invocation Interface. An API that allows a client to make dynamic method invocations on remote CORBA objects that were generally not known about at compile time.

Disaggregated – broken up into a number of constituent parts.

Distributed – the ability to access data and processing, as well as collaborate with other users located throughout the world, i.e. not concentrated in a single location.

DOM – Document Object Model. A platform- and language-neutral interface that allows programs and scripts to dynamically access and update a document's content, structure, and style. See *SAX*.

DTD – Document Type Definition. A type of file that defines the structure and properties of an XML document, and is used by a parser to validate the structure of an XML document. It has been superseded by the XML Schema Language. See *XML Schema*.

EAI – External Authoring Interface. A mechanism, developed by Chris Marrin, to allow Java programs to manipulate the VRML scenegraph in a VRML plug-in that supports the EAI.

EJB – Enterprise JavaBeans. A Java-based distributed object architecture developed by Sun Microsystems, that facilitates the development and deployment of reusable, object-oriented, server-side components.

Element – A component of an XML document that represents a logical data structure, delimited by start and end tags.

Feature – A digital representation of measurable or describable phenomena about a real world entity or an abstraction of the real world. It is the fundamental unit of geospatial information and consists of both spatial and attribute data.

.geo – Proposal by SRI for a new top level domain to simply indexing and discovery of spatial data.

Geocoding - the process of defining the positions of geographical objects relative to a standard reference datum.

Geodata – geographically related data

Geographic information system (GIS) - a computer hardware and software system capable of handling the storage, manipulation, analysis and display of spatial and related attribute data.

Geoprocessing – the processing of geographically related data

Georeference - to establish the relationship between page coordinates (i.e. x, y) of a planar map or image with known real-world coordinates (i.e. longitude/latitude, UTM, etc).

Geospatial data – spatial data that is referenced to the earth

GeoVRML – a 3D data format, based on VRML. It was developed by the GeoVRML Working Group to overcome the limitations of using VRML for large terrain visualisation.

GIS – see *Geographic Information System.*

GML – Geographic Markup Language. An XML encoding of the Simple Features Specification, developed by the OGC, which is likely to be widely adopted as a geospatial data exchange format.

GPS - Global Positioning System. A position-finding system, which uses a radio receiver to pick up signals from special satellites to compute the location of the receiver.

GUI – Graphical User Interface

HTML – Hypertext Markup Language. A very simple markup language used to format text, create form fields, and embed images, sound, and other multimedia files using URLs in a text file. This HTML file is generally downloaded across the Internet and interpreted by a Web browser.

HTTP – Hypertext Transfer Protocol. The Internet protocol used by Web browsers for fetching hypertext objects from remote hosts.

IDL – The Interface Definition Language is used to define interfaces that enable communication between modules implemented in different languages.

IIOP – Internet Inter-ORB Protocol. A protocol developed for communication between CORBA ORBs.

Interoperability - ability of software (possibly distributed on multiple machines) from multiple vendors to freely exchange data between systems.

IR – CORBA's Interface Repository. A service that contains all the registered CORBA objects' interfaces, as well as the methods they contain and the parameters they require.

ISO/TC 211 – The International Standardisation Organisation (ISO) technical committee that was formed to develop standards for working with geographic information. ISO/TC 211 is now working closely with the OGC to ensure that standardisation efforts are harmonised. ISO/TC 211 is currently concentrating more on data standards than the provision of GIS services and the standardisation of Web Map/Feature Servers, which is being looked at by the OGC.

J2EE – Java 2 Enterprise Edition is an environment for developing and deploying multi-tiered, Web-based enterprise applications.

JAR – The **J**ava **Ar**chive file format is essentially a ZIP file that contains Java classes and optionally a manifest file to describe the classes.

Java3D – 3D API for the Java programming language

JDBC – A Java API for database connectivity. Although actually a trademark name, it is often thought of as an acronym for Java Database Connectivity. Also see *ODBC*.

JNLP – Java Network Launch Protocol. JNLP is a web-centric software distribution protocol based on XML that enables the deployment of Web-based Java applications.

JVM – Java Virtual Machine. A specification for an abstract computing machine, which is implemented in software or hardware, that interprets Java programs that have been compiled into Java byte-codes.

Kriging - an optimised interpolation technique (after Dr. D. G. Krige) that uses information about the stochastic (random, local) aspects of spatial variation.

LDAP – The Lightweight Directory Access Protocol is an extensible client-server protocol and information model that allows one to access and manage information in a tree-structured database. Each entry in an LDAP server has a distinguished name that allows easy identification, and stores associated information as attributes. Each attribute has an associated type and one or more values.

Location-Based Services – The convergence of wireless communication, Web and GIS technologies that allows a user to gain access to data based on her/his (specified) location. Location Services are particular applications of spatial and analytic functions found in GIS applications.

LOD management – Level of Detail management is a graphics optimisation technique whereby an application renders an object at different levels-of-detail according to particular predefined criteria (e.g. performance requirements for rendering a scene, or the distance between the viewpoint and the object)

Marshal – Convert a request from its representation in the programming language to one that is suitable for transmission to the target object.

Metadata – data that describes the characteristics of an information or processing resource.

MLS - Mobile Location Service. See *Location Based Service.*

Mobile Agent – An agent that is able to migrate from one machine to another in a heterogeneous network.

Naming Service – A Service that allows objects to be named by means of binding a name to an object reference. A client can obtain a reference to a desired object from the Naming Service by simply specifying the name of the object.

ODBC – Open Database Connectivity. A standard API for accessing data in both relational and nonrelational DBMS, i.e. it provides the programmer with a standardised manner of accessing data in an underlying database, which is independent of that database's data storage format and programming interface. Also see *JDBC*.

OGC – Open GIS Consortium. A not for profit trade association whose purpose is to promote interoperability within the field of GIS through the creation an open standards GIS.

OLE – Microsoft's Object Linking and Embedding is a way to create documents containing objects from other programs.

OMG – Object Management Group. A non-profit organisation whose charter is to standardise and promote the use of object-oriented technology.

OpenGIS® Specification – A software interface standard developed by the Open GIS Consortium that enables interoperable geoprocessing and data sharing between GIS systems from different vendors.

Open system – A system that complies with standards, which have been made available throughout the industry, and therefore can be connected to other systems that comply with the same standards.

ORB – Object Request Broker. The infrastructure that connects objects requesting services to objects providing them, in a distributed environment.

OSD – Open Software Description. An XML-based language for automated software distribution over the Internet, developed by Microsoft.

PDA – Personal Digital Assistant.

RADGIS – Runtime Application Development of GIS. The runtime-extensible GIS client architecture developed by the author.

Raster – A data structure composed of a grid of cells that represent geographic features. A group of cells with the same value represents a feature. See *Vector*.

RMI - Remote Method Invocation. Java's distributed programming architecture.

SAX – The Simple API for XML is a standard interface for event-based parsing. See *DOM*.

SDK – The Java Software Development Kit is a set of Java class libraries, help documentation and the Runtime-environment, which is used by a Java application developer.

Sequential - one after the other, in tandem order.

SGML – Standard Generalized Markup Language. SGML is a vendor, platform, and media independent standard for documents based on DTDs. It was adopted as an ISO standard in 1986, and is the predecessor of XML.

Soap – The Simple Object Access Protocol is a method of making remote procedure calls over the Internet using HTTP.

Spatial analysis – the process of applying analytical techniques to geospatial data. Spatial analysis may be used to model, examine and interpret complex geographical interactions, make decisions based on spatial relationships or make predictions about future events. This is the essence of Geographic Information Systems, and is what distinguishes them from automatic map-making systems.

Spatial data – the locations of geographical entities together with their spatial dimensions. Spatial data may be vector (points, lines, areas or surfaces) or raster data (bit-mapped data).

Spatiotemporal GIS – A GIS which models features that may change shape or position over time.

SQL (Structured Query Language) – a powerful query language supported by most relational databases.

Standard – A definition or format, approved an authority or accepted as a de facto standard by industry.

SVG – Scalable Vector Graphics is a language for describing two-dimensional vector and mixed vector/raster graphics in XML.

Swing – GUI API for Java programming language.

Temporal GIS – A GIS that allows the user to work with data that has a time component.

TIN - Triangular Irregular Network. A method of creating a 3-D surface from irregularly spaced point data in a *vector* data model. See *DEM*.

Trader Service - a brokerage facility that allows objects to publicise their services and bid for jobs.

UDDI – The Universal Description, Discovery, and Integration standardis a repository-based directory service which facilitates the automated lookup of Web Services.

Unmarshal – Convert a request from a client, from its transmissible form to a programming language form.

Vector – One of the fundamental ways of representing and storing spatial data (the other being raster). It is a coordinate-based data structure that is comprised of a series of points (coordinates), some of which are joined by lines (i.e. sets of related points), and some line segments (arcs) are joined to form polygons. See *Raster*

Voyager – Application server developed by Objectspace.

VRML – The Virtual Reality Markup Language, is an ISO standard for displaying 3D objects over the Web.

VRML NG – VRML Next Generation

W3 – World Wide Web Consortium. Responsible for maintaining and developing emerging Internet standards, including any new standards for HTML.

Web Feature Server – originally intended as a major extension to the Web Map Server specification, it has now become a separate interface specification. The Web Feature Server specification, developed by the OpenGIS Consortium, enables a client to specify a request that returns a feature set, e.g. as GML, to the client.

WMS – Web Map Server. A specification developed by the OpenGIS Consortium for an online service that is able to provide maps in one of a number of standard image formats, e.g. GIF, JPEG, PNG, or as vector-based graphical elements, e.g. using SVG.

WWW – World Wide Web. Otherwise referred to as the Internet.

X3D – an XML-based version of VRML NG

XML – Extensible Markup Language. XML is subset of SGML, and does not require a document to have an associated schema described in a DTD.

XML Schema – It has recently replaced the DTD as the recommended practice for specifying the format, and allowable datatypes, in an XML document. See *DTD*.

XSL – Extensible Style Language. Is responsible for how the XML data is presented to the user.

XSLT – Extensible Style Language Transformations. This is the language responsible for transforming XML documents in one format into other XML documents, e.g. GML to SVG.

XwingML – A wiring language developed by HP Bluestone Software, which enables users to build XML documents that define a complete Java Swing GUI.

References

Adler, S. et al., 2000, "Extensible Stylesheet Language (XSL) Version 1.0", W3C Working Draft 27 March 2000, http://www.w3c.org/TR/xsl.

Agosta, L., 1998, "Advances in Web Computing", DM Review Magazine, June 1998, http://www.dmreview.com/issues/1998/jun/articles/jun98_70.htm

Alameh, N., 2001, "Scalable and Extensible Infrastructures for Distributing Geographic Information Services on the Internet", PhD Thesis, http://web.mit.edu/nadinesa/oldWWW/thesis

Albrecht, J., 1996, "Universal GIS Operations: Task-oriented systematization of GIS functionality", http://www.ncgia.ucsb.edu/~jochen/diss/dissabst.html

America, P., 1990, "Designing an object-oriented programming language with behavioral subtyping.", In der Bakker, J., de Roever, W. and Rozenberg, G., editors, Foundations of Object-Oriented Languages, 1990 REX School/Workshop, Noordwijkerhout, The Netherlands, number 489 in LNCS, Springer-Verlag, pp. 60-90.

Asaipillai, C., 1997, "The History of Distributed Object Oriented Technologies", http://www.metronet.co.uk/weegee/OOD.htm

Associated Press, 2001, "IBM Turns to (Server) Farming", Wired News, http://www.wired.com/news/technology/0,1282,45769,00.html?tw=wn20010802

Bangay, S., 1997, "30 Days to Build a City: Building Virtual Worlds from Maps", Computer Science Department, Rhodes University.

Barroca L., Hall J., and Hall P.(Editors), 2000, "Software Architectures: Advances and Applications", Springer-Verlag London Limited.

Basili, V., Briand, L., and Melo, W., 1994, "How Reuse influences productivity in Object-Oriented Systems", Communications of the ACM, Vol. 37 No. 5, 1994, 104-115.

Batory, D., and Geraci, B. J., 1996, "Validating Component Composition in Software System Generators", in Proceedings of the Fourth International Conference on Software Reuse, IEEE Computer Society Press, April 1996.

Bergner K., Rausch A., Sihling M., and Vilbig A., 1999, "Componentware - Methodology and Process", CBSE 99, Proceedings of the International Workshop on Component-Based Software Engineering: Held in conjunction with the 21st International Conference on Software Engineering (ICSE99) Los Angeles, CA, USA, May 17-18.

Berre, A., Grønmo, R., Hoff, H., Solheim, I., Lantz, K., and Østensen, O, 2000, "DISGIS: An Interoperability framework for GIS**",** SINTEF Telecom and Informatics, GIS Denmark, Norwegian Mapping Authority, http://www.gsdi.org/docs/capetown/abstracts.htm

Buehler, K., and McKee, L. (Editors), 1996, "The OpenGIS Guide", OGIS Project Technical Committee of the Open GIS Consortium, Inc., OGIS TC Document 96-001, http://www.opengis.org/techno/guide/guide1.htm

BlueStone, 1999, "About HP Bluestone XwingML", http://www.bluestone.com

BlueStone, 2000, "XML FAQ", http://www.bluestone.com/downloads/doc/051900_XML-FAQ.doc

Booch G., Rumbaugh J., and Jacobson I., 1999, "The unified modeling language user guide", Addison-Wesley.

Bray, T., Paoli, J. and Sperberg-McQueen, C., 1998, "Extensible Markup Language (XML) 1.0", W3C Recommendation, http://www.w3c.org/TR/REC-xml.

Bray, T., Hollander, D., Layman, A. (Editors), 1999, "Namespaces in XML", Recommendation, World Wide Web Consortium, Jan 1999, http://www.w3.org/TR/REC-xml-name

Brown, A., (Editor), 1997, "Component-Based Software Engineering", IEEE Computer Society Press.

Brown, A., and Wallnau, K., 1998, "The Current State of Component-Based Software Engineering", IEEE Software, September/October 1998.

Browne D., Norman M., and Totterdell P. (Editors), 1990, "Adaptive User Interfaces", Academic press, ISBN 0-12-137755-5.

Brox, C., and Kuhn, W., 2001, "Marketplaces for Geographic Information", In the proceedings of the 4th AGILE Conference, Brno, Czech Republic, http://agile.uni-muenster.de/Conference/Brno2001/New_Economy.pdf

Brutzman, D., and Williams, J., 2001, "How to install and compile Xj3D", Web3D Consortium, http://www.web3d.org/TaskGroups/x3d/Xj3D/HowToInstall.html

Buziek, G. and Hatger, C., 1998, "Interactive animation and digital cartometry by VRML 2.0 and JAVA within a temporal environmental model on the basis of a DTM of the Elbe estuary and a 12 hour tide period", University of Hannover, Germany, http://visart.ifk.uni-hannover.de/~buziek/COMVIS/COMVIS98/buziek/comvis98.html

Cagle, K., 2000, "A Tale of Two Parsers", Webtechniques, http://www.webtechniques.com/archives/2000/07/progrevu/

Canal, C., Fuentes, L., Troya, J, and Vallecillo, A., 2000, "Extending CORBA interfaces with p calculus for protocol compatibility", In Proceedings of TOOLS Europe 2000, IEEE Press, France, pp. 208-225.

Cimetiere, J., 2001, "The Web Services Value Chain", Intranet Journal, http://www.intranetjournal.com/articles/200108/tm_08_29_01a.html

Clements P.C., Bass L. [et al.], 2000, "Constructing Superior Software", MacMillan Technical Publishing, Indianapolis.

Conde, J., 1988, "Mobile Agents in JAVA", CERN/IT/ASD/RD45/98/12, December 1998, http://wwwinfo.cern.ch/asd/rd45/white-papers/9812/agents2.html

Coppit, D., and Sullivan K., 2000, "Multiple Mass-Market Applications as Components", ICSE 2000, ACM, Limerick, Ireland.

Cox, S., 2000, "Geospatial Data Transfer", CSIRO, http://www.ned.dem.csiro.au/XMML/about/geospatial.html

Daisey, P., 2000, GeoJava: "Where will you be tomorrow?", Geography Division, U.S. Census Bureau, Java Location Services, http://www.jlocationservices.com/company/Census/where_will_you_be.html

Davis, S., 2000, "OpenGIS Simple Features For CORBA work in DSTO", http://www.opengis.org/techno/interop/dsto/

Dhara, K., and Leavens, G., 1996, "Forcing behavioral subtyping through specification inheritance. In Proceedings of the 18th International Conference on Software Engineering (ICSE-18), Berlin, Germany, IEEE Press, pp. 258-267.

Dey, A., Abowd, G., Pinkerton, M., and Wood, A., 1997a, "CyberDesk: A Framework for Providing Self-Integrating Ubiquitous Software Services", ACM Inc., UIST.

Dey, A., Abowd, G., Pinkerton, M., and Wood, A., 1997b, "CyberDesk: Automated Integration of Desktop and Network Services", CHI 97, ACM.

Doyle, A., 2001, "An Introduction to the Open GIS Consortium and its programs", Proceedings of Digital Earth Conference, Fredericton, Canada.

Duman, A., et al, 1998, "Are CORBA Services Ready to Support Resource Management Middleware for Heterogeneous Computing?", IEEE, Proceeds of the Eighth Heterogeneous Computing Workshop.

Dykes, J. A., Moore, K. M. and Fairbairn, D., 1999, "From Chernoff to Imhof and Beyond: VRML & Cartography". In Proceedings of the Fourth Symposium on the Virtual Reality Modeling Language, Paderborn, Germany. pp. 99-104.

Ekins, C., and Davis, S., 1999, "The EXC3ITE Geospatial Services Segment", Information Technology Division, Australian ESRI & ERDAS User Conference, DSTO Australia.

Encarnação, M., 1997 "Concept and realization of intelligent user support in interactive graphics applications", PhD Thesis, der Eberhard-Karls-Universität zu Tübingen, http://www.crcg.edu/company/staff/mencarna/publs/diss/diss.html

Fairbairn, D., and Parsley, S., 1997, "The use of VRML for cartographic presentation", Computers & Geosciences, Vol. 23 No. 4, pp. 475-481, http://www.elsevier.nl/homepage/sad/cageo/cgvis/fairbair/vr_carto.htm

Fan, M., Stallaert, J., and Whinston, A., 2000, "The adoption and design methodologies of component-based enterprise systems", European Journal of Information Systems, Vol. 9 No. 1, pp. 25-35.

Fayad, M., Schmidt, C., and Johnson R., 1999, "Building Application Frameworks", Wiley Computer Publishing, John Wiley & Sons, Inc.

Ferris, N., 1998, "GIS Gets Down to Business", Managing Technology, http://www.govexec.com/tech/articles/0998mantech1.htm

Fingar, P., Clarke, J., and Stikeleather, J., 1997, "Object Magazine", April 1997, http://home1.gte.net/pfingar/obj_mag_497.htm

Franklin S., and Graesser, A., 1996, 'Is it an Agent, or just a Program?", A Taxonomy for Autonomous Agents, Institute for Intelligent Systems, University of Memphis, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag.

Fischer, G., 1994, "Domain-Oriented Design Environments", Automated Software Engineering, Johnson, L., and Finkelstein, A., (Editors), Kluwer Academic Publishers, Vol. 1 No. 2, June 1994.

Gabriel, P., and Wagner, R., 2001, "GIS meets E-commerce: Pricing in a distributed environment", Fraunhofer ISST, Proceedings of Digital Earth Conference, Fredericton, Canada.

Galdos Systems Inc., 2001, "Why GML?", http://www.galdosinc.com/technology-whygml.html

Gamma E., Helm R., et al., 1995, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley.

Garlan, D., and Perry, D., 1995, "Introduction to the Special Issue on Software Architecture", IEEE Transactions on Sofiware Engineering, Vol. 21 No.4, April 1995, pp. 269–74.

Gartner Group, 1997, "Componentware: Categorization and Cataloging," Applications Development and Management Strategies Research Note, by K. Loureiro and M. Blechar, December 5, 1997, http://www.gartnergroup.com

Gardels, K., *current* 26/10/2001, "The Open GIS Approach to Distributed Geodata and Geoprocessing", http://www.regis.berkeley.edu/gardels/envmodel.html

GeoEurope, 2000, "Geofocus: GIS in business", April 2000, available at http://www.geoplace.com

GeoInformatics, 2001, "Review of the Autodesk GIS 2000 International User Group Conference", Conferences and Meetings, GeoInformatics Online Issue, December 2000, http://www.geoinformatics.com/issueonline/issues/2000/12_2000/pdf_12_2000/conf2_8.pdf

Gifford, F., 1999, "Internet GIS Architectures-Which Side Is Right for You?", Maxim Technologies Inc., *http://ceiba.cc.ntu.edu.tw/GIS/Architecture.htm*

Goodchild, M., Egenhofer, M., Fergeas, R., 1997, "Interoperating GISs", Report of a Specialist Meeting Held under the Auspices of the Varenius Project Panel on Computational Implementations of Geographic Concepts, California, http://www.ncgia.ucsb.edu/conf/interop97/interop_toc.html

Gorman, T., 1997, "Why VRML is more than eye candy", http://www.netscapeworld.com/nw-07-1997/nw-07-vrml.html

Grady, R., *current* 26/10/2001, "Geoengineering meets Information Technology Mainstream", Applied Geographics, Inc. Environmental and Geographic Information Systems, http://www.bentley.com/geosummit/white/geowhite.htm

Griscom, D., 1999, "Code Signing for Java Applets", Suitable Systems, http://www.suitable.com/Doc_CodeSigning.shtml

Gunther, O. and Muller, R., 1999, "From GISystems to GIServices: Spatial Computing on the Internet Marketplace", *Interoperating Geographic Information Systems*, Edited by Goodchild, M., Egenhofer, M., Fegeas, R. and Kottman, C., Kluwer Academic Publishers, pp. 427-442.

Hall, R., Heimbigner, D., and Wolf, A., 1999, "A Cooperative Approach to Support Software Deployment Using the Software Dock", Proc. of ICSE'99: The 1999 Int'l Conf. on Software Engineering, Los Angeles, CA, pp. 174-183.

Heiler, S., 1995, "Semantic interoperability", ACM Computing Surveys, Vol. 27 No. 2, pp. 265-275.

Herzum P., Sims O., 2000, "Business Component Factory", John Wiley & Sons Inc.

Hernández, J., Troya, J., and Vallecillo, A., 2000, "Lesson 3: Component Interoperability", http://www.lcc.uma.es/~av/Docencia/Doctorado/tema3.pdf

Heywood, I., Kemp, K., Reeve, D., 1998, "Interoperable Education for Interoperable GIS", Interoperating Geographic Information Systems, Kluwer Academic Publishers.

Hurwitz, J., 1998, "Component Directions", DBMS Online, May 1998, http://www.dbmsmag.com/9805d04.html

IBM, 2001, "Visualize: Using dynamic e-business to open new markets for existing products", Case Study, http://www-4.ibm.com/software/solutions/webservices/casestudies/visualize.html

ICANN Committee, 2001, "Reconsideration Request 00-14", ICANN, March 2001, http://www.icann.org/committees/reconsideration/rc00-14.htm

IDC (The International Data Corporation), 2000, "1999 Worldwide Spatial Information Management Markets and Trends", http://www.idc.com

IKV++ (GmbH Informations - und Kommunikationstechnologie), 1999, Grasshopper: The Agent Platform, Germany, http://www.ikv.de/products/grasshopper/grasshopper.html

ISO/TC 211, 2000, "Draft Business Plan of ISO/TC 211 - Geographic Information/Geomatics", 27/11/2000, http://www.statkart.no/isotc211/

Kamthan, P., 2000, "XML Euphoria in Perspective", http://tech.irt.org/articles/js203/index.htm

Kim, K., Lee, K., Lee, H. and Ha, Y. 1998, "Virtual 3D GIS's Functionalities Using Java/VRML Environment", GIS Lab. Image Processing Dept. Systems Engineering Research Institute, S.Korea, Published in: J. Strobl and C. Best (Eds.), 1998: Proceedings of the Earth Observation & Geo-Spatial Web and Internet Workshop '98 Salzburger Geographische

Materialien, Volume 27. Instituts für Geographie der Universität Salzburg. ISBN: 3-85283-014-1.

Koeppel, I., *current* 26/10/2001, "What are Location Services? - From a GIS Perspective", ESRI Location Service Industry Manager, http://www.geojava.net/company/esri/What%20are%20Location%20Services.html

Kreveld, M, 1996, Variations on sweep line algorithms: Efficient computation of extended viewsheds and class intervals, Proceedings of the seventh International Symposium on Spatial Data Handling, pp. 843-855

Kurki, T, 1998, "Java and (mobile) software agents", Research Seminar: Java-based Software Technologies, Tik-76.270 http://smartpush.cs.hut.fi/tjk/javaagents.htm

Lake, R., 2000, "GML – The Future of Internet Mapping", Galdos Systems Inc., http://www.web-mapper.com/articles/GML.html

Lake, R., 2001a, "GML 2.0 – Enabling the Geo-spatial Web", Galdos Systems Inc., http://www.geojava.net/company/galdos/articles/GML3.htm

Lake, R., 2001b, "The hitchhiker's guide to the new web mapping", GeoPlace.com, http://www.geoplace.com/ge/2001/0201/0201web.asp

Langran, G., 1992, "Time in Geographic Information Systems", Technical Issues in GIS, Taylor & Francis, London, UK.

Leavens, G., and Sitaraman, M., (Editors), 2000, "Foundations of Component-Based Systems", Cambridge University Press.

Leclerc, Y. (Primary), Reddy, M., Iverson, L., and Eriken, M., 2001, "The GEOWEB (aka dot-geo) – Indexing data on the internet by location", SRI International, Proceedings of the Digital Earth Conference, Fredericton, Canada.

Lemmens, R., 2001, "Distributed interoperable processing in GIS: Towards Plug-and-Play GIS components", International Institute for Aerospace Survey and Earth Science, http://www.itc.nl/sitacs/research/Qualifier%20presentation%20abstract%20for%20webpage.htm

Leung, A., 1998, "Interactive viewing of 3D terrain models using VRML", School of Computer and Information Science, Syracuse University, http://www.dhpc.adelaide.edu.au/reports/043/html/index.htm

Liron, T., 2000, "Launching into Java - New client-side technologies bring Java apps out of the Web and onto the desktop", http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-launch_p.html

Liskov, B. and Wing, J., 1994, "A behavioral notion of subtyping", ACM Transactions on Programming Languages and Systems, Vol. 16 No. 6, pp. 1811-1841.

Location Interoperability Forum, http://www.locationforum.org

Looney, M., Lunga, S., and Budgen, D., 1998, "Software Component Reuse in Open System Software Design: A UK Perspective", http://www.dodccrp.org/Proceedings/DOCS/wcd00000/wcd000fb.htm

Lopez, X., 2000, "GeoJava for Internet and Mobile Location Services", Oracle Corp., 2000, http://www.geojava.net/company/Oracle/geojava_for_Internet_and_Mobile _Location_Services.html

Maes, P., and Lieberman, H., *current* 26/10/2001, "Intelligent Interfaces", Intelligent Interfaces seminar, MIT Media Laboratory, http://lcs.www.media.mit.edu/people/lieber/Teaching/Int-Int/Int-Int-Announcement.html

Maler, E., 2000, "Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1", W3C, February 2000, http://www.w3.org/XML/1998/06/xmlspec-report.htm

Marrin, C., and Campbell, B., 1997, "Teach Yourself VRML2 in 21 Days", Sams Net, ISBN 1-57521-193-9.

Marshall, J., 2000, "Developing Internet-Based GIS Applications", http://www.giscafe.com/TechPapers/Papers/paper058/

Martin, D. and Higgs, G., 1997, The Visualisation of Socio-Economic GIS Data Using Virtual Reality Tools, *Transactions in GIS*, Vol. 1 No. 4, pp. 255-266.

Matsuba, S., and Roehl, B., 1996, "Special Edition Using VRML", Que Publishers, ISBN: 0-7897-0494-3.

Megginson, D., 2000, "SAX 2.0: The Simple API for XML", http://www.megginson.com/SAX/

Miller, D., and Schirnick, H., 1999, "Smallworld and OpenGIS", Navigant Consulting Inc., http://www.smallworld.co.uk/large_docs/sw99_americas/Deb_Miller.pdf

Millman, H., 1998, "Agents at your service", http://www.idg.net/gomail.cgi?id=9-47025

Morais, M., 2000, "Realizing the Benefits of an N-Tiered Enterprise GIS", GIS at About, http://www.about.com

Nadeau, D., 1997, "Publishing 3D content with VRML", NASA EOSDIS Desktop Computing Workshop, July 1997, http://www.sdsc.edu/~nadeau/Talks/NASA_EOSDIS/java3d.htm

Nadeau, D., Ames, A., and Moreland, L., "Optimizing the Performance of VRML Worlds", http://linux.tomsk.ru/docs/programming/DrDobbs/articles/1996/9607/9607a/9607a.htm

Niagara SIG Meeting, 2000, (Special Interest Group) CBDi Buying & Selling Components, September 2000.

Niemeier, D. and Beard, K.,1993, "GIS and Transportation Planning: A Case Study", Computers, Environment, and Urban Systems, 17, pp. 31-43.

ObjectSpace Inc., 1998, "Voyager ORB 3.0 Developer Guide", USA.

Öhrström, P., 2001, "Investigating the Feasibility of an OpenGis GeoBox Prototype", Kungl Tekniska Högskolan Institutionen för geodesi och fotogrammetri, Universitetsservice US AB, Stockholm.

OMG (Object Management Group), 1999, "CORBA Components and Component Model", document orbos/99-02-05, available at http://www.omg.org/

Open GIS Consortium, 1998a, "The Benefits of OGC Membership", http://www.opengis.org/info/benefits/Benefits.rtf

Open GIS Consortium, 1998b, "OpenGIS Simple Features Specification for CORBA", Revision 1.0.

Open GIS Consortium, 1999, "The OpenGIS Abstract Specification Topic 0: Abstract Specification Overview", Version 4, OpenGIS Project Document Number 99-100r1, http://www.opengis.org/techno/spec/99-100r1.pdf

Open GIS Consortium, 1999, "The OpenGIS Abstract Specification Topic 12: Service Architecture", Version 4, OpenGIS Project Document Number 99-112, http://www.opengis.org/techno/spec/99-112.pdf

Open GIS Consortium, 2001a, "The OpenGIS Abstract Specification Topic 12: Service Architecture", Version 4.1, OpenGIS Project Document Number 01-112, http://www.opengis.org/techno/spec/01-112.pdf

Open GIS Consortium, 2001b, "OpenGIS Web Map Server Interfaces Implementation Specification", Revision 1.1.0, http://www.opengis.org/techno/specs/01-047r2.pdf

Orfali, R. and Harkey, D., 1998, Client/Server Programming with Java and CORBA, 2nd Edition, Wiley Computer Publishing.

Ousterhout, J., 1998, "Scripting: Higher Level Programming for the 21st Century", IEEE Computer, Vol. 31 No.3, pp. 23–30.

Panel-GI, 2000, “A Guide to GI and GIS”, Pan European Link for Goegraphical Information, http://www.unigis.hu/library/PANEL_GI_book.pdf

Peuquet, D. and MacEachren, A., 1998, “An integrated approach for representation and analysis of Space/Time Environmental Data”, http://www.geog.psu.edu/apoala/abstract.htm

Perry, D., 1989, “The Inscape Environment”, in Proceedings of the 11th International Conference on Software Engineering, IEEE Computer Society Press, May 1989.

Pesce, M., 1997, “VRML and Java - A marriage made in heaven”, http://developer.netscape.com/viewsource/pesce_vrml2/pesce_vrml2.html

Pharoah, A., Seigel, J., and Brooke, C., 2000, “Creating Commercial Components: CORBA Component Model (CCM)”, ComponentSource, http://www.componentsource.com/BuildComponents/WhitePapers/CORBAWhitePaper.asp

PortWood, M., 2000, “Using Java^{TM} Technology Reflection to Improve Design”, JavaOne, Presentation made at Java Developer Conference, http://servlet.java.sun.com/javaone/javaone2000/pdfs/TS-706.pdf

Preston, M., Clayton, P. and Wells, G., 1997, “Distributed Virtual GIS”, In the proceedings of the Teletraffic ’97 Conference, Grahamstown, South Africa.

Preston, M., Clayton, P. and Wells, G., 1998a, “Web-based Spatiotemporal GIS”, In the proceedings of the Masters and Ph.D. Conference, Stellenbosch, South Africa.

Preston, M., Clayton, P. and Wells, G., 1998b, “An overview of Temporal Object Models and their application to VGIS”, In the proceedings of the SATNAC ’98 Conference, Cape Town, South Africa.

Preston, M., Clayton, P. and Wells, G., 1999a, “Using CORBA to implement distributed GIS tools for use in a Web-based Spatiotemporal GIS”, In the proceedings of the Masters and PhD ’99 Conference, Golden Gate National Park, South Africa.

Preston, M., Clayton, P. and Wells, G., 1999b, “Integrating CORBA and Mobile Agents as a method for reducing Internet Bandwidth”, In the proceedings of the SATNAC ’99 Conference, Durban, South Africa.

Preston, M., Clayton, P. and Wells, G., 1999c, “Creating a Web-based Spatiotemporal GIS using Java and VRML”, In the proceedings of the WebNet ’99 Conference, Honolulu, Hawaii.

Preston, M., Clayton, C. and Wells, G., 2000, "Dynamic run-time application development using CORBA Objects and XML", In the proceedings of the SATNAC 2000 Conference, Stellenbosch, South Africa. http://www.cs.ru.ac.za/research/Preston_satnac2000.pdf

Preston, M., Clayton, C. and Wells, G., 2001a, "Location Services: The importance of location in the location-transparent medium of the Internet", In the proceedings of the SATNAC 2001 Conference, Wild Coast, South Africa.

Preston, M., Clayton, C. and Wells, G., 2001b, "Dynamic run-time application development using CORBA Objects and XML in the field of Locations Services and Distributed GIS", Submitted to the International Journal of Geographic Information Science.

Prieto-Diaz, R., and Neighbors, J. M., 1986, "Module Interconnection Languages", Journal of Systems and Software, Vol. 6, 1986, pp. 307–334.

Raj, G., 1998, "The EJB Model", Gopalan Suresh Raj's Web Cornucopia, http://www.execpc.com/~gopalan/java/ejb/ejbmodel.html

Radiya, A. and Dixit, V., 2000, "Get started using XML Schema instead of DTDs for defining the structure of XML documents", AvantSoft, Inc., http://www-106.ibm.com/developerworks/xml/library/xml-schema/

Reddy, M., Iverson, L., and Leclerc, Y., 1999a, "Enabling Geographic Support in Virtual Reality Modelling with GeoVRML", SRI International, Technical Note #13 in the journal Cartography and Geographic Information Science, Vol. 26 No. 3, July 1999, http://www.ai.sri.com/~reddy/pubs/ica/

Reddy, M., Leclerc, Y., Iverson, L., and Bletter, N., 1999b, "TerraVision II: Visualizing Massive Terrain Databases in VRML", IEEE Computer Graphics and Applications (Special Issue on VRML), Vol. 19 No. 2, pp. 30-38.

Reddy, M., and Iverson, L., 2000a, "Indexing and Using 3D GeoData on the Web", SRI International.

M. Reddy, L. Iverson, and Y. G. Leclerc, 2000b, "Under the Hood of GeoVRML 1.0", In Proceedings of The Fifth Web3D/VRML Symposium, Monterey, California.

Reddy, M., Iverson, L., and Leclerc, Y., 2000c, "GeoVRML 1.0 – Adding Geographic support to VRML", GeoInformatics Magazine, September 2000, http://www.geoinformatics.com/issueonline/issues/2000/09_2000/pdf_09_2000/art3_6.pdf

Reddy, M., 2000, "Update on GeoVRML 1.0", ACM Siggraph 2000, BOF Presentation.

Resnick, R., 1996, "Bringing Distributed Objects to the World Wide Web", http://www.interlog.com/~resnick/javacorb.html

Rhyne, M., 1997, "Going virtual with geographic information and scientific visualization", Computers & Geosciences Vol. 23 No. 4, pp. 489-491, http://www.elsevier.nl/homepage/sad/cageo/cgvis/rhyne/rhyne.htm

Rhyne, M., and Fowler, T., 1998, "Examining Dynamically Linked Geographic Visualisation", http://www.epa.gov/vislab/svc/publications/awma-gisvis.html

Ring K., and Ward-Dutton N., 1998, "Componentware – Building it, Buying it, Selling it", Ovum Ltd.

Robertson, A., Sale, R., Morrison, J. and Muehrche, P., 1984, "Elements of Cartography". New York, John Wiley & Sons.

Rohaly, T., 2000, "Client-side Java makes a comeback - Java Web Start, a new product from Sun, aims to breathe life back into client-side Java", http://www.javaworld.com/javaworld/javaone00/j1-00-webstart.html

Roman, E., 2000, "Mastering EJB", Elliott, R. (Editor), John Wiley & Sons, Inc., New York, ISBN 0-47133229 – 1.

Shneider-Hufschmidt M., Kühme T. & Malinwski U., 1993, "Adaptive user interfaces: principles and practice", Human factors in technology, ISBN 0-444-81545-7.

Schmidt, D., 2001, "Overview of CORBA", http://www.cs.wustl.edu/~schmidt/corba-overview.html

Schneider, J. and Nierstrasz, O., 1999, "Components, Scripts and Glue", in Software Architectures – Advances and Applications, Leonor Barroca, Jon Hall, and Patrick Hall (Editors.), Springer, pp. 13–25.

ScreamingMedia, 1999, "CORBA Component Model Will Help Developers Quickly Design and Implement Mission Critical Distributed Systems, *Business Wire*, September 02, 1999, http://industry.java.sun.com/javanews/stories/story2/0,1072,18380,00.html

Sessions, R., 1998a, "COM and DCOM", John Wiley Press.

Sessions R., 1998b, "Component-Oriented Middleware", Component Strategies, October 1998.

Siegel, J., 2000, "CORBA 3: Fundamentals and Programming" OMG Press, John Wiley & Sons, Inc.

Siegel, J., 2001, "What's coming in CORBA 3.0", OMG, July 2001, http://www.omg.org/technology/corba/corba3releaseinfo.htm

Silicon Graphics, 1998, “VRML Development with Cosmo Worlds”, Technical Education, Part number: VRMLDCW-1.0-6.2/3/4-S-SD-SW, http://corsi.cineca.it/dispense/worlds/html/

Snodgrass, R., Boehlen, M., Jensen, C., and Steiner, A., 1998, “Transitioning Temporal Support in TSQL2 to SQL3”, Temporal Databases: Research and Practice, Etzion, S. Jajodia and S. Sripada, Springer Verlag, LNCS 1399 March 1998.

Software AG, *current* 26/10/2001, “XML – The Benefits”, http://www.softwareag.com/xml/about/xml_ben.htm

Spruit, S., 1997, “Reflections on Java, Beans, and relational databases”, JavaWorld, http://www.javaworld.com/javaworld/jw-09-1997/jw-09-reflections.html

SRI Internet Initiative, 2000, “The Proposed .geo Top-Level Domain Name”, SRI International, http://www.dotgeo.org/proposal/html/contents.html

St.Laurent, S., 1999, “Java, XML, and a New World of Open Components”, New York Developers Group, http://www.simonstl.com/articles/nycod/sld001.htm

Stojanovic, Z., 2000, “Generic Component-Based Framework for Effective Telematics Application Development”, http://www.betade.tudelft.nl/projects/Proposal_Stojanovic_20001115.htm

Sukaviriya, P., and Foley, J., 1993, “Supporting Adaptive Interfaces in a Knowledge-Based User Interface Environment”, Intelligent User Interfaces ’93, ACM.

Sun MicroSystems, 2001a, “Java^{TM} Web Start Software Delivers Full-Featured Applications With a Single "Click", LOS ANGELES, CA Spring Internet World, http://java.sun.com/pr/2001/03/pr010314-02.html

Sun MicroSystems, 2001b, “Enterprise JavaBeans technology”, http://java.sun.com/products/ejb/

Sun Microsystems, 2001c, “JavaHelp”, http://www.javasoft.com/products/havahelp/

Szyperski C., 1998, “Component Software: Beyond Object-Oriented Programming”, ACM Press, Addison-Wesley.

Szyperski, C., 2000, “Components versus objects”, ObjectiveView #5, pp. 8-16.

Toon, M., 1997, The World by your Window, GIS Europe, Vol. 6, No. 11, pp. 38-41.

Traynor, C., and Williams, M., 1995, “Why are Geographic Information Systems Hard to use?”, ACM, CHI’95 Mosaic of Creativity.

Tremblett, P., 1998, “Java Reflection Not just for tool developers”, Dr. Dobb’s Journal, January 1998, http://www.ddj.com/articles/1998/9801/9801c/9801c.htm

Tsou, M. and Buttenfield, B., 1998, "Client/Server Components and Metadata Objects for Distributed Geographic Information Services". Proceedings, GIS/LIS '98, Fort Worth, TX, November, pp. 590-599, http://map.sdsu.edu/publications/Tsou-GIS98.pdf

UDDI.org, 2000, "UDDI Executive White Paper", http://www.uddi.org/pubs /UDDI_Executive_White_Paper.pdf

van Hoff, A., Partovi, H., and Thai, T., 1997, "The Open Software Description Format (OSD)", Submitted to W3C 13 August 97, http://www.w3.org/TR/NOTE-OSD.html

Veryard R., 2000, "Plug and Play: Towards the Component-Based Business", Springer London, in preparation, November 2000.

Voss, H., and Birlinghoven, S., 2000, "CommonGIS - Common Access to Geographically Referenced Data", Fourth Global Spatial Data Infrasture Conference - GSDI-4, Cape Town, South Africa, http://www.gsdi.org/docs/capetown/ abstracts.htm

Vretanos, P. (ed.), 2001, "OpenGIS Discussion Paper #01-023: Web Feature Service Draft Candidate Implementation Specification 0.0.12", January 2001, http://www.opengis.org/techno/discussions.htm

VRML97, ISO/IEC 14772-1:1997, 1997, "The Virtual Reality Modeling Language", http://www.vrml.org/Specifications.

W3C DOM WG, 2001, "Document Object Model (DOM)", September 15, 2001, http://www.w3.org/DOM/

Wahl, M., Howes, T., and Kille., S., 1997, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt

Wang, X., Yang, C., and Liu, D., 2001, "Web Mapping with Geographic Markup Language", Institute of Remote Sensing Applications, Proceedings of Digital Earth Conference, Fredericton, Canada.

Web3D Press Release, 2001, "New-Generation X3D Open Web3D Standard Launched with Leading Browser-Company Support", August 2001, http://web3d.org/fs_X3Dpressrelease8_01.htm

Webtechniques 1999, "Freeware Tool Xwings Two Ways", http://www.webtechniques.com/archives/1999/04/newsnotes/

Whitlock, N., 2001, "XML Schema becomes W3C Recommendation", Casaflora Communications, May 2001, http://www-106.ibm.com/developerworks/xml/library/x-schrec.html?open&l=132

Wiederhold, G., Wegner, P., and Ceri, S., 1992, "Toward Megaprogramming", Communications of the ACM, Vol. 35, No. 11, November 1992.

X3D FAQ, 2001, "X3D (Extensible 3D) Frequently Asked Questions (FAQ)", Compiled by Martin Reddy, Version 1.17, Aug 9 2001, http://www.web3d.org/TaskGroups/x3d/faq/index.html

Yellen, D., and Strom, R., 1997, "Protocol specifications and component adapters", ACM Transactions on Programming Languages and Systems, Vol. 19 No. 2, pp. 292-333.