Enhancing Data Discovery Using The ArcSDE Java™ API

G.E.Hobona*, S.J.Abele, Philip James
Geomatics@Newcastle, School of Civil Engineering and Geosciences,
University of Newcastle upon Tyne, NE1 7RU

Abstract

This paper describes the development of a set of interactive web based preview tools for enhancing the data discovery process to aid the metadata search facilities offered by the North East Environmental Data Server (NEEDS). NEEDS is a multi-tier environmental data clearinghouse based at the University of Newcastle upon Tyne, UK. The research outlined above demonstrates an implementation of the ArcSDE Java API to enable the NEEDS data browser to catalogue and preview ArcSDE data-sources. Due to the variable nature of environmental data, a requirement was identified to offer a single integrated view of multiple source and format datasets, allowing users to visually detect *implicit* relations between datasets. The catalogue is presented as HTML with visual previews as interactive Java Applets. This paper will discuss the lessons learnt during the development process and the authors' suggestions for extending the ArcSDE Java API.

1.0 Introduction

A spatial data clearinghouse is an electronic facility for searching, viewing, transferring, ordering, advertising, and disseminating spatial data from numerous sources via the Internet (Crompvoets, 2003). The efficiency of clearinghouses is dependent on users knowing what Geographic Information (GI) data exists, the source of the data, its possible uses and in what form it is. The process of searching through a clearinghouse for relevant datasets is known as 'data discovery'. Data discovery is carried out from a geographic data portal, which acts as the Human Computer Interface (HCI) between the user and the data repository. The implementation of a data clearinghouse provides a number of challenges to a great many developers. This paper describes a set of interactive web-based data discovery tools that were developed as part of the North East Environmental Data Server (NEEDS). A primary aim of the NEEDS clearinghouse was to offer environmental researchers tools for finding datasets held by NEEDS as well as third-party spatial data repositories from a single interface. A secondary aim of the clearinghouse was to investigate and develop new and innovative ways of searching spatial data repositories and previewing datasets held in repositories. NEEDS is a multi-tier environmental data clearinghouse that includes amongst others an ESRI ArcSDE data source. The data portal provides a single interface for browsing metadata held on heterogeneous servers. This paper outlines how the ArcSDE Java API was used to catalogue ArcSDE layers. Also discussed in this paper, is a facility for allowing users to export layers into formats that can be viewed from web browsers.

2.0 The Catalogue Facility

As NEEDS was to incorporate the use of database servers from different vendors including MySQL, ESRI ArcSDE and Microsoft SQL Server, it became apparent that a portal that was able to catalogue and preview datasets held in all these would have to

be developed. Whereas ArcIMS 4 can be used to view ArcSDE layers, it requires that projects or AXL files be set up. For this research, a system that automatically created previews of layers during each request was required. This preview would need minimal human input. The technology that was selected for this task was the ArcSDE Java API that is included in the ArcSDE client package. The next sections describe how a catalogue system was developed that allows users to search for layers on an ArcSDE data source.

2.1 Listing All Layers

To maintain platform independence and reduce the need for plug-ins, it was decided that the mode of presentation of the catalogue should be the HyperText Markup Language (HTML) and that each layer should be served as a JPEG image embedded in an HTML page. The technologies selected for generating the dynamic HTML pages were the Java Servlet and Java Server Pages (JSP) APIs. Servlets are server-side components that are invoked on the server when a client visits the servlet's web address. In comparison to servlets, JSP pages provide a fast and simplified way of creating web page content. Although JSP often look more like a regular HTML file than a servlet, behind the scenes, the JSP page is automatically converted to a normal servlet, with the static HTML simply being printed to the output stream associated with the servlet's service method (Hall, 2000). Figure 1 illustrates the client-server architecture adopted in implementing NEEDS and indicates the point at which ArcSDE was incorporated into the multi-tier architecture.

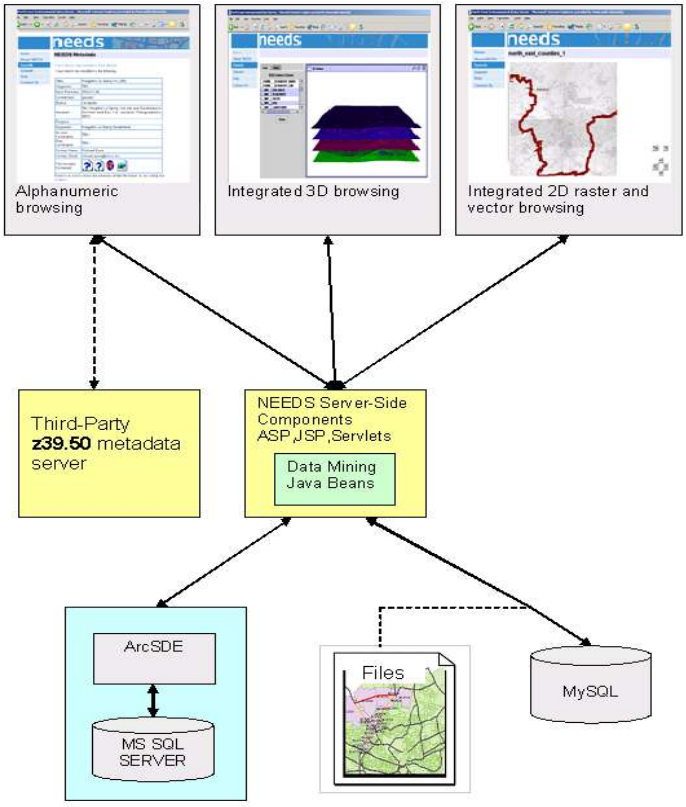

Figure 1. The architecture (Hobona et al, 2004)

A servlet was created that lists all available layers on an enterprise geodatabase. The catalogue page was implemented using the `SeConnection` class for representing an ArcSDE data source. An `SeConnection` object references a particular ArcSDE instance hence any number of `SeConnection` objects can be created to reference different instances of ArcSDE in a distributed information system. An issue with this class is that it accepts the ArcSDE username and passwords as parameters, hence it is important to ensure that these are served securely. Uses of this class on client-side Java applications such as applets is highly discouraged as the classes can be decompiled once they have been uploaded onto the client and thus leaving your passwords exposed. Creating `SeConnection` objects on the server-side in servlets or Java beans offers more security as the classes are not uploaded onto the client during execution. Java beans are java classes that adhere to the JavaBean specification in which they are defined as reusable software components (Sun, 1997). They can be instantiated by either servlets or JSP pages when the servlet or JSP pages are invoked.

An `SeConnection` object offers a `getLayers` method that returns a collection of ArcSDE feature classes (layers) wrapped in `SeLayer` objects. A loop was created to enumerate through the collection of layers. The name of each layer was then extracted from the `SeLayer` object using its `getName` method. Each name of a layer was made to be a hypertext link to another servlet-generated page that provided a preview of the layer.

This section described how the ArcSDE Java API was used to generate dynamic web-content for creating a catalogue of layers held on an ArcSDE data source. It also discussed the security issue that needs to be taken into consideration when using the `SeConnection` class. An alternative to listing all available layers was to implement an ability to search for particular layers by name or by spatial coverage. The next section describes how the search facility was implemented.

2.2 Searching For Specific Layers

As mentioned in section 1, a secondary aim of NEEDS was to develop new and innovative ways of allowing users to search for datasets online. The NEEDS Extended Spatial Search (NESS) was developed to offer other spatial operations as alternatives to the spatial ‘contains’ method that is often used in geo-data portals. The ability to search beyond the boundaries of extents specified by the user was implemented because of the concept of spatial autocorrelation, that is the degree of influence exerted by something over its neighbours (Goodchild, 1986).

Whereas most web-based portals only test whether a layer lies within the coordinates specified by a user, this research extended the ArcSDE Java API to offer the ability to test whether a layer contained, was contained by, intersected, overlaid, was equal to or was disjoint from a spatial coverage specified by a user. The `SeLayer` class offered the `getExtent` method for returning the Minimum Bounding Rectangle (MBR) of an ArcSDE Layer. The minimum and maximum X/Y coordinates could then be used to create a Java `GeneralPath` object from which other operations such as contains, intersects or overlaps could then be carried out. The `GeneralPath` class was used

to abstract an OpenGIS(OGC) `multilinestring`. A `multilinestring` is the equivalent of an ESRI polyline with multiple parts. Additional OGC spatial operations were developed in Java and used to extend the tests that could be carried out on `GeneralPath` objects. Figure 2 shows some spatial operations that were made possible by extraction of MBRs through the `getExtent` method of the ArcSDE `SeLayer` class.

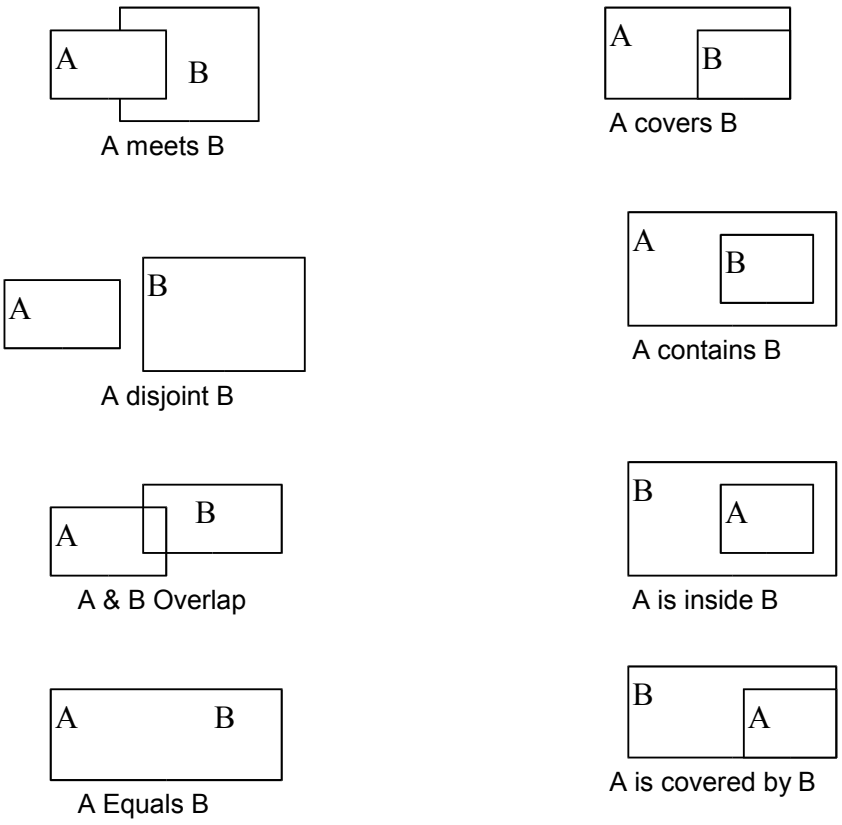

Figure 2. Spatial operations possible through extraction of the Minimum Bounding Rectangle using `SeLayer`'s `getExtent`

This section discussed the motivation and implementation of a facility for searching the NEEDS repository using different spatial operations. Having determined that a dataset exists, it is then important to provide a user with a preview of the dataset so that he or she may determine its relevance before purchase or download. The next section discusses a preview facility developed using the ArcSDE Java API.

3.0 The Preview Facility

With one of the aims of NEEDS being the ability to preview datasets in the context of a user's application, it was decided that the system would need to be able to transform ArcSDE layers into other file formats viewable from existing web-browsers. It must be noted that even though NEEDS also contains an ArcIMS 4 component, the highly varying requirements of potential users meant that an alternative interface had to be developed. This meant that the system had to allow the user to specify that a particular dataset should be previewed as a map, a mathematical chart, textual metadata or simply as ASCII text. Not only would the system need to create this preview but it would need to be able to create the preview quickly and efficiently. For example, the preview tools could allow users to preview a grid of points as a surface rather than as

a collection of dots. The variety in preview outputs was meant to cater for the different disciplines that makeup environmental research. Another challenge expected during development of the tools was that the preview tools had to be platform-independent to allow users to preview the datasets from different operating systems including Windows, Solaris, IRIX and so on. The tools also had to be able to export a dataset as an ASCII file that could be interpreted by any plug-in, for example a Virtual Reality Modelling Language (VRML) plug-in such as ParallelGraphics' Cortona.

The approach taken was based on mappings between vector geometry models. This is because different vector file formats use the same constructs of points, lines, polygons and attributes to describe geographic features. The next few sections describe how this facility was implemented.

3.1 Previewing As JPEG Images

A facility for previewing ArcSDE layers as JPEG images was developed for incorporation into NEEDS. This facility was mainly meant for users running legacy web-browsers without Java 1 or JavaScript capabilities. First a query was run using the `SeQuery` class. The query was provided with an `SeSQLConstruct` object that did not constrain the search (that is, there was no WHERE clause included in the SQL statement). A loop was then created to enumerate through the rows returned by the `SeQuery`. For each row, the `getShape` method was executed to return an `SeShape` object. From the `SeShape` object, a `GeneralPath` object was then obtained using the `toGeneralPath()` method. The `GeneralPath` objects were then passed to a Java `BufferedImage` object for rendering.

Having rendered the shapes onto a `BufferedImage` object, the next step was to convert the `BufferedImage` into a JPEG. The freely-available Java Advanced Imaging (JAI) API was used for this. The JPEG image encoder supplied with the JAI was used as illustrated in figure 3. The content type of the servlet's response was set to a JPEG MIME type. A Multipurpose Internet Mail Extension or MIME for short is a specification for formatting non-ASCII messages so that they can be sent over the Internet (Webopedia, 2004). Setting a JPEG MIME type would notify any clients receiving the servlet's response that they were to expect a JPEG. A common pitfall for developers is not to set the content type, resulting in clients expecting HTML text. This results in clients rendering the binary JPEG as text similar to when opening a JPEG with a text editor such as Notepad. After setting the content type, the encoded image was then written directly to the servlet's output stream.

```
//where "out" is the ServletOutputStream
//an "image" is the BufferedImage we are exporting as a JPEG

com.sun.image.codec.jpeg.JPEGImageEncoder encoder =
com.sun.image.codec.jpeg.JPEGCodec.createJPEGEncoder(out);

com.sun.image.codec.jpeg.JPEGEncodeParam param =
encoder.getDefaultJPEGEncodeParam(image);

param.setQuality(1.00f,false);
```

Figure 3. Use of the JPEG Image encoder

3.2 Previewing as VRML or GML models

To ensure interoperability it was then necessary to enable the export of layers in an open ASCII text-based standard. The two formats chosen for this were the Geography Markup Language (GML) and VRML formats. The former is an XML encoding for the transport and storage of geographic information, including both the geometry and properties of geographic features (OGC, 2003). The latter is a file format for describing interactive 3D objects and worlds (ISO, 1997).

To create the GML exporter, a servlet was created that wrote a GML `point` feature every time it encountered an ArcSDE `point` feature, a GML `multiLineString` every time it encountered an ArcSDE `polyline` and a GML `polygon` for every ArcSDE `polygon` encountered. These mappings were made easy to implement by the fact that ArcSDE layers are made of features of the same type. An extract from a GML file is shown in figure 4

```
<gml:LineString srsName='osgb:BNG'>
<gml:coordinates>
426518.790,567821.170 426516.330,567820.780
426509.060,567874.080 426509.880,567877.570
426510.000,567877.770 426515.660,567882.570
426516.200,567883.220 426516.980,567884.260
426517.440,567884.950 426525.470,567882.130
</gml:coordinates>
</gml:LineString>
```

Figure 4. An example of a GML Linestring

The VRML exporter was more challenging to implement than the GML exporter. This was because unlike GML which encodes content, VRML encodes both content and presentation. The aim of implementing a VRML exporter was to give users the ability to display a Digital Elevation Model (DEM) as a surface rather than as a grid of points. Such a facility would represent a DEM in the context of its application rather than its composite structures of points. A servlet was created that rendered triangular faces in-between sampled points of a grid. The resultant VRML construct was an `IndexedFaceSet` representing a DEM. An `IndexedFaceSet` is a collection of points that are indexed into triangular surfaces to describe a VRML object's geometry. The servlet's Internet address had to be set to have a *WRL* extension to enable Internet Explorer plug-ins such as Cortona to recognise the resource as a VRML file. The servlet's content type had to be set to a VRML MIME type to enable plug-ins to interpret the encodings appropriately. Figure 5 shows a DEM created by a servlet that reads an ArcSDE point layer then translates the data into VRML.

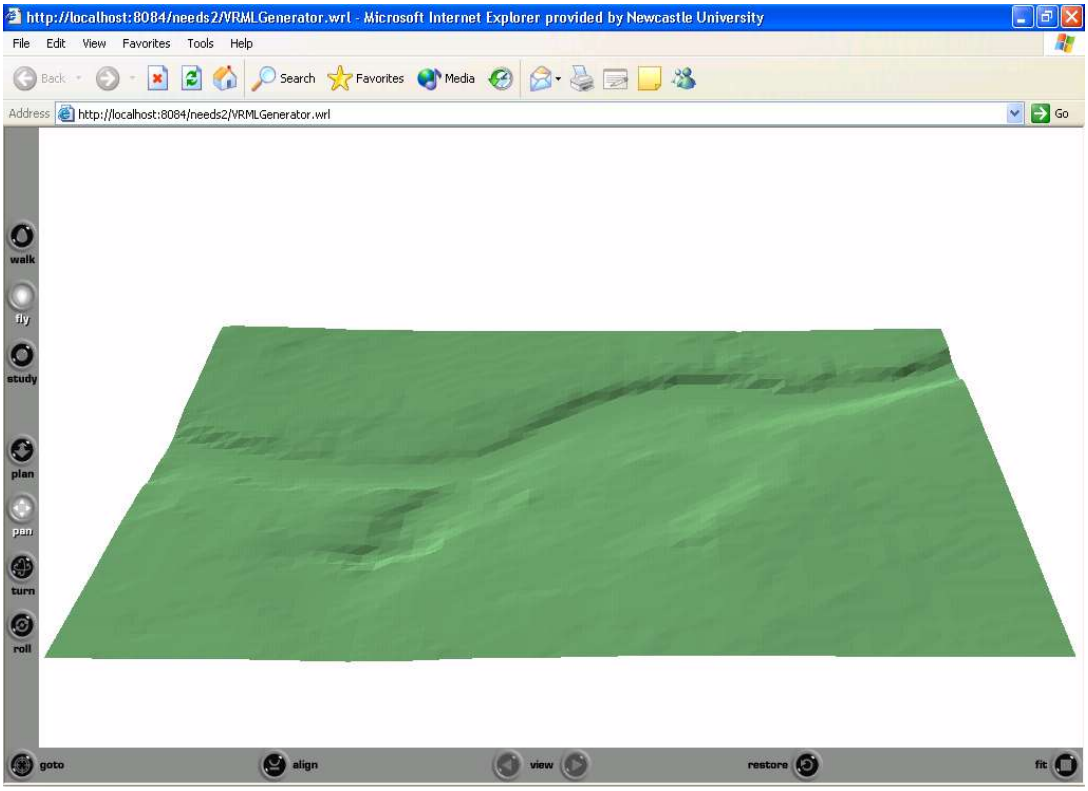

Figure 5. An ArcSDE layer exported to VRML by a servlet.

It must be noted that VRML is soon to be replaced by the relatively new eXtensible 3D (X3D) file format. X3D is a software standard for defining interactive web- and broadcast-based 3D content integrated with multimedia (Web3D Consortium, 2004). The specification is currently awaiting approval by the International Standards Organisation (ISO). Also, the approach to exporting ArcSDE layers can be extended to any ASCII text-based format including Scalable Vector Graphics (SVG).

3.3 Previewing On Java 1 Applets

One major issue in Java-based web development is that Internet Explorer (IE) is shipped with an old Java Runtime Environment (JRE). IE is able to interpret Java 1.1.8 whereas one needs to install a Java 2 JRE to execute any applets using the latest Java libraries. The ArcSDE Java API uses Java 2 libraries and hence it is also subject to this limitation. This means that using the API in applets where the client does not have Java 2 installed, leads to failure of the applet to execute. During this research, an approach to enabling ArcSDE layers to be viewed as objects in Java 1 applets was developed. The approach is based on the concept of 'casting' Java objects. Another concept used for this approach is the Java serialization facility that allows Java objects to be sent across a network as a stream of bytes. Furthermore, it is made possible by the apparent consistency in Java binary code during the migration from Java 1 to Java 2.

First, a geometry object model was developed using Java primitive data types. The reason for using primitive data types is that they are consistent from Java 1 to Java 2. Java classes have had additional fields and methods added to them during the course of Java's life. For example, the `setLockingKeyState` method has been available

in the `Toolkit` class since version 1.3 and not 1.1 . If Java classes are used, they must be compiled with a Java 1 compiler to ensure that they will be available on a client's Internet Explorer runtime environment, without a Java 2 plug-in. An example geometry object model is described in figure 6. Having created the object model into Java classes, next the ArcSDE Java API was used on the server-side, where Java 2 was available, to read a layer and create a collection of appropriate geometry objects. This collection was then sent across the internet to a client. As the same object model was available on the client, IE's runtime environment was able to read the byte stream, cast the received objects back into their original object model and then render the objects.

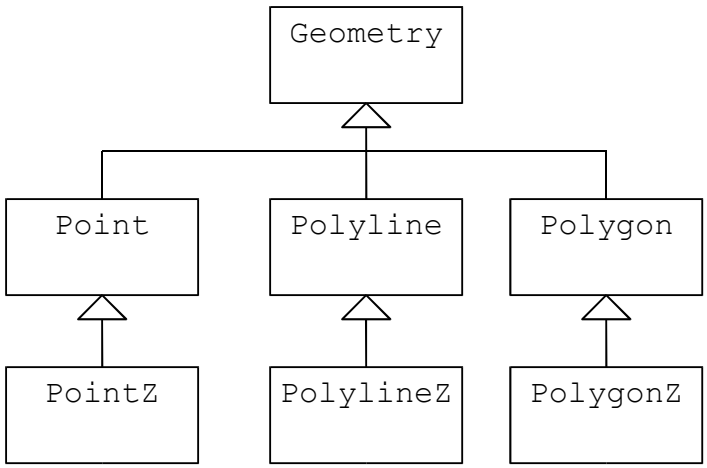

Figure 6. An example Geometry Object Model

Another Java 1 applet that read ASCII text was also developed and tested. This was done so as to determine whether or not sending GML across would be more efficient. It was found that sending Java objects across and casting them back into their object model took approximately half the time it required to send a GML file across and read by the applet. This was expected as GML is more verbose than serialised Java objects.

4.0 Conclusion

This paper has provided a description of tools that were created for cataloguing and previewing layers held on an ArcSDE data source. It has outlined some of the issues involved in the development of these tools. Possible pitfalls and security considerations are also suggested. The successful development of these cataloguing tools shows the great flexibility that the ArcSDE Java API offers to developers.

Section 3.3 outlined the motivation for developing Java classes that abstract the ArcSDE geometry model. It also discussed issues involved in deciding which Java compiler to use to compile the geometry model. The developers recommend that the ESRI geometry model be implemented in Java 1 to ensure compatibility with legacy Java applets and applications.

It has been shown on this paper that the ArcSDE Java API adds enormous value to the ArcSDE product. Consequently, the authors recommend that the API be further developed in future releases of ArcSDE.

Acknowledgements

The authors wish to thank the NEEDS development team at the University of Newcastle upon Tyne's School of Civil Engineering and Geosciences; Professor David Parker, Philip James, Simon Abele and Gobe Hobona. Java™ is a trademark of Sun Microsystems, Inc. MS SQL Server is a trademark of the Microsoft Corporation. Cortona is a trademark of ParallelGraphics, Inc.

References

1) **Hobona G., James P., Fairbairn D.** (2004), Facilitating Data Discovery In Environmental Data Clearinghouses Through Spatial Data Mining in *Proceedings of the GIS Research UK 2004 Conference*

2) **Open GIS Consortium (OGC).** (2003), *Geography Markup Language 3.0*, OpenGIS Consortium

3) **Hall M.** (2000), *Core Servlets and Java Server Pages Volume 1*, Sun Microsystems Press

4) **Goodchild, M**. (1986), Spatial Autocorrelation. Concepts and Techniques in *Modern Geography 47*. Geo Books, Norwich

5) **International Standards Organisation (ISO)** (1997), *Virtual Reality Modelling Language (VRML) in ISO/IEC 14772-1:1997*, The VRML Consortium, http://www.web3d.org/x3d/specifications/vrml/vrml97/vrml97specification.pdf, last visited 18/06/2004

6) **Crompvoets J., Bregt A.** (2003), *World Status of National Spatial Data Clearinghouse* http://www.urisa.org/Journal/APANo1/crompvoets.pdf , Last visited 24/06/2004

7) **Sun Microsystems** (1997), *JavaBeans Specification version 1.01*, http://java.sun.com/products/javabeans/docs/spec.html, Last visited 24/06/2004

8) **Webopedia** (2004), *MIME*, http://www.webopedia.com/TERM/M/MIME.html, Last visited 24/06/2004

9) **Web3D Consortium** (2004), *eXtensible 3D specification*, http://www.web3d.org, Last visited 24/06/2004

Author Information

Gobe Ernesto Hobona
PhD Student in Geomatics
School of Civil Engineering and Geosciences
University of Newcastle upon Tyne
NE1 7RU
United Kingdom
email: g.e.hobona@ncl.ac.uk

Simon Abele
PhD Student in Geomatics
School of Civil Engineering and Geosciences
University of Newcastle upon Tyne
NE1 7RU
United Kingdom
email: s.j.abele@ncl.ac.uk

Philip James
Lecturer in Geographic Information Science
School of Civil Engineering and Geosciences
University of Newcastle upon Tyne
NE1 7RU
United Kingdom
email: philip.james@ncl.ac.uk