

XML Information Services in Archaeological Excavation

Matthew Reed
University of Kent
mr5@kent.ac.uk

Abstract

The increasing economic viability of mobile devices has led to a new breed of computational tools that can be applied to existing problems. Archaeological excavation is one application area to which mobile devices are being used and a range of tools that aid in this domain are being developed. However as archaeologists use a variety of these tools in recording data there becomes the need to store this data that is represented in numerous formats. In this paper an information store is proposed that is able to manage the multitude of data recorded at excavation in an XML format. The proposed store, xmldig also provides collaboration facilities so that data can be transformed into the format of any tool registered with the store.

1. Introduction

Recent work into mobile collaborative tools has led to the creation of the Jnet [1] tool, a general purpose graph browser specialising in stratigraphic sequencing [10]. The Jnet tool itself extended concepts of previous tools that focused on stratigraphic sequencing and browsing of graph data structures; implementing them into a tool that utilised some of the latest viable technology developments available. Most notably Jnet introduced mobile stratigraphic sequencing for use in mobile devices; giving it the capability of recording sequences whilst in the field excavating, this development was made available through use of wireless networked environments, document exchange using XML [2] and JDBC[3] databases, and developments in the Java language allowing deployment on a range of platforms and environments.

The design of Jnet and its collaborative nature gave rise for the need of a centralised data store that could manage the data inputted and requested by multiple users and output Jnet data to applications other than Jnet, such as a web browser. With this in mind this paper will concentrate on the provision of such a system; discussing a prototype design and implementation. Providing such a centralised

service to Jnet requires on the basic level a need of some form of storage, a database, and communication between the database and the application. However there is a need to greatly extend this service to provide alternative access to other applications so that data can be visualised and analysed by different tools. Also there is the need to abstract the level of detail provided to a user depending upon their knowledge. Therefore a providing service should not simply be a backend system for the Jnet application but provide data in a format that is openly accessible to a variety of relevant tools, as well as this the service should be able to manage the varying uses of Jnet as a general purpose graph browser.

Providing data access to multiple tools requires the use of an openly accessible data format that is widely supported, it is here that Jnet's use of XML document exchange can be employed. The use of XML opens many possible development options in terms of data accessibility, such as XML data access and visualisations (SVG[4]) through web browsers, and document transformations (XSLT[5]). Manipulations aside the one of the main obstacles resides in the storage of our data. Established thinking dictates the use of a relational model [16] database that holds our data in various tables being accessed and updated using SQL[21] queries. Although this approach is extremely mature and proven the relational model requires that XML data be transformed in conformance with a specific relational schema; known as shedding. This may be useful for validated XML documents that need to maintain strictly to a schema definition, however one may require storage of schema-less documents within the database which may extend schema or not fully implement schema, and secondly there is the cost associated with validating and transforming a series of relational tables into XML and back again every time document access is required. A remedy to this issue has been found however in the recent development of XML databases, these databases manage collections of XML documents as persistent XML using various database specific logical model representations. To query XML databases most implement the increasingly popular XPath [6], a language for addressing parts of XML documents, and XQuery [7] a

language for providing flexible query facilities. One such accepted XML database that implements the XQuery standard is the open source eXist project [8]. In using the eXist database a number of options present themselves as how to integrate application access, much of these dependent on choices of service containment and delivery discussed in later sections.

2. Aims

The overall goal of the paper is the discussion of a design and implementation of a central data management service for the Jnet tool to manage collaborative working in mobile environments. In doing this there must be full understanding of a number of different application areas.

Archaeological analysis and recording tools show the format data takes and how it is abstracted. It also gives an insight as to how this data is best visualised, what presentation is most pertinent to the excavator and to other audiences that the data may be prepared for.

XML document database management systems studies allow the specification of management features such as efficient implicitly ordered document collections, access control at varying levels of granularity, and resourceful use of querying and updating data using XQuery and Xupdate[19].

Considering the choice of the eXist database research must be completed into the communication methods which dictate the way the service will be contained and the methods in which data will be delivered to client applications. Designs available for integrating the eXist database vary from a entire web based application completed entirely in XQuery to Java applications using a common implementation of the XML:DB API[20].

3. Background

3.2. Stratigraphy and the Harris Matrix

The general term stratigraphy refers to a branch of geology concerned with the study of rock layers and layering. Within the context of archaeology; stratigraphy is used in the course of excavation where artefacts are known to be located beneath the surface of the ground. In this case an excavator will attempt to model the stratigraphic sequence by recording soil, rock layers and other features. Features may include

for example a 'cut' within the sequence such as a ditch or man-made trench; the material deposited in the cut is known as a 'fill'. In recording the stratigraphic sequence one must follow the law of superimposition, this is a simple rule stating that subsequently discovered deposits that are more deeply buried are older than that of the previous ones. In most cases this holds true and allows excavators to relatively date all the layers and features they find, however the exception to this rule becomes apparent when multiple cuts can effectively place a younger layer below an older one; and careful examination is required to extract the correct sequence.

Excavators record the stratigraphic sequence in a common form known as the Harris Matrix [10]. The Harris Matrix, named after its creator Dr. Edward C. Harris is mathematically a simple directed graph, with nodes representing the layer or feature and connecting edges representing the physical connection in the form of a relative age relationship. This recording method provided a way to view the sequence in a diagrammatic form and the *de facto* method for stratigraphic recording. The form of the diagram is essentially that of an orthogonal graph layout (see Figure 1) resulting in a very clear, concise display. Nodes are identified by a context identification label and orthogonal edges protruding from the top or bottom of the node denote an older or younger relationship respectively, edges protruding from either side of the node represent a contemporary relationship.

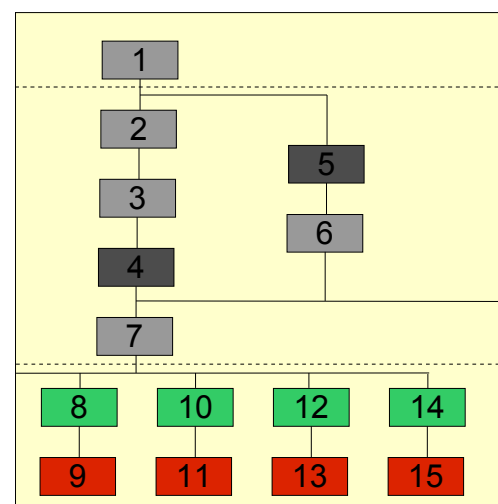


Figure 1: Harris Matrix graph layout extract

3.2. Computing in archaeological excavation

Initial computing uses in archaeological analysis began over 30 years ago in the 1970's

and even though technologies have changed drastically over time the basic aims of these tools remain fairly consistent as the requirements of archaeologists has stayed the same. Early examples of archaeological stratification tools were limited by the technology of the day which made appropriate use questionable, with some notable archaeologists advocating against the use of computers in excavation recording and post excavation analysis [9]. The argument being posed that the quality of excavation recording cannot simply be achieved by using computational tools but that it was dependent upon the excavator that is observing. This is still very much an issue today as computer input interfaces can skew the abstraction of the information to be recorded, for the intention of keeping data models consistent.

Advances in other fields of computing especially Computer Aided Design (CAD) and databases allowed a new breed of stratigraphic analysis tools to be built. One of the first of these tools was the HindSite application [11]. This tool utilised CAD and databases to provide recording of deposits and stratigraphic relationships between, this can then be visualised as both a standard Harris matrix and a three dimensional model. This visualisation also gave the ability to phase stratigraphic units into coloured groups, a concept of great importance in tools today.

In a similar vein to the HindSite application came the ArchEd tool for visualising Harris matrices [12]. This tool offered basic visualisation functions for navigation around and interaction with a Harris Matrix, and also a useful automated layout mechanism. However one of the most appealing features of the ArchEd tool was its import and export facilities, matrices could be exported into an enhanced Metafile format containing vector and bitmap-format graphical data, these functions allowed ArchEd to be used in conjunction with other archaeological tools, specifically the Bonner WINBASP application [13].

Following on from these applications and the actual predecessor of the Jnet tool came gnet. The gnet tool included all the basic facilities included in previous similar archaeological tools such as visualisation and document format interchange but was developed as a more general graph browser that suited the directed graph nature of the Harris matrix. The greater robustness offered by gnet inherited from its predecessor gtree, a similar tool used for browsing tree like data structures. There

were issues associated with gnet however as development continued the software relied upon advanced hardware that put usage out of the reach of some of gnet's audience, also developments in gnet's operating platform, Microsoft Windows lead to incompatibilities; eventually leading to its discontinued development.

As new technologies became increasing economically available to archaeological excavators there became new ways to apply them. One specific advancement came in the field of mobile computing which today has become extremely prominent. With computers not tied to fixed locations archaeological excavators could take recording equipment to sites and document data directly. In this area not only stratigraphy can be recorded using mobile devices but much other data as well, data such as digital photographs, tachymeter readings, voice recordings, hand recorded diagrams and automated location awareness, such systems for recording are discussed in [14, 15].

3.3. Jnet and XML file formats

The initial prototype for Jnet is discussed in [1]; recently however a first working implementation has been produced (see figure 2). Its design has changed little from the discussed prototype although at its early current implementation stage there are some additional functional offerings that Jnet provides. As well as all the abilities provided by its predecessor gnet such as basic Harris matrix visualisation, stratigraphic unit grouping and layout functions; Jnet also offers full three dimension graph modelling where node, or stratigraphic unit dimensions are provided. Also present are some automated grouping functions such as articulated sub graphs, and applicable to stratigraphic graphs only; functions for feature grouping.

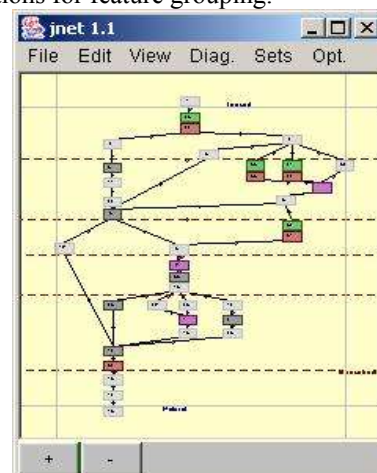


Figure 2: Jnet, main window

There are variety of connection setups designed for Jnet, the most basic and simple connection model requires no connectivity at all as all required data is stored locally within Jnet's operating environment in the form of JDBC access to a database or Jnet XML files stored in a directory. The second model is that of a client server; with connections through HTTP requests, this allows Jnet to centralise data storage allowing it to become a tool of collaboration with other Jnet units, again data can be stored in a JDBC database or simply as XML. The final connection model has yet to be implemented but represents an interesting way providing the Jnet tool to numerous users, here Jnet is run from within a servlet container and accessed through a web browser, with each user having an instance of Jnet at their disposal, this setup allows centralisation for data storage and simply requires the user device to have a web browser instead of installing the Jnet application.

The Jnet XML file format in which all graph documents are communicated follow a conceptually simple schema that is divided into two distinct parts, the graph type, and node and edges. The graph type defines the specific type of graph that the document is, giving type description and name. Next there is a definition of nodetypes and edgetypes, these types describe to Jnet how nodes and edges of a particular type should be displayed and any special properties they have; they are written in an SVG style syntax (see figure 3). The node and edge part of the document describes the actual parts of the graph, each node and edge has identification and for nodes some form of spatial information associated with it, edges however contain the identification of which nodes they come from and go to (their direction), edges may also hold weighting information. Each node and edge also has a parent type from which they inherit their display characteristics, nodes and edges may also describe additional characteristics they wish to display in the same SVG style as edgetypes and nodetypes. Also node three dimensional display data maybe presented here in the form of an SVG path.

```

<node id="34" type="wall" x="687" y="421">
  <objectshape>
    <path id="34" d="M 1083 2743 L 303 2739 l
  </objectshape>
</node>

<edge type="over" from="28" to="14"/>
<edge type="over" from="29" to="14"/>

```

Figure 3: Jnet file format extraction

3.4. XML databases

Unlike traditional relational databases, XML databases are fairly new in their existence, systems available vary greatly and until very recently many were not compatible unlike their relational counterparts which conform to regulated standards in their storage models, access and query methods. XML databases can be divided into two types known as XML enabled databases and native XML databases. XML enabled databases use a well established storage models such as the relational one, from which XML data is shredded or flattened into, upon retrieval data in this model is used to build the required XML document on the fly. This system approach ensures that XML data conforms to storage schemas and maintains validity, however where storage of non validated data is required such storage models will store documents simply as a CLOB (Character Large Object) or another proprietary data object. This is useful for complete document handling although where partial documents fragments are required entire documents need to be parsed to prepare a fragment. It is in this area that native XML databases pave their niche. Native XML systems describe their data model in a data-centric fashion with each XML document broken down into a tree like data structures, specific XML data models are discussed further in [18].

The eXist project[8] is an open source attempt at implementing a native XML database system that can be used in multitude of ways from web-based systems to stand alone applications. The system is written entirely in Java and can be configured in a variety of ways such as Java servlet containment or directly embedding system into software.

eXist is unlike many other existing native XML database implementations in that it is designed specifically to provide optimised index-based query processing; by indexing all nodes that constitute an XML resource loading the actual node content is not required until query results are displayed. This is implemented using four index files that form the core of the storage backend: collections.dbx manages all the collection hierarchy; collections are the containers to which any XML resource must be held. dom.dbx is a paged file of all nodes and their associated identifiers to the actual node data. elements.dbx provides the index for all elements and attributes and words.dbx provides an extension for full text searching; keeping track of word occurrences. All indexes described are based on B+-trees and are

organised by collection and not by inserted document, this means that a number of documents are represented by a single collection; this enables users to query entire sets of common documents efficiently. For more on the internal functionality of eXist see [27].

Due to the numerous setup configurations available to eXist there are number of ways to connect and interact with the database. The simplest and fastest way to access the database is provided through a REST-style web API. This method only requires HTTP client however the functionality that it provides is fairly limited, although it can run both in a servlet and stand alone context.

To allow eXist to be accessed from other applications such as CGI scripts, PHP and JSP eXist uses the Apache XML-RPC library. This library allows common procedure calls to be sent to eXist as an XML document. For more information on XML-RPC see[28].

Provided as a more convenient alternative to XML-RPC is a SOAP interface, this however is available only from within the servlet configuration. The original aspect that SOAP brings is that from a specified WSDL service description connecting SOAP tools will automatically generate the low level code that is required, this simplifies much of what has to be hand written by XML-RPC; although SOAP tools are currently fairly large and complex. SOAP is discussed in greater detail in section 3.5.

For use in web applications eXist enables entire applications to be written completely in XQuery scripts implementing the entire processing logic, these can then be passed to a Java servlet that processes and outputs the results to the requesting web browser.

The favourable way to access eXist through Java applications through using the implemented XML:DB API[20]. The XML:DB API is a vendor neutral API specifically built for XML databases, it is designed to be implemented in a variety of programming languages although currently the only release is a package of Java interfaces. The API is divided into four main concepts that allow applications to connect, address, update and query XML databases that implement it. Firstly databases must provide a specific driver that envelops it's access logic, this concept is similar to that of JDBC and ODBC. Secondly native XML databases introduce the notion of collections, collections

are the containers in which XML data is stored and find themselves roughly equivalent to the relational concept of a table, all databases must include at least one collection from which all sub collections reside. Thirdly is the concept of services, these provide the API with the majority its interaction capabilities with XML data. Each service is defined to enable a certain set of functions to be used on the database, for example the XPathQueryService and the XUpdateQueryService allow the respective languages to be used to query and update the database that implements the API. Other services are described that implement maintenance functions such as access control and collection editing. The final concept is that of the resource. The resource is a generic abstraction of data that provides a common way to express the content stored in the database, these can be specialised however so databases can include a variety of data types for example binary data.

3.4. All things X

Forming the basis of addressing any XML based resource is the XPath[6] XML Path Language. Originally the result of an effort to find a common syntax and semantics that are shared by XSLT and XPointer[26], XPath is a compact syntax that is not XML but similar to URI path notation (hence the name), included within the language are also some simple functions that allow basic manipulation of strings, numbers and booleans. Rather than navigating its character based syntax XPath operates on the node-tree logical structure allowing any part of the document to be referenced by its type or value.

XML database systems can offer a variety querying mechanisms generally in the form of extensions to previously established ones such as XQL[22], XML-QL[23] and Quilt[24]. Recent attempts to standardise XML querying however have resulted in the W3C working draft of XQuery[7] a predecessor of XQL and specific XML querying language. XQuery derives itself from Quilt, also it borrows aspects from previous XML querying mechanisms and more well established ones including SQL[21] and OQL[25]. XQuery is essentially three languages combined, a surface syntax that most will encounter, an XML-based alternative syntax to this which has better processing performance and a formal algebraic language that describes an XQuery in much greater detail. The familiar language most use have the properties of a strongly typed functional language, this is based upon expressions using keywords, symbols, and

operands, also using XPath for resource addressing. Also there is support for FLWOR (For, Let, Where, Order, Return) expressions for iterations and intermediate result binding. With this ability XQuery can be embedded into XML documents prepared for processing either for querying of XML documents or databases directly or through middleware connected to the data source. Also an XQuery may also be directly inserted and stored within a database, this allows the query to be directly executed internally; without compilation.

XQuery and XPath present themselves as extremely useful mechanisms to query and address XML resources respectively. However these languages provide no functionality to update and remove XML resources and so therefore a major gap is left in ones interaction methods, as a solution the XUpdate[19] XML Update Language can be used. XUpdate is an XML syntax based language that specifies what changes to be made to a target XML resource. The XUpdate proposed standard is neither a W3C Recommendation nor an ISO or IETF standard, having not progressed passed it's September 2000 Working Draft. Due to this it has not been very well specified however it's highly convenient; simple functionality have spawned a number of implementations. The eXist project implementation allows for the very useful option of inserting XUpdate fragments into an XQuery script.

3.4 Service containment

Given the robustness of the eXist project we are given many options on how contain eXist and the proposed application. As it is written in Java eXist itself must be contained either within another Java application or as a web application in a servlet container. The latter option in this case provides the maximum functionality of eXist allowing all connection methods to be used, also this gives the ability to access and edit all content through a web browser. With this in mind it also seems logical to create the proposed application within a servlet container also. With this setup the application can be inserted into eXist running in the same context utilising all its connection abilities, also it could be implemented within a separate context without the need for much alteration.

The use of Java servlets is a cornerstone of web based applications. Essentially they are Java programs that run on web servers and are accessed through the containing web server's URI by HTTP based program clients. In using Java one can obtain all the benefits that come

with it such as portability; not only across operating platforms but also across server implementations, as well as this servlets are able to access the full power of the Java APIs; both core and those that reside in the J2EE platform; some specific for use with servlets. The servlet containers that hold servlets are basically specialised web servers. Apache's Tomcat server is the official reference implementation for how containers should support servlets. This container like many others can operate as a stand alone application itself or be embedded into an existing web server such as that of the Apache HTTP server.

3.5. SOAP

SOAP[29] (Simple Object Access Protocol) is an XML based messaging protocol designed for exchanging structured information in distributed environments. Constructed in XML; messages are able to be exchanged over a number of underlying protocols most commonly HTTP, and may also include data that is not XML format but can be described by MIME type, because of these reasons SOAP is able to be platform and language independent.

A SOAP message is a standard XML document constructed from a number of specific elements. The primary element is the 'Envelope' element, this the root element of a SOAP message and defines the document as a SOAP message. Within the envelope come up to three child elements that hold the XML content of the message. Firstly an optional 'Header' element can be present, the header element contains application specific information; authentication for example. Secondly comes the mandatory 'Body' element, within this the actual message XML content that is intended to be used resides. Finally an optional 'Fault' element can be included to hold error and application status information relevant to the content to the SOAP message, the fault element must be included as a child of the body element.

Beyond the XML content of a SOAP message are SOAP attachments. Attachments can be in any format and are specifically designed to enable the transmission of resources from images such as JPEG to binary files. To facilitate this the entire message with it's attachments is put into a multi-part MIME structure for transportation.

Alone SOAP represents a simple way to invoke communication between two agents, however this functionality can be extended greatly when coupled with other XML based technologies

contributing to what is known as the web service architecture[30,31]. As a generality web services are complex distributed pieces of software built to bring clients what they need, when they need it, on whatever device they chose, although not currently entirely true it serves as a good description of the situation where one would like to arrive at. Within this architecture SOAP represents the communication mechanism for which all components can distribute their content. Currently the other major XML technologies included are WSDL (Web Services Description Language) used to describe the functionality that a service offers, and UDDI (Universal Description, Discovery and Integration) that allows services to discover other services and specific service to 'advertise' itself.

4. The xmldig tool

4.1. An Overview

The xmldig tool represents a prototype implementation of the proposed store outlined and researched in earlier sections. Conceptually it comprises of three distinct functions that provide users with file and collection visualisations, and also client specific file format transformations. For the implementation of xmldig the Jnet file format has been the test base, providing a registered application for which to transform files to.

When building the tool key properties that the software must have are taken into consideration. For xmldig key to its design are extensibility and robustness. As the software is a prototype it represents only the start of development, therefore provision for its continued development must be an important factor and one that is considered in this case. Also a robust piece of software allows one to look at a variety of way with which to extend and improve it through considering other avenues left open by the original implementation.

The xmldig tool makes much use of the eXist project native XML database [version 1.0beta2 build 1107] and for the design and implementation this database is organised for a development set-up. This organisation requires the database to be configured as a web application in a servlet container, with this all connection methods are available and web based status tools can be used to monitor the database collections and its condition.

Much of the design makes great use of servlet containers, as does this eXist. For deployment of these services the Tomcat servlet container [version 5.5.7] has been used, this container is a well proven piece of software for use in both application development and deployment. The xmldig tool and the eXist database will reside in the same container although running within different contexts.

4.2. Design

As stated previously the design is based on three concepts, the main and most important one is that of file transformations, then that of collection visualisation then file visualisation. The three concepts translate logically into three different servlets, each having one of the respective tasks to do. The servlets will run in the same context within the container as they share resources between them also this provides access to them at a similar URI, that of: `http://containeraddress/xmldig/servlet-mapping`.

Essentially the three servlets provide the processing logic to transform client requests in commands on the eXist database either updating the database or accessing content and building a reply to send back to the client. Within eXist three collections reside that provide components of xmldig to access to build documents for the clients that use the tool. Figure 4 below, shows the internal collection structure of the eXist XML database for use with the xmldig tool.

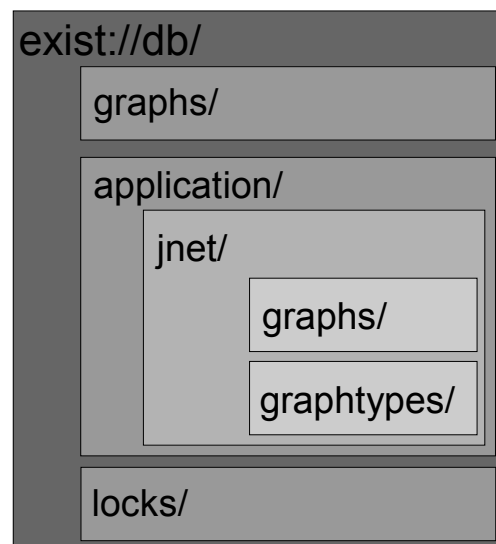


Figure 4: eXist internal collection structure for xmldig

The main collection is the generic graph store at collection 'graphs/'. This collection holds all the data that can be accessed within a single

generic format based on the notion of graphs and nodes (edges are stored as attributes of nodes), to this effect it does conform to an implicit schema as the 'graphs/' collection is a set of graph elements each one containing a set of child node elements, however this is as far as it goes in terms of schema. All children of the node element are elements that are shared across client applications, this allows any client using xmldig to access this information in the format required by the application (see figure 5 below).

```
<graph id="ap2004" format="stratigraphy">
  <description>The ap2004 excavation</description>
  <node id="1">
    <type>h-interface</type>
    <dateOfRegistration day="05" month="08" year="2003"/>
    <nameRegistrator>peter</nameRegistrator>
    <sector>trench 1</sector>
    <interpretation>walking level</interpretation>
    <relatedLayers>
      <under>2</under>
      <under>3</under>
      <under>4</under>
    </relatedLayers>
  </node>
  <node id="2">
    <type>layer</type>
    <dateOfRegistration day="05" month="08" year="2003"/>
```

Figure 5: an extract of XML from the graph/collection

Although the generic collection holds shared data, to correctly and accurately build specific XML file formats conforming to schema may require a little more information. This information is client application specific therefore each client or file format that is being delivered by xmldig has its own collection within the 'application/' collection so that it can outline the extra information to be included. In the case of Jnet extra visualisation information is required and these are described in two sub collections of the 'jnet/' collection that of 'graph/' and that of graphtypes, 'graphtypes/'. The graphtypes collection contains different types of graphs that dictate the visual and semantic properties that a node can inherit, all nodes in a graph of a certain type must inherit a specific nodetype contained within the graphtype. Within the graphs collection are stored positional data that the node has, also a distinct node can override the nodetype information that it has inherited and define its own here.

The third collection is the 'locks/' collection. This collection implements the store for xmldig's varying granule locking. Due to the mobile nature of clients using xmldig the distributed environment may not allow for constant updates, network connectivity maybe low out in the field and because of this xmldig uses a three tier locking granularity. The

locking granules cover the notion of graph, node, and none. A graph lock locks an entire graph and its constituent nodes from being accessed by any client until that graph resource is updated. A node lock locks only a particular node and its children within a graph allowing clients to access the graph and update other nodes but not the one currently locked, the lock held is released when the node is updated. The final locking mechanism basically is not one, it implements a no locking policy where no locks are ever written.

Communication with eXist is carried out with the XML:DB API, this provides xmldig all the functionality required to access and update the database. When used as a remote connection as the case is here, with xmldig and eXist running in different contexts the XML:DB API implementation accesses the database using XML-RPC calls to eXist's XML-RPC server. If eXist were embedded the API would connect directly to the database.

With knowledge of the backend storage and connectivity figure 6 provides a diagram of the servlet design that constitutes the processing logic of xmldig.

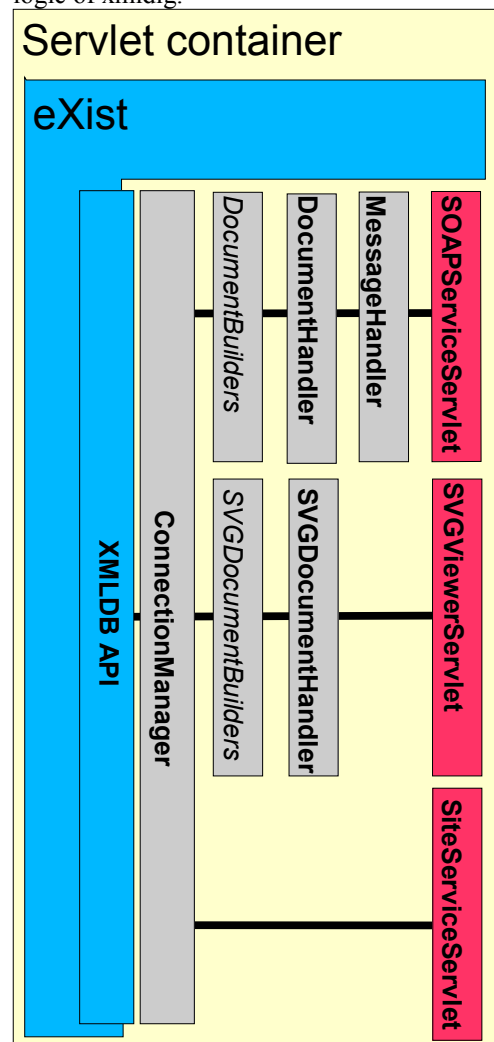


Figure 6: xmldig design diagram

The largest and providing the most prominent servlet of the tool is the SOAPServiceServlet. This servlet is the communication mechanism through which xmldig client applications send data to and receive from. The data that the servlet sends and receives is in the form of SOAP messages, a client application sends a request to access or update a resource and the xmldig tool responds in some way to this request; again with a SOAP message. The format of the request messages used to communicate with xmldig is discussed in greater detail in the next section.

As a message comes into the SOAPServiceServlet it is passed to the MessageHandler where the SOAP message is de-constructed and the request command extracted. The MessageHandler then passes the command and request information to the DocumentHandler. The DocumentHandler deals with the calling the individual database transactions that are required that correctly lock and obtain the appropriate resources, locks are acquired and released directly from the ConnectionManager, building or updating the database is delegated to DocumentBuilders. DocumentBuilders are required for each client application to provide the correct query mechanisms to pass to the ConnectionManager which interacts directly with the XML:DB API.

The design of the other two servlets is much simpler as they only support read-only operations to the database. The SiteServiceServlet is built to provide navigational support around the database displaying graph collections that can be viewed in an SVG format and the current locks are held within the locks collection. The main aim of this design is to provide a web based support portal for the tool, with the intention of extending it to include other support roles.

The final servlet is the SVGViewerServlet. This servlet simply returns an SVG file of the given resource and file format included as a parameter at the servlets URI. The servlet passes the resource identifier an SVGDocumentHandler, this will then invoke the correct implementing SVGDocumentBuilder, one of which exists for each client application. The document builder will then reply with an SVG file to the viewer servlet which will provide an SVG browser it's file visualisation.

4.3. Implementation

The implementation of the tool has followed much to the design laid out previously. The two minor servlets are implemented by extending the `java.servlet.http.HttpServlet` and beyond this simply apply some of the core Java APIs.

The SVGDocumentBuilder is built as an interface with each client application that requires its use implementing this, for example JnetSVGDocummentBuilder provides the logic to actually build the SVG file by querying different parts of the database and pulling them together. For this purpose the implementing document builders use the ConnectionManager class, and although it does not apply the majority of the manager's functionality the inclusion of the connection manager gives this area extra functionality should it be needed later on.

The SOAPServiceServlet is a little different from the other two in that it implements the SAAJServlet, part of the SOAP with Attachments API for Java (SAAJ), this servlet is especially designed to receive SOAP messages and it does just that. Essentially the SAAJ API gives access to the `javax.xml.soap` package that gives object representations of the SOAP message structure as well as methods to modify and access specific parts of the message. The SAAJ API is extensively used in the MessageHandler class providing a way to tailor each reply, this also allows for the use of other parts of the SOAP message should the message structure need to become more complex. The MessageHandler class's main responsibility is the identifying the right command requested and calling the associated method in the DocumentHandler class.

Commands to the SOAP servlet are of five types that compromise the root element of the SOAP body request, these commands are: request, create, update, insert and delete. Following a request, create or delete command come a series of node or graph identifiers, this will tell the command what resource to act upon. The create and delete commands are fairly simple in that they will either create the base tags required for node and graph creation or delete them and all the children that they have. The request command is passed to a client format specific DocumentBuilder (the type of which is included in the request message), the specific document builder can then prepare the required graph file or node properties file to be sent back. Update and

insert commands are a little more complex, again after each command element a come a series of node or graph identifiers, however in this case the actual update and insert information is found as children of the identifiers. With this editing information the DocumentHandler will pass it to the appropriate DocumentBuilder which will break it down and store it in either the shared graph store or if client file format specific, in it's desired collection. A more detailed guide to the transaction mechanism is included with the software.

4.4. Beyond the prototype

The current prototype provides the all the basic functionality that is required of it but in implementing the tool there have been certain areas identified that require extension to improve the quality of the software an bring it beyond the prototype phase.

The locking mechanism that xmldig provides is useful for the environment that clients operate in. Differing locking granules allow ensured consistency depending on the amount of collaboration involved. To extend the collaborative idea within locking xmldig could implement a more 'intelligent' locking mechanism. Currently when a resource is requested and it is locked xmldig simply replies with a notification the the resource is currently locked and informs the client of the locking policy currently enforced. Included in this xmldig could also add who currently holds the lock in terms of a user or device so that a client could better understand whom is working on what resource. Implementing this would however require registered knowledge of users or devices so that clients can identify them and initiate a request to the lock holder to release.

Currently for xmldig to carry out a command a number of different transactions to the eXist backend must be carried out, this is due to the locking phase, the resource gathering phase and then possibly an unlocking phase. If at any of these phases eXist were to fault and crash the database may be left in an inconsistent state. To try and remedy this xmldig could combine all transactions into a single one. If eXist were to fall it would be able to be able to use its recovery mechanisms to rollback the single transaction, which although not applying the changes required leaves the database in a consistent state. There are two ways to implement this single transaction model, either combine the XUpdate script for locking within the XQuery script; used to gather the resource. Or alternatively build a commit/rollback

mechanism within the connection manager, whereby all transaction requests are buffered until a call is made to combine and execute them within a single transaction.

The use of SOAP in xmldig is limited to the content of SOAP body's, however within a SOAP envelope are many more useful features. Utilising SOAP headers enables the inclusion of authentication and validation information, allowing xmldig to restrict access by using passwords and user access permissions within the database, this is not much of consideration at this stage however its one that may become more important if deployed. Also SOAP faults can enable a standardised fault reporting mechanism to all clients, providing reporting for not exception states but also general failure information, a locked resource for example.

Making the information as accessible as possible in the store is key to it's success and by altering the generic store format to something that is more common and established would enable this. GraphML[32] is an XML based format especially for graph structures, it is well established and represents a standard that almost any graph format can be built from. By implementing a standardised graph format such as GraphML the store could be much more accessible to other applications and transformations to other formats would require less processing.

5. Conclusions

As a simple prototype the xmldig tool provides a basic concept from which to construct a store to support a multitude of file formats. Its use of newer XML based technologies has allowed much of the document processing to be carried out within the backend storage eXist database. The support eXist provides also enables applications to utilise it in a number of ways, it's support for the new XQuery standard and the XUpdate language provides a huge amount of robust functionality, and it is now becoming a viable option to construct entire web applications from XQuery and its related technologies. Also eXist now currently supports the storage of binary collections and not just XML, with this in mind eXist can now combine all recorded resources within a single database.

The deployment of xmldig could take a number of forms, used in excavation for example the tool could be run locally on the excavation site from a PC with wireless capabilities, this would enable wireless mobile devices to be used by observers as they excavate. In fact as

long a network connection exists on site or be provided on a site the xmlDIG tool can be accessed and utilised from anywhere.

Currently the act of excavation recording is done on paper forms which are entered into a computer away from the site and possibly post excavation, this method is well established however it is time consuming and prone to recording errors; which when realised could be too late to be rectified as deposits are destroyed. Bringing mobile devices in at the excavation stage offers a solution to these problems and conceptually could be implemented very easily. In reality the physical environment of an excavation site is not one that many mobile devices could tolerate, exposed to wind and rain as well as possible extremes of temperature; few of today's devices would stand up to the task let alone cope with data entry after a user has spent all morning digging in a muddy trench. However with the increasing development of low cost, highly connected mobile devices; one feels it is not long before devices are created that can stand up to the harsh operating environment and capably run the software required.

The increasing use of computation tools not only in archaeological excavation but other field surveying activities requires the use of an organised store for all the numerous types of data that are recorded. xmlDIG provides a common organisation for XML based files, however many recorded formats are in a proprietary form that requires specific knowledge of that format for a tool to extract information. This may be an issue in some cases although increasingly data is now prepared in XML with maximum availability in mind, even in instances such as media formats where XML is used as metadata for the format.

6. Acknowledgements

Special thanks to Dr. Nick Ryan for his guidance and proposing the project in the first place. And also to Alia, for listening to me ranting about computers all the time.

7. Bibliography

[1] N. Ryan, 'jnet: a successor to gnet', In W. Borner, editor, *Archaeologie und Computer, Workshop 6*, Forschungsgesellschaft Wiener Stadtarchaologie, November 2001

[2] The World Wide Web Consortium, Extensible Markup Language (XML), <http://www.w3.org/XML/>

[3] Sun Microsystems, JDBC Technology. <http://java.sun.com/products/jdbc/>

[4] The World Wide Web Consortium, Scalable Vector Graphics (SVG) XML Graphics for the Web, <http://www.w3.org/Graphics/SVG/>

[5] The World Wide Web Consortium, XSL Transformations (XSLT) Version 1.0, <http://www.w3.org/TR/xslt/>

[6] The World Wide Web Consortium, XML Path Language (XPath) Version 1.0, <http://www.w3.org/TR/xpath/>

[7] The World Wide Web Consortium, XQuery 1.0: An XML Query Language, <http://www.w3.org/TR/xquery/>

[8] eXist, Open Source Native XML Database, <http://exist.sourceforge.net/>

[9] E.C. Harris, 'Stratigraphic Analysis and the Computer', *Computer Applications in Archaeology*, 1975, pp. 33–40

[10] E.C. Harris, 'Principles of Archaeological Stratigraphy', *Academic Press London*, 1989

[11] B. Alvey, 'Hindsight', *Archaeological Computing Newsletter*, 19, 1989, pp. 4–5.

[12] P. Mutzel, B. Reitgruber, and B. Schuhmacher, this volume, 'ArchEd: an Interactive Tool for Visualizing Harris Matrices', In W. Borner, editor, *Archaeologie und Computer, Workshop 6*, Forschungsgesellschaft Wiener Stadtarchaologie, November 2001

[13] I. Scoller, The Bonn Archaeological Software Package (WINBASP), <http://www.uni-koeln.de/~a1001/basp.html>

[14] D. I. Bibby, 'CAD Based Excavation Recording Systems and the Harris Matrix: Some Possibilities and Limitations', In W. Borner, editor, *Archaeologie und Computer, Workshop 6*, Forschungsgesellschaft Wiener Stadtarchaologie, November 2001

[15] N. S. Ryan, J. Pascoe, and D. R. Morse. In V. Gaffney, M. van Leusen, and S. Exxon, editors, 'Enhanced reality fieldwork: the context-aware archaeological assistant', *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998

[16] E. F. Codd, 'A Relational Model of Data for Large Shared Data Banks', *Comm. ACM*, vol. 13, no. 6, 1970, pp. 377–387

[17] The World Wide Web Consortium, Document Object Model (DOM), <http://www.w3.org/DOM/>

[18] A. Salminen, F.W. Tompa, 'Requirements for XML Document Database Systems', *Proceedings of the 2001 ACM Symposium on Document Engineering*, November 2001

- [19] XML:DB Initiative, XUpdate - XML Update Language, <http://xmldb-org.sourceforge.net/xupdate/>
- [20] XML:DB Initiative, Application Programming Interface for XML Databases, <http://xmldb-org.sourceforge.net/xapi/>
- [21] International Organization for Standardization (ISO). *Information Technology-Database Language SQL*. Standard No. ISO/IEC 9075:2003
- [22] The World Wide Web Consortium, J. Robie, J. Lapp, D. Schach. XML Query Language (XQL)
- [23] AT&T, Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A Query Language for XML
- [24] Don Chamberlin, Jonathan Robie, and Daniela Florescu. 'Quilt: an XML Query Language for Heterogeneous Data Sources', *In Lecture Notes in Computer Science*, Springer-Verlag, Dec. 2000
- [25] Rick Cattell et al. 'The Object Database Standard: ODMG-93', *Release 1.2. Morgan Kaufmann Publishers*, San Francisco, 1996
- [26] World Wide Web Consortium. XML Pointer Language (XPointer). W3C Working Draft, <http://www.w3.org/TR/WD-xptr>
- [27] Wolfgang Meier, eXist: An Open Source Native XML Database, *Darmstadt University of Technology*, <http://exist-db.org/webdb.pdf>
- [28] Apache Software Foundation, Apache XML-RPC, <http://ws.apache.org/xmlrpc/>
- [29] World Wide Web Consortium, SOAP Version 1.2 Part 1: Messaging Framework <http://www.w3.org/TR/soap12-part1/>
- [30] Stans Kleijnen and Srikanth Raju, 'An Open Web Services Architecture', *ACM Queue*, 2003
- [31] Curbera, F. Duftler, M. Khalaf, R. Nagy, W. Mukhi, N. Weerawarana, S., 'Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI', *Internet Computing, IEEE*, March/April 2002
- [32] GraphML Project Group, GraphML, <http://graphml.graphdrawing.org/>