

An Architecture for Archiving and Post-Processing Large, Distributed, Scientific Data Using SQL/MED and XML

Mark Papiani, Jasmin L. Wason, and Denis A. Nicole

Department of Electronics and Computer Science,
University of Southampton, Southampton, SO17 1BJ, UK.
{mp | jlw98r | dan}@ecs.soton.ac.uk

Abstract. We have developed a Web-based architecture and user interface for archiving and manipulating results of numerical simulations being generated by the UK Turbulence Consortium on the United Kingdom's new national scientific supercomputing resource. These simulations produce large datasets, requiring Web-based mechanisms for storage, searching and retrieval of simulation results in the hundreds of gigabytes range. We demonstrate that the new DATALINK type, defined in the draft SQL Management of External Data Standard, which facilitates database management of distributed external data, can help to overcome problems associated with limited bandwidth. We show that a database can meet the apparently divergent requirements of storing both the relatively small simulation result metadata, and the large result files, in a unified way, whilst maintaining database security, recovery and integrity. By managing data in this distributed way, the system allows post-processing of archived simulation results to be performed directly without the cost of having to rematerialise to files. This distribution also reduces access bottlenecks and processor loading. We also show that separating the user interface specification from the user interface processing can provide a number of advantages. We provide a tool to generate automatically a default user interface specification, in the form of an XML document, for a given database. The XML document can be customised to change the appearance of the interface. Our architecture can archive not only data in a distributed fashion, but also applications. These applications are loosely coupled to the datasets (in a many-to-many relationship) via XML defined interfaces. They provide reusable server-side post-processing operations such as data reduction and visualisation.

1 Introduction

We have been working with the UK Turbulence Consortium [1] to provide an architecture for archiving and manipulating the results of numerical simulations. One of the objectives of the consortium is to improve collaboration between groups working on turbulence by providing a mechanism for dissemination of data to members of the turbulence modelling community. The consortium is now running simulations on

larger grid sizes than has previously been possible, using the United Kingdom's new national scientific supercomputing resource.¹ One complete simulation, comprising perhaps one hundred timesteps, requires a total storage capacity of some hundreds of gigabytes. This necessitates new Web-based mechanisms for storage, searching and retrieval of multi-gigabyte datasets that are generated for each timestep in a simulation. In particular, an architecture is required that can minimise bandwidth usage whilst performing these tasks.

The Caltech Workshop on Interfaces to Scientific Data Archives [2] identified an urgent need for infrastructures that could manage and federate active libraries of scientific data. The workshop found that whilst databases were much more effective than flat files for storage and management purposes, trade-offs existed as the granularity of the data objects increased in size. If a database is being created to manage metadata describing scientific results, then ideally the database should also be used to store the actual scientific result data in a unified way. However for large output files it becomes costly and inefficient to store the data as binary large objects (BLOBS) within the database.

We describe a solution that uses an implementation of the new SQL:1999 (formerly known as SQL3, see for example [3]) DATALINK type, defined in *SQL Management of External Data (SQL/MED)* [4], to provide database management of scientific metadata and large, distributed result files simultaneously with integrity. We apply this technology to the Web, by providing a user-interface to securely manage large files in a distributed scientific archive, despite limited bandwidth.

A database table containing an attribute defined as a DATALINK type can store a URL that points to a file on a remote machine. Once a URL has been entered into the database, software running on the remote machine ensures that the file is treated as if it was actually stored in the database, in terms of security, integrity, recovery and transaction consistency. We use this mechanism to allow large result files to be distributed across the Web.

Our system generates a user interface, from a specification defined in Extensible Mark-up Language (XML) [5], to a database that supports DATALINKs. We have created an XML Document Type Definition (DTD) to define the structure of the XML file.

Our architecture provides the following features for scientific data archiving:

1. *The system can be accessed by users of the scientific archive, who may have little or no database or Web development expertise.* Users are presented with a dynamically generated HTML query form that provides a search interface akin to Query by Example (QBE) [6]. We generate this interface automatically from an XML user interface specification file (XUIS). The XUIS specifies the Web interface to a particular object-relational database (which may contain BLOBS and DATALINK

¹ A 576 processor Cray T3E-1200E situated at the University of Manchester, which forms part of the *Computer Services for Academic Research (CSAR)* service run on behalf of the UK Research Councils. <http://www.csar.cfs.ac.uk/>

types). This file is constructed automatically using metadata extracted from the database catalogue but can be customised to provide specialised features in the interface.

2. *The default interface specification adds a novel data browsing facility to maintain a Web-based feel.* Contrary to Manber's statement that finding ways to include browsing in even relational databases would be a great step [7], we show that one simple way to browse relational databases is to follow relationships between tables, implied by referential integrity constraints defined in the database catalogue. We use this principle to provide hypertext links in displayed results that access related information.
3. *Large result files can be archived at (or close to) the point where they are generated.* For the UK Turbulence Consortium, this means that files can be archived at the Manchester site on a local machine that is connected via a high-speed link to the supercomputer. By entering the URLs of these files into a DATALINK column of a remote database (via a Web-based interface to the remote database), database security and integrity features can then be applied to the files. An alternative to this, which achieves similar database security and integrity for result files, is to use a Web interface to upload a file across the Internet and then store it as a BLOB in a centralised archive at the new location. However, this alternative is not feasible for large files due to limited Internet bandwidth. Even if a file can be transferred to a centralised site, additional processing cost is incurred (which is not present with the DATALINK mechanism) when loading the file as a BLOB type into the database.
4. *Because simulation results are stored in unmodified files, existing post-processing applications, that use standard file I/O techniques, can be applied to the files without having to rewrite the applications.* An alternative would be to modify applications to first access result objects from a database but this would be very undesirable for many scientific users who often apply post-processing codes written in FORTRAN.
5. The Caltech workshop [2] recommended 'cheap supercomputers for archives'. The report suggests that research is necessary to establish whether high-performance computing resources, built from commodity components, are viable for data-intensive applications (as well as compute-intensive applications, as has been shown to be the case in for example, the Beowulf project [8]). *We are using our architecture to build a large scientific archive from commodity components, with many distributed machines acting as file servers for a single database.* Security, backup and integrity of the file servers can be managed using SQL/MED. This arrangement can provide high performance in the following areas:
 - Data can be distributed so that it is physically located closest to intensive usage.
 - Data distribution can reduce access bottlenecks at individual sites.
 - Each machine provides a distributed processing capability that allows multiple datasets to be post-processed simultaneously. Suitable user-directed post-processing, such as array slicing and visualisation, can significantly reduce the amount of data that needs to be shipped back to the user. Post-processing codes that have been archived by our system can be associated with remote data files

using the XUIS. This allows dynamic server-side execution of the stored applications, with chosen datasets as input parameters.

The rest of this paper is structured as follows. Section 2 begins with a high level description of our architecture. We next describe how we use XML to specify customisable user interfaces with searching and browsing capabilities. We explain the functionality that SQL/MED brings to our architecture. Section 3 describes some related work. Finally section 4 draws some conclusions from our work.

2 System Architecture and User Interface

This section starts with a high level view of our architecture. We then describe how XML is used to specify the functionality of the user interface. We show how our system supports two kinds of information retrieval, searching and browsing. We illustrate four different types of browsing links that our system can include automatically in Web pages displaying query results. ‘DATALINK browsing’ is particularly important in our architecture for managing large, distributed scientific data files. We therefore provide an overview of the DATALINK type defined in the draft SQL/MED standard. We also describe how the user interface can be customised through modifications to the XUIS, and how post-processing codes can extend the functionality of the interface by the inclusion of ‘operations’ in the XUIS.

2.1 System Architecture

The architecture of our system is shown in Fig 1. It consists of a database server host (located at Southampton University) and a number of file server hosts that may be located anywhere on the Internet. All of these hosts have an installed Web server to allow HTTP (or HTTPS – HTTP plus Secure Socket Layer) communications directly from a Web browser client.

A user of our system initially connects to the Web server on the database server host. The URL of our system invokes a Java Servlet [9] program. Separate threads within the Servlet process handle requests from multiple Web browser clients. Each user is first presented with a login screen. Once a user has been verified interaction with the database is possible via HTML pages that are dynamically generated by our Servlet code. These pages consist of HTML forms, Javascript and hypertext links.

The database server stores metadata describing the scientific information such as, simulation titles, descriptions and authors. This data is stored locally in the database and is accessed by our Servlet code using Java Database Connectivity (JDBC) [10]. The data is represented by tables with attributes defined as standard SQL-types, BLOB types, or CLOB (character large object) types. The latter types are used in our system to store *small* image/video files, executable code binaries or *small* ASCII files containing source code or descriptive material for the turbulence simulations.

For scientific data archiving, an essential feature of our interface is the novel use of remote file servers which store files referenced by attributes defined as DATALINK SQL-types. These file servers manage the *large* files associated with simulations, which have been archived where they were generated. When the result of a database access yields a DATALINK value, our interface presents this to the user as a hypertext link that can be used to download the referenced file. The URL contains an encrypted key that is prefixed to the required file name. This key is verified by DATALINK file manager code (running on the file server host) which intercepts attempts to access any files controlled by the remote database. Without this key, files cannot be accessed, either via the locally installed Web server or directly from the file system by a locally connected user. As well as allowing a user to simply download a dataset, our interface also allows user-selected post-processing codes to execute on the remote file server host to reduce the volume of data returned.

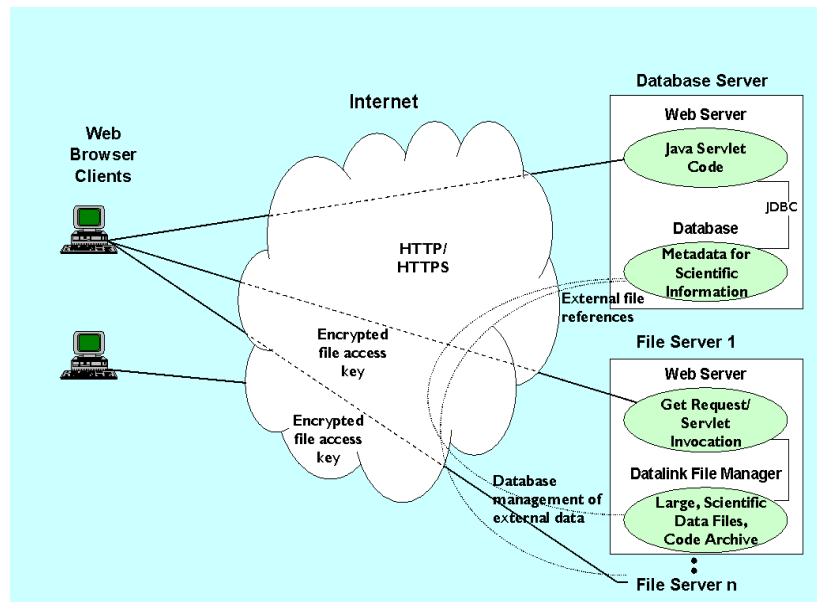


Fig. 1. System architecture

2.2 XML Specification of the User Interface

Our system is started by initialising the Java Servlet code with an XUIS. This initialisation can take several seconds but it is a one-off cost that is not repeated during subsequent requests from users. A default XUIS can be created prior to system initialisation using a tool that we provide. This tool, written in Java, uses JDBC to extract data and schema information from the database being used to archive simulation results. This default XUIS conforms to a DTD that we have created. The default XUIS

can be customised prior to system initialisation. The XUIS contains table names, column names, column types, sample data values for each column, and details of primary keys and foreign keys that participate in referential integrity constraints. The XUIS also allows aliases to be defined for table and column names.

In the following sections we explain how this information is used to provide an interface with searching and browsing capabilities and how it facilitates customisation of the interface and dynamic execution of post-processing applications.

2.3 Searching and Browsing Data

After logging in users of our interface can begin to locate information in the scientific archive by searching or browsing data or by using a combination of both techniques.

2.3.1 Searching

To search for data a user selects a link to a query form for a particular table. On the query form, the user selects the fields to be returned. Also for each field present, restrictions including wildcards may be put on the values of the data. After submission the page returned to the user is a table whose columns are the marked fields and whose rows are the data items satisfying any restrictions that were entered.

The query form provides several other features to aid direct searching. These include selection of restrictions and sample values from drop-down lists. By offering the user choices of attribute names, relation names and operators, the instance of syntactic error during query formation can be reduced. Providing attribute names and sample values can aid substantially in narrowing the semantic meaning of a domain [11]. The table names, column names, data types and sample values used in the query form are obtained from the XUIS used to initialise the system.

2.3.2 Browsing

Fig. 2 shows a sample result screen that was generated after a query on the SIMULATION table described by the schema shown in Fig. 3. The SIMULATION result table contains data corresponding to simple SQL-types, such as the title and date for a simulation. It also contains a number of hypertext links. Our interface automatically inserts several different types of link that we describe below.

The SIMULATION result table (Fig. 2) displays a link on *each* AUTHOR_KEY attribute. This is because AUTHOR_KEY is a foreign key in the SIMULATION table that references the AUTHOR table (see Fig. 3). Selecting a link on an AUTHOR_KEY value will retrieve full details of the author by displaying the appropriate row from the AUTHOR table.

The inverse relationship provides a *primary key link* to a table in which the primary key value appears as a foreign key. Each individual SIMULATION_KEY value of Fig. 2 contains a *primary key link*. Because primary keys may appear in several tables as a foreign key, there may be a choice of tables to browse to. Selecting one of these values will return all the rows that the key appears in from one of the referenced ta-

bles. The particular table is indicated in the currently checked radio button in the column header.

The table of data returned by our software interface may be quite large. Cells associated with simple values, such as numbers or short character strings, display the actual values. As well as simple types, we also use BLOB and CLOB types, to store small files that can be uploaded over the Internet. Cells associated with these types display a *LOB link* to the object. Clicking on such a link causes the data associated with the cell to be rematerialised and returned to the client. The link displays the size of the object in bytes, which may help users decide whether they want to retrieve the object. For example, Fig. 3 indicates that the DESCRIPTION attribute in the SIMULATION table is a CLOB type. Selecting the hypertext links on the DESCRIPTION fields in Fig. 2 will retrieve the description and display it directly in the browser window since it contains character data. For BLOB data (e.g. a stored image or executable) selecting the link will allow the user to retrieve the data as a file.

Row	SIMULATION_KEY links to SIMULATION_KEY in CODE_FILE SIMULATION_KEY in RESULT_FILE SIMULATION_KEY in VISUALISATION_FILE	TITLE	DESCRIPTION AUTHOR_KEY links to AUTHOR	DATE_RUN	URL
1	S19990110150932	Channel Flow DNS	504 Byte A19990110151042		
2	S19990209150932	Laminar Separation Bubbles DNS	704 Byte A19990209151042	1997/1998	

Fig. 2. Result table from querying SIMULATION table

The sample schema of Fig. 3 also contains three attributes of the DATALINK SQL-type. These DATALINK attributes serve the main purpose of our architecture – to archive large scientific data via the Web. The attributes in the Turbulence Consortium schema are used for storing result files, code files, and visualisation files. These values are displayed as a filename in a result table, with a hypertext link that contains an encrypted key, required to access the file from the remote file server (see section 2.4).

Hyperlinks on BLOB/CLOB and DATALINK types are included in the interface if the XUIS specifies a column as consisting of one of these types. Foreign key and primary key links are included in the XUIS by default if referential integrity constraints

that registered files are not renamed, deleted and optionally, check the user's access authority.

A DATALINK value can be entered via a standard SQL *INSERT* or *UPDATE* statement. If read permission is managed by the database, an SQL *SELECT* statement retrieves the value in the form:

```
http://host/filesystem/directory/access_token;filename
```

The file can then be accessed from the filesystem in the normal way using the name: *access_token;filename*, or, by using the full URL if the file is placed on a Web server (as in our system). The access tokens have a finite life determined by a database configuration parameter. This can be set to expire after an interval.

2.5 Interface Customisation through XUIS Modification

The XUIS can be customised to provide aliases for table names and column names as well as user-defined sample values. It is also possible to prevent tables or columns being displayed in the query forms and results, by removing them from the XUIS. Hypertext links for navigation between tables can also be added or removed by modification of the XUIS. A fragment of the XML for the XUIS is shown below:

```
<table name="AUTHOR" primaryKey= "AUTHOR.AUTHOR_KEY">
  <tablealias>Author</tablealias>
  <column name="AUTHOR_KEY" colid="AUTHOR.AUTHOR_KEY">
    <type><VARCHAR/><size>30</size></type>
    <pk><refby tablecolumn="SIMULATION.AUTHOR_KEY"/></pk>
  <samples>
    <sample>A19990110151042</sample>
    <sample>A19990209151042</sample>
  </samples>
</table>
```

Initial feedback from users indicated that they would like to replace the keys displayed in the results with more meaningful data. The XUIS can be modified to specify a 'substitute column' to be displayed in the results in place of foreign keys. The substitute column is a user-specified column from the table that the foreign key references. For example, the *AUTHOR_KEY* attributes in the *SIMULATION* table of Fig. 2 can be replaced by, say, the *NAME* attribute from the referenced *AUTHOR* table. These values still provide a browsing link to the full author details. The XUIS can also be customised to replace primary keys with the names of linked tables.

2.6 Suitable Processing of Data Files Prior to Retrieval: 'Operations'

McGrath [12] asserts that users of scientific data servers usually only require a selected part of a dataset that is typically stored in a large complex file, and that it is critical therefore, to provide a means to identify data of interest, and to be able to retrieve the selected data in a useable format. Referring back to our system architecture of Fig.1 we see the Web servers on the remote file servers can process a standard

HTTP 'Get request' to return a complete result file to the client (if the encrypted file access key is correct). This is the functionality that we have described so far. However, Fig. 1 also shows that a Java Servlet can be invoked on the file server to handle the incoming request.

Our architecture allows the XUIS to be modified to allow post-processing applications that have been archived using DATALINK values to be dynamically executed server-side to reduce the data volume returned to the user. These applications can consist of Java classes or any other executable format, suitable for the file server host on which the data resides, including C, FORTRAN and scripting languages. These applications do not have to be specially written for our architecture (in fact, *operations* stored as DATALINKs can be downloaded separately for standalone execution elsewhere) and they can be packaged in a number of different formats including various compressed archive formats (such as tar.Z, gz, zip, tar etc.). The only restriction is that the initial executable file accepts a filename as a command line parameter. This filename will correspond to the name of a dataset to be processed. Archived applications are associated with a number of archived datasets using a mark-up syntax that we have defined for 'operations' in the XUIS. If the application allows other user-specified parameters, the syntax for *operations* has been defined so that an HTML form will be created to request these parameters at invocation time. The XML syntax we have defined for *operation* parameters is similar to that used for HTML forms. Applications that correspond to *operations* do not have to be stored as DATALINKs. They can also be specified in the XUIS in the form of URLs that refer to CGI programs or Servlets that run on the file server host to provide a post-processing capability. A fragment of the XML for an *operation* is shown below:

```
<column name="DOWNLOAD_RESULT"
  colid ="RESULT_FILE.DOWNLOAD_RESULT">
  <type><DATALINK/></type>
  <operation name="GetImage" type="JAVA"
    filename="GetImage.class" format="jar"
    guest.access="true" column="false">
  <if>
    <condition colid="RESULT_FILE.SIMULATION_KEY">
      <eq>'S19990110150932'</eq></condition>
  </if>
  <location>
    <database.result
      colid="CODE_FILE.DOWNLOAD_CODE_FILE">
      <condition colid="CODE_FILE.CODE_NAME">
        <eq>'GetImage.jar'</eq>
      </condition>
    </database.result>
  </location>
  <parameters>
    <param>
      <variable>
        <description>DO you require FORTRAN
          or JAVA Sample Code?</description>
        <input type="radio">
```

```

name="output_format"
value="Fortran" >FORTRAN
</input>
</param>

```

An example of the processing associated with an *operation* stored as a DATALINK follows. Fig. 4 shows the result of a query on the RESULT_FILE table from the schema of Fig. 3. The columns containing a cog icon in the column header present the user with operations to execute against the preceding DATALINK column containing dataset result files.

Row	DOWNLOAD_RESULT links to	GetChunk	GetImage	MEASUREMENT	FILE_FORMAT	RESULT_DESCRIPTION	TITLE SIMULATION links to
1	data_01.dat	GetChunk	GetImage	u,v,w,p	IEEE binary		Channel Flow I
2	data_80.dat	GetChunk	GetImage	u,v,w,p	IEEE binary		Channel Flow I
3	data_79.dat	GetChunk	GetImage	u,v,w,p	IEEE binary		Channel Flow I
4	data_78.dat	GetChunk	GetImage	u,v,w,p	IEEE binary		Channel Flow I
5	annras.hdf				HDF	2 raster image plus nice annotation	dummy simulati HDF data files
6	sdsdata.hdf				HDF	HDF SDS and Vdata example	dummy simulati HDF data files
7	layers8bit.hdf				HDF	8-bit raster images for layer feature demo	dummy simulati HDF data files

Fig. 4. Result table showing ‘operations’ available for post-processing datasets

Selecting the ‘GetImage’ hyperlink corresponding to the DATALINK value containing ‘data_01.dat’ produces a screen that describes what the operation does, and requests that the user specifies (via text boxes, radio buttons etc.) any input parameters that were defined for the operation in the XUIS. The information contained in this form was specified in the XUIS and the operation itself is stored as a DATALINK in the CODE_FILE table of the schema shown in Fig. 3. The example ‘GetImage’ operation extracts a user-specified slice from a dataset corresponding to a simulation of turbulence in a 3-D channel and returns a GIF image of a user-selected velocity component or pressure from the flow field. The result (applied to ‘data_01.dat’) is shown in Fig 5. When the user submits the request this initiates a sequence of processing. The location of the DATALINK file corresponding to the dataset to be processed and the location of the DATALINK file corresponding to the ‘GetImage’ operation along with the selected parameter values, are passed to a Servlet running on the file server host. This Servlet also receives other information (stored as hidden fields in the HTML form that was initially specified in the XUIS, such as the executable type for the operation (e.g. FORTRAN, C, Java) and whether the application is contained in an compressed archive format.

The Servlet then makes external system calls to unpack the application in the temporary directory created for the user and to execute the application with the given dataset filename and other command line arguments. The Servlet redirects standard output and standard error from the executable to files in the temporary directory. When the application ends any output is returned to the user as illustrated in Fig. 5. The user is also given the opportunity to download any files that are created by the application. Since the 'GetImage' operation produces GIF images, which the browser understands, these are also displayed on the Web page.

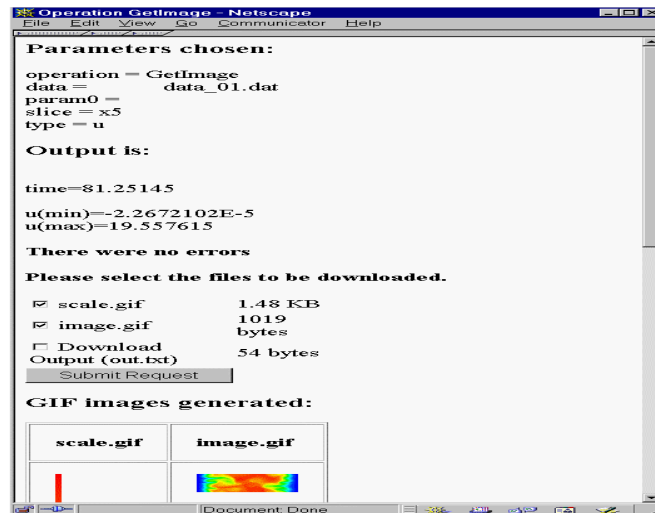


Fig. 5. Output from 'operation' execution

When the user hits 'submit request' in the results page of Fig. 5 a multipart HTTP response is used to return all the requested files to the user, and the temporary directory and all of its contents are deleted. If the user does not download the files in this way the temporary directories are removed periodically when they are no longer being used. This is monitored by keeping track of active Servlet sessions IDs.

As well as incorporating stored operations with XML defined interfaces, the system also allows authorised users to upload arbitrary Java programs for secure server-side execution, to post-process datasets stored on the file server hosts. (In Fig. 4 this facility is requested using the arrow icon.)

3 Related Work

There has been a substantial body of research in the area of graphical interfaces to databases. A comprehensive survey can be found in [13]. Most of the systems that incorporate data browsing are applied to database models that have *explicit* relationships between objects in the database. Such database models include entity-

relationship (E-R) and object-oriented data models (see for example the PESTO interface described in [14]). Our data browsing technique does not rely on explicit relationships defined in the data model, but extracts implied relationships from the relational model to dynamically include hyperlinks in results. Our interface is automatically generated, requiring no HTML page maintenance. As such, we do not rely on our scientific user-base having Web and database development experience. Our interface also allows customisation through changes to XML mark-up. Furthermore our interface can manage distributed data through the DATALINK SQL-type and be extended by the inclusion of executable code described by XML mark-up.

NCSA's *Scientific Data Service* (SDS) [15] [12] provides a Web-based interface targeted at manipulating scientific datasets. It consists of a CGI program for browsing data in a number of scientific formats such as the Hierarchical Data Format (HDF) (<http://hdf.ncsa.uiuc.edu>). This CGI program is written in C and provides functionality such as visualisation and extraction of subsets of data specifically written for the supported scientific data formats. However SDB does not provide sophisticated storage and search capabilities for archiving and selecting the data files that it can process. It assumes that these files are simply placed in a directory structure belonging to the Web server. We were able, however, to incorporate SDB as an 'operation' in our interface by the inclusion of simple mark-up in the XUIS that specified SDB for post-processing result files in HDF format. This allows SDB to benefit from our secure data management architecture and search capabilities. The XML fragment to include the 'SDB' operation is shown below:

```
<operation name="SDB" type="" filename=""
  format="" guest.access="true" column="false">
  <if><condition colid="RESULT_FILE.FILE_FORMAT">
    <eq>'HDF'</eq>
  </condition></if>
  <location>
    <URL> http://dns.ecs.soton.ac.uk/cgi-bin/SDB</URL>
  </location>
  <description>NCSA Scientific Data Browser</description>
</operation>
```

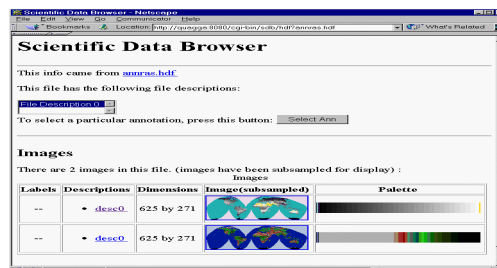


Fig. 6. NCSA'S SDB [15] has been specified as an 'operation' in the XUIS

4 Conclusions

We have constructed a prototype system to meet a requirement of the UK Turbulence Consortium to make available to authorised users, large result files from numerical simulations, with a total storage requirement in the hundreds of gigabyte range. We also archive applications that can be dynamically invoked to post-process data. A demonstration is available at <http://www.hpcc.ecs.soton.ac.uk/~turbulence>.

Our interface is specifically aimed at users with a scientific background who are not familiar with SQL. As such, we aim to help users locate scientific data files of interest, using an intuitive searching and browsing mechanism in keeping with a Web-based look and feel. One of the implications of this is that we automate the interface construction so that it requires little database or Web development experience to install and access. This is a generic, schema-driven system that can be used to manage many different types of large, distributed data archives. We achieve the automated construction by allowing the user interface specification to be defined in an XML file used to initialise the system and by providing a tool that can generate a default XML specification. Separating the user interface specification from the user interface processing can provide a number of further advantages:

- The user interface, although schema driven can be customised to use aliases for table and column names and to present different sample values. Tables and attributes can also be hidden from view.
- Hypertext links to related data can be specified in the XML even if there are no referential integrity constraints defined for the database.
- Different Users (or classes of user) can have different XML files thereby providing them with different user interfaces to the same data.
- For scientific data archiving a major benefit, facilitated by the XML user interface specification, is the capability to associate ‘operations’ with database columns, so that a user can extend the interface by including standard post-processing codes. These applications are loosely coupled to the datasets via XML defined interfaces. They provide reusable server-side post-processing operations such as data reduction and visualisation.

Our architecture uses distributed commodity computing and non-proprietary technologies such as the new SQL DATALINK type, defined in SQL/MED. We demonstrate a complete architecture including Web-based user interface for archiving large, distributed files whilst maintaining database security, integrity and recovery. As far as we are aware, this is the first use of this technology for Web-based scientific data archiving.

Bandwidth is a limiting factor in our environment. We greatly reduce bandwidth requirements by avoiding costly network transfers associated with uploading data files to a centralised site and by allowing data reduction through post-processing. Data distribution also reduces retrieval bottlenecks at individual sites.

5 Acknowledgements

The UK Turbulence Consortium (EPSRC Grant GR/M08424) provided data for this project. IBM's DB2 Scholars programme provided us with DB2 licenses. The DB2 Data Protection and Recovery Team at IBM were extremely helpful when responding to questions regarding DB2's DATALINK implementation.

References

1. Sandham, N.D. and Howard, R.J.A. Direct Simulation of Turbulence Using Massively Parallel Computers. *In: A. Ecer et al., eds. Parallel Computational Fluid Dynamics '97*, Elsevier, 1997.
2. Williams, R., Bunn, J., Reagan, M., and Pool, C., T. *Workshop on Interfaces to Scientific Data Archives*, California, USA, 25-27 March, 1998, Technical Report CACR-160, CALTECH, 42pp.
3. Eisenberg, A. and Melton, J., SQL:1999, formerly known as SQL3. *SIGMOD Record*, 28(1), March, 1999.
4. Mattos, N., Melton, J. and Richey, J. Database Language SQL-Part 9:Management of External Data (SQL/MED), ISO/IEC Committee Draft, CD 9075-9, December, 1988. <ftp://jerry.ece.umassd.edu/isowg3/dbl/YGJdocs/ygj023.pdf>
6. Jim Bray, J., Paoli, J. and Sperberg-McQueen, C., M. eds. Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February, 1998. <http://www.w3.org/TR/REC-xml>
7. Zloof M.M. Query By Example. *American Federation of Information Processing (AFIPS) Conf. Proc.*, Vol. 44, National Computer Conference, 1975, 431-8.
8. Manber, U. Future Directions and Research Problems in the World Wide Web. *Proc ACM SIGMOD Conf.*, Montreal, Canada, June 3-5, 1996, 213-15.
9. Warren, M., S., et al. Avalon: An Alpha/Linux Cluster Achieves 10 Gflops for \$150k. Gordon Bell Price/Performance Prize, Supercomputing 1998. <http://cnls.lanl.gov/avalon/>
10. Davidson, J., D., and Ahmed, S. Java Servlet API Specification, Version 2.1a, November, 1998. <http://java.sun.com/products/Servlet/index.html>
10. White, S., Hapner, M. JDBC 2.0 API, Sun Microsystems Inc., Version 1.0, May, 1998.
11. Haw D., Goble, C., A., and Rector, A., L. GUIDANCE: Making it easy for the user to be an expert. *Proc. 2nd Int. workshop on User Interfaces to Databases*, Ambleside, UK, 13-15th July, 1994, 19-44.
12. McGrath, R., E. *A Scientific Data Server: The Conceptual Design. White Paper*, NCSA, University of Illinois, Urbana-Champaign, January, 1997.
13. Catarci, T., Costabile, M., F., Levialdi, S., and Batini, C. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, 8, 1997, 215-60.
14. Carey, M., J., Haas, L., M., Maganty, V., and Williams, J., H. PESTO: An Integrated Query/Browser for Object Databases. *Proc. VLDB Int. Conf.*, India, 3-6 September, 1996, 203-14.
15. Yaeger, N. *A Web Based Scientific Data Access Service: The Central Component of a Lightweight Data Archive*, National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign. <http://hopi.ncsa.uiuc.edu/sdb/sdb.html>