An Engineering Data Access System for a Finite Element Program

Jun Peng [1], David Liu [2] and Kincho H. Law [3,*]

Abstract

This paper describes a prototype implementation of an engineering data access system for a finite element analysis program. The system incorporates a commercial off-the-shelf (COTS) database as the backend to store selected analysis results; and the Internet is utilized as the communication channel to access the analysis results. The objective of using an engineering database is to provide the users the needed engineering information from readily accessible sources in a ready-to-use format for further manipulation. Three key issues regarding the engineering data access system are discussed, including data modeling, data representation, and data retrieval. The engineering data access system gives great flexibility and extendibility to the data management in finite element programs and can provide additional features to enhance the applicability of finite element analysis software.

Keywords

Engineering database; Finite element program; Selective data storage; Object serialization; Data query language; Project management; Multi-tired architecture; Software service

---

[1] Graduate Student, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305. E-mail: junpeng@stanford.edu

[2] Graduate Student, Department of Electrical Engineering, Stanford University, Stanford, CA 94305. E-mail: davidliu@stanford.edu

[3,*] Professor, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305. E-mail: law@cive.stanford.edu; Tel: +1-650-7253154; fax: +1-650-7237514

1. Introduction

The importance of engineering data management is increasingly emphasized in both industrial and research communities. The objective of using an engineering database is to provide the users the needed engineering information from readily accessible sources in a ready-to-use format for further manipulation. Such trend can also be observed in the field of finite element analysis (FEA). Modern finite element programs are increasingly required to be linked to other software such as Computer-Aided Design (CAD) programs, graphical processing software, or databases [1]. Data integration problems are mounting as engineers confront the need to move information from one computer program to another in a reliable and organized manner. The handling of data shared between disparate systems requires the definition of persistent and standard representations of the data, and corresponding interfaces to query the data. Data must be represented in such a manner that they can facilitate interoperation with humans or mechanisms that use other persistent representations [2]. As new element and material types and new solution strategies continue to be introduced to a FEA program, the data structure of a FEA program is becoming ever more complex [3]. To cope with the evolutionary nature of a FEA program, data management needs to be flexible and extendible.

This paper presents a prototype design and implementation of an engineering data access system for a FEA program. This is part of an effort in building an Internet-enabled collaborative software system for the development of a finite element analysis program [3-5]. The objective of the collaborative framework is developing a software platform that would allow researchers and engineers to easily access to the FEA platform, and to incorporate new element technologies and solution strategies for nonlinear dynamic analysis of structures. Fig. 1 shows the system architecture of the distributed collaborative structural analysis platform. The core of the platform is based on an object-oriented finite element program, which is named OpenSees

[6] (Open System for Earthquake Engineering Simulation, for details, see *http://opensees.berkeley.edu*). OpenSees is currently being developed at the Pacific Earthquake Engineering Research (PEER) Center. In the collaborative software model, the analysis core is running on a central server as a compute engine. A compute engine is a remote software program that takes a set of tasks from clients, runs them, and returns the results. Users of the system play the role of clients and have direct or remote access to the core program and analysis results. Elements can be built as separate services and be linked with the core server via a standard protocol. The element services can be accessed by the core remotely over the Internet. An engineering data management system is linked with the central server to provide the persistent storage of selected analysis results.

To design the engineering data management system, we first need to decide what kind of media should be used to store the data. Presently, the data storage for finite element analysis programs primarily relies on the direct manipulation of file systems. Since most modern operating systems have built-in support for file creation and file usage, directly using file systems to store data is a straightforward process. However, there are many intrinsic drawbacks associated with the direct usage of file systems for storing large volume of data. File systems generally do not guarantee that data cannot be lost if it is not backed up, and they do not support efficient random access in which case the locations of data items in a particular file are unknown. Furthermore, file systems do not provide direct support for a query language to access the data in files, and their support for a *schema* of the data is limited to the creation of file directory structures. Finally, file systems cannot guarantee data integrity in the case of concurrent access. Instead of directly using file systems to store the analysis results of a finite element analysis, these results can also be saved in database systems. Most database management systems (DBMS) allow certain structures for the saved data, allow the users to query and modify the data, and help manage very large amounts of data and many concurrent operations on the data.

In the prototype implementation of the engineering data access and management system, both file system and database systems can be employed for data storage. Because of the benefits of database systems over file systems, we focus our efforts on using COTS database systems to store the selected analysis results. The techniques described herein for database systems can also be applied to data management systems based on file systems.

By adopting a COTS database system, the collaborative system can address many of the problems encountered by the prevailing data management based on file systems. A customized interface to link the OpenSees core with the database system is provided. In the prototype system, the data management is supported by using Oracle 8i [7] DBMS (Database Management System) with customized enhancements to meet certain additional requirements. The Internet is utilized as the data communication channel. The online data access system would allow the users to query useful analysis results, and the information retrieved from the database through the core server is returned to the users in a standard format. Since the collaborative system is using a centralized server model, numerous users would access the system with multiple projects. The data access system can also support certain project management functions, and has simple access control and revision control mechanisms.

In this paper, we will first present the overall architecture of the data access system. The system is built with a multi-tier architecture that provides a flexible mechanism to organize distributed client-server systems. The communication between different components will be discussed in details. The rest of the paper addresses three aspects of the data management system:

- **Data storage scheme**: A selective data storage scheme is introduced to provide flexible support for the tradeoff between the time used for reconstructing analysis domain and the space used for storing the analysis results. Rather than storing all the interim and final analysis results, the data management system allows saving only the selected analysis data in the database. That is, the user has the flexibility to specify storing only the required and needed data. All the other analysis results can be accessed through the FEA core with certain re-computation during the postprocessing phase of an analysis.

- **Data representation**: Data are organized internally within the FEA analysis core based on an object-oriented model. Data saved in the COTS database are represented in three basic data types: Matrix, Vector, and ID. Project management capabilities, especially access control and revision control, are also supported by the system. For external data representation, XML (eXtensible Markup Language) is chosen as the standard for representing data in a platform independent manner.

- **Data retrieval**: The system needs to support the interaction with both human and other application programs. A data query language is defined to provide support for data retrieval as well as postprocessing functionalities. Through the data query language, users can have uniform access to the analysis results by using a web-browser or other application programs, such as MATLAB.

2. Related work

The importance of data management system in scientific and engineering computing has been recognized for over thirty years. Techniques for generalized data management were gradually making inroads in scientific computing during the 1970s. This development paralleled in many ways the rapid acceptance of the centralized database concept in business-oriented processing. However, engineering data manipulation systems faced a specialized environment with its own set of operational requirements. Tuel and Berry [8] outlined the basic database requirements for computer-based information systems needed to handle the diverse and changing information requirements of geographic facilities. Recognizing the fact that computer solution of structural analysis problems results in the generation of large volumes of data, Lopez et al. [9] proposed a database management system for large-scale structural engineering problems. To present the specialized environment and operational requirements of engineering data management systems, Felippa [10-12] published a series of three papers on database usage in scientific computing. These papers reviewed general features of scientific data management from a functional standpoint, including the description of a database-linked engineering analysis system, the organization of a database system, and the program operational compatibility. The general data structures and program architecture were also presented, together with the issues regarding implementation and deployment. In 1983, Blackburn et al. [13] described a relational database (RDB) management system for computer-based integrated design, including application to the analysis of various structures to demonstrate and evaluate the ability of RDB system to store, retrieve, query, modify, and manipulate data. These papers emphasized the importance of centralized data management for large-scale computing. Two factors that determined the favor of centralized scientific data management were: the sheer growth of large-scale engineering analysis codes to the point of incipient instability as regards to propagation of local program errors, and the appearance of integrated program networks that share a common project database. Centralized data management was most effective when used in conjunction with a highly-modular, structured program architecture [11].

The role of databases as repositories of information (data) highlighted the importance of data structures. The component data elements of data structures could be either atomic (i.e. non-decomposable) or the data structures themselves. The relationships between these component data elements constitute the structure and have implications for the functions of the data structure [14]. Several general approaches for organizing the data models have been developed. They are: the hierarchical approach, the network approach, the relational approach, and the object-oriented approach. The hierarchical approach and the network approach are the traditional means of organizing data and their relationships. The relational model has been adopted in several finite element programs [13, 15, 16]. The object-oriented approach is the foundation for many object-oriented database management systems, such as EXODUS [17], which is an extensible database system to facilitate the fast development of high-performance, application-specific database systems. No matter which data model is used, data structures need to be self-describable [10]. This can be achieved by requiring each program to label its output data, i.e. to attach a descriptive label to each data structure that would be saved in the database. Such tags can then be examined by the control structure of other programs and appropriate actions can be taken.

Today, file system remains the most popular approach in managing data storage. The loosely-coupled systems could talk to each other through the same file system. However, this does not imply that they speak the same language. In other words, data placed by an application program into the file system may not be acceptable to another program because of format incompatibility. To tackle this problem, Yang [16] defined a standard file format for the analysis data, called the universal file (UF). Two interfaces have been proposed. The first is a specified set of subroutines to transfer the input or output files of the programs into UF. The second is a set of subroutines to translate UF into the database configured to aid FEM modeling operations. Another effort to address file format compatibility is the neutral file approach introduced for integrated Computer-Aided Design (CAD) systems [18]. The neutral file approach establishes a standard file format and information structures to be used for the digital representation and communication of product definition data. Using a neutral standard for transferring information across systems drastically reduces the requirements for file format translators.

For finite element programs, the postprocessing functions need the recovery of analysis results and provide extensive graphical and numerical tools for gaining an understanding of results. In this sense, querying database is an important aspect and query languages need to be constructed to interrogate databases. A free-format data query language has been designed and provided in SADDLE (Structural Analysis and Dynamic Design Language) [15]. Although the commands to create, edit, and update the data have been provided, the query language was hard for human to interpret. In order to manage engineering databases, a data query system should provide query commands that resemble English, as well as simple data manipulation procedures [19]. Simple natural language interface has also been attempted in querying the qualitative description of dynamic simulation data [20]. The commands of this language are easy for human to interpret, but it is difficult to write a parser.

3. Data access system architecture

In the collaborative computing environment, the finite element analysis core is running on a central server as a compute engine [3]. A compute engine is a remote software program that takes a set of tasks from clients, runs them and returns the result. A COTS database system is utilized to provide the data storage for the central finite element computer engine. Since a client-server model is utilized for the collaborative system, the engineering data access system needs to be designed accordingly. Fig. 2 depicts the architecture of the data access system, which employs a multi-tiered architecture as opposed to the traditional two-tier client-server architecture. The multi-tiered architecture provides a flexible mechanism to organize distributed client-server systems. Since the components in the data access system are modular and self-contained, they can be designed and developed separately.

- A standard interface is provided for the **Remote Client** programs to access the server system. Application programs, such as web browsers or MATLAB, can access the server core and the analysis results from the client site via the pre-defined communication protocols. Using dynamic HTML pages and JavaScript code, together with the mathematical manipulation and graphic display capability of MATLAB, the client has the ability to specify the format and views of analysis results. Details about the user interfaces of the collaborative framework can be found in Peng and Law [3].

- Java Servlet enabled **Web Server** is employed to receive the requests from the clients and forward them to the *Application Server*. The *Web Server* also plays the role of re-formatting the analysis results in certain HTML format for the web-based clients. In the prototype system, Apache HTTP web server is employed to handle the requests from the users, and Apache Tomcat is utilized as the Java Servlet server.

- The **Application Server** is the middle layer for handling communication between the *Web Server* and the *Database*. The *Application Server* provides the core functionalities for performing analysis and generating results. In the prototype system, the analysis core (OpenSees) is situated in the *Application Server*. Since OpenSees is a C++ application, the integration of OpenSees with Java Servlet server needs to be handled with special care. Although Java provides JNI (Java Native Interface) as an interface to procedures written in native programming language (C, C++, Fortran, etc.), accessing C++ applications from Java can be a challenging task. To keep the design modular and loosely coupled, the communication between Java applications and C++ programs is constructed in the data access system via a socket connection, instead of directly using JNI. Specific socket classes written in both Java and C++ are implemented to provide communication support between Java Servlets and OpenSees.

- A COTS database system is utilized as the **Data Server** for facilitating the storage and retrieval of selected analysis results. Oracle 8i [7] is employed as the database system in the prototype implementation. The communication between the *Application Server* and the *Database* is handled via the standard data access interfaces that Oracle 8i provides. In the original design of OpenSees, the *Channel* class is defined to facilitate the communication between core objects and remote processes (for details of *Channel* class, see McKenna [6]). The *Channel* class can be extended to include a new subclass *DB_Datastore*, which can be used to establish the communication between OpenSees core objects and the COTS database. The *DB_Datastore* class uses Open Database Connectivity (ODBC) to send and retrieve data between the OpenSees core objects and the database. Partial listing of the interface for the *DB_Datastore* class is shown in Fig. 3.

4. Data storage scheme

The usage of a database system in the data access system has two distinct phases. The first phase is during the finite element analysis of a model, in which certain selected analysis results are stored in the database. The second phase occurs during the postprocessing of a finite element analysis, where the analysis results are queried for the response of the analysis model. The goal of the data storage is to facilitate the data query, and the design of a data storage scheme needs to make the data query efficient and to minimize storage space. Rather than storing all the interim and final analysis results, the online data management system allows saving only selected analysis data in the database. That is, the user has the flexibility to specify storing only certain selected data during a structural analysis. All the other analysis results can be accessed through the analysis core with certain re-computation. The selective storage scheme can substantially reduce the data storage space without severely sacrificing the performance of accessing analysis results.

4.1. Selective data storage

A typical finite element analysis generates large volume of data. The analysis results can be saved and retrieved in two methods. One approach is to pre-define all the required data and save only those pre-defined data during the analysis. However, when analysis results other than the pre-defined ones are needed, a complete re-analysis is needed to generate those analysis results. For a nonlinear dynamic analysis of large structural models, the analysis needs to be restarted from scratch, which is an expensive process in terms of both processing time and storage requirement. The other approach is simply dumping all the interim and final analysis data into files, which are then utilized later to retrieve the required results as a postprocessing task. The drawbacks of this approach are the substantial amount of storage space and the potential poor performance due to the expensive search on the large data files.

There is an alternative to store only selected data, rater than storing all interim and final analysis results. Many approaches can be adopted for selecting the data to be stored during an analysis. The objective is to minimize the amount of storage space without severely sacrificing performance. For many commercial finite element analysis packages, such as ANSYS and ABAQUS, two types of output files can be created during an analysis. One type is a results file containing results for postprocessing. The results file is the primary medium for storing results in computer readable form. The results file can also be used as a convenient medium for importing analysis results into other postprocessing programs. Users are able to specify in the analysis input the kind of data to be saved in the results file. The other type of output file is a restart file containing results for continuing an analysis or for postprocessing. The restart file essentially stores the state of an analysis domain so that it can be used for subsequent continuation of an analysis. Users are allowed to specify the frequency at which results will be written to the restart file.

In the engineering data access system, these two types of data storage (results and restart) are also supported. The data access system allows the collection of certain information to be saved as the analysis progresses, e.g. the maximum nodal displacement at a node or the time history response of a nodal displacement. A *Recorder* class is introduced in OpenSees to facilitate the selective data storage during an analysis. The *Recorder* class can keep track of the progress of an analysis and output the users' pre-specified results. Details about the usage of `recorder` command have been described elsewhere by McKenna [21]. Besides the recording functionalities, the data access system also has the restart capability. Certain selected data are stored during the analysis that allows the analysis domain to be restored to a particular state. The selected data need to be sufficient for the re-computation during postprocessing. In the data access system, we use object serialization [22] to facilitate the restart function. Object serialization captures the state of an object and writes the state information in a persistent representation, for example in the form of a byte stream. Consider a *Truss* element as an example, its nodes, dimension, number of DOFs, length, area, and material properties can be saved in a file or a database system during an analysis. Based on these stored data, a copy of the *Truss* object can be restored, and the stiffness matrix of the *Truss* element can then be re-generated. The object serialization technique can be associated with other storage management strategies to further reduce the amount of storage space. As an example, a data storage strategy named *sampling at a specified interval* (SASI) can be applied to nonlinear incremental analyses to dramatically reduce the storage requirement.

The restart function introduced in the engineering data access system is different, however, from those supported by current commercial finite element programs (e.g. ANSYS, ABAQUS, etc.). The restart function in the data access system relies on object serialization, which allows the developer of each class to decide what kind of information needs to be saved. As long as a replica of an object can be recreated with the saved data, the developer of the class can freely manipulate the saved data. This decentralized development control provides great flexibility and extendibility to the developers, especially in a distributed and collaborative development environment. For most commercial finite element programs, the data saved in the restart file must conform to certain data format. Furthermore, the restart file of most commercial finite element programs is organized as a sequential file, which may make the data retrieval efficient. On the other hand, the restart data saved in the data access system is retrieved randomly – the state of a particular object is accessed through a key value. Therefore, a particular object or a sub-domain of the finite element domain can be easily restored without retrieving unnecessary data. Because COTS database systems generally have indexing capability to support key-based searching, the required data retrieval mechanism of the data access system is one reason that makes COTS database systems preferable to file systems.

In the data access system, a COTS database system is associated with the finite element analysis core to support data storage and data query. For a typical structural analysis, the analysis core stores selected data into the database. During the postprocessing phase, a request from a client for certain analysis results is submitted to the analysis core instead of directly querying the database. Upon receiving the request, the analysis core automatically queries the database for saved data to instantiate the required new objects. If necessary, these objects are initialized to restart the analysis to generate the requested results. Compared with re-performing the entire analysis to obtain the data that are not pre-defined, this re-computation is more efficient since only a small portion of the program is executed with the goal of fulfilling the request. As opposed to storing all the data needed to answer all queries, the selective storage strategy can significantly reduce the amount of data to be stored in the data management system.

4.2. Object Serialization

Ordinarily, an object lasts no longer than the program that creates it. In this context, persistence is the ability of an object to record its state so that the object can be reproduced in the future, even in another runtime environment. To provide persistence for objects, we can adopt a technique called object serialization, where the internal data structures of an object is mapped to a serialized representation that can be sent, stored, and retrieved by other applications. Through object serialization, the object can be shared outside the address space of an application by other application programs. A persistent object might store its state in a file or a database, which is then used to restore the object in a different runtime environment. The object serialization technique is one of the built-in features of Java and is used extensively in Java to support object storage and object transmission. There are currently three common forms of object serialization implementation in C++ [23]:

- **Java Model**: The Java serialization model stores all non-transient member data and functions for a serializable object by default. User can change the default behavior by overriding the object's `readObject()` and `writeObject()` methods, which specify the behaviors for serialization and deserialization of the object, respectively. This behavior can be emulated in C++ by ensuring that each serializable object implements two methods: one for serialization and another for deserialization.

- **HPC++ Model**: HPC++ [24] is a C++ library and a set of tools being developed by the *HPC++ Consortium* to support a standard model for portable parallel C++ programming. The serialization model was originally introduced in HPC++ to share objects in a network environment to facilitate parallel and distributed computing. Every serializable object declares a global function to be its *friend*. The runtime environment then uses this global function to access an object's internal state to serialize or deserialize it. In C++, a class can declare an individual function as a *friend*, and this *friend* function has access to the class' private members without itself being a member of that class.

- **Template Factory Model**: The template factory based serialization model is used in Java Beans [25], and this model can be emulated in C++. A template is defined for each object type by a template factory. For serialization, the runtime environment can invoke the serialization method `setX()` of each object to write the state of the object to a stream. For deserialization, the type of an object needs to be obtained from its byte stream representation first. A template of the object then can be created by the template factory based on the object type. Subsequently, the internal states of the object needs to be accessed from the stream with `getX()` method. Since a template of the object can be created based on its type and the template usually already includes some member data and methods, the `setX()` method only need to write the member data that are not defined in the template. This is the major difference between the template factory model and the Java model, whose `writeObject()` method accesses all the member data and methods.

In the engineering data access system, object serialization is supported via a technique that is similar to the Template Factory Model. In the OpenSees system, a *Domain* object is a container responsible for holding all the components of the finite element model, i.e. the *Node*, *Element*, *Constraint*, and *Load* objects [6]. For other object-oriented finite element programs, class that is used for this purpose may use different names, such as NAP [26], LocalDB [27], Partition [28], FE_Model [29], Model [30], and Domain [31]. The functionality of the *Domai*n class can be divided into two categories. One is responsible for adding components to the *Domain* object; and the other is for accessing the *Domain* components.

In the OpenSees system, all the *modeling* classes (Domain, Node, Element, Constraint, and Load, etc.), and *Numerical* classes (Matrix, Vector, ID, and Tensor, etc.) share a common superclass called *MovableObject*, which defines two member functions: *sendSelf()* and *recvSelf()*. The *sendSelf()* method is responsible for writing the state of the object so that the corresponding *recvSelf()* method can restore it. The typical mechanism for saving the object's state can be invoked by using a *Channel* object to send out all its member fields and other selected information. The *Channel* class is implemented in OpenSees to construct a communication channel with remote processes, which could be a remote application, a file system or a database system. For instance, the *DB_Datastore* class (shown in Fig. 3), which is a subclass of *Channel*, can be utilized to construct a communication channel with a database.

In the data access system, the data query processing involves restoring the domain state. The restoring process (deserialization) requires the domain state to be saved (serialization) first. During *Domain* serialization, the *Domain* object accesses all its contained components and invokes the corresponding *sendSelf()* methods on all the components to send out their states as a stream. The stream will then be piped to certain storage media (file system or database system) by the specified *Channel* object. For each object, the first field that it needs to send out is an integer *classTag*, which is a unique value used in OpenSees to identify the type of an object.

During *Domain* deserialization, the pre-stored data can be used to restore the *Domain* and its contained components. For each component, we first need to retrieve its *classTag* from the stream. The retrieved *classTag* then can be passed to the template factory, which is a class named *FEM_ObjectBroker*. The main method of the *FEM_ObjectBroker* class is

```
MovableObject* getObjectPtr(int classTag);
```

A template of the class corresponding to the *classTag* can be created by calling the constructor without arguments. The returned value is a pointer to an object with the generic *MovableObject* type. Since each object knows its own type (one of the basic features of object-oriented programming), the returned *MovableObject* can be further cast to create a specific template of the object. After a template for the object has been created, the remaining task of creating a replica of the object is to fill in the member fields. The replica of the object is created by calling the member method *recvSelf()*, which is responsible for reading the member fields from the associated *Channel* object. The restored component objects can then be added to the *Domain.* Fig. 4 lists partially the pseudo code for the process of invoking *recvSelf()* on *Domain* to restore its state to a specific step.

4.3. Sampling at a Specified Interval

We illustrate the usage of selective data storage strategies in this section by sampling the results at specified intervals (SASI). This data storage strategy can be applied for nonlinear incremental analysis. For numerical analysis of structures, formulation of equilibrium on the deformed geometry of a structure, together with nonlinear behavior of materials, will result in a system of nonlinear stiffness equations. One method for solving these equations is to approximate their non-linearity with a piecewise segmental fit [32]. For example, the single-step incremental method employs a strategy that is analogous to solving systems of linear or nonlinear differential equations by the Runge-Kutta methods. In general, the incremental analysis can be cast in the form

$$\{\Delta_i\} = \{\Delta_{i-1}\} + \{d\Delta_i\} \tag{1}$$

where $\{\Delta_{i-1}\}$ and $\{\Delta_i\}$ are the total displacements at the end of the previous and current load increments, respectively. The increment of unknown displacements $\{d\Delta_i\}$ is found in a single step by solving the linear system of equations

$$[K_i]\{d\Delta_i\} = \{dP_i\} \tag{2}$$

where $[K_i]$ and $\{dP_i\}$ represents the incremental stiffness and load respectively.

In contrast to the single-step schemes, the iterative methods need not use a single stiffness in each load increment. Instead, increments can be subdivided into a number of steps, each of which is a cycle in an iterative process aimed at satisfying the requirements of equilibrium within a specified tolerance. The displacement equation thus can be modified to

$$\{\Delta_i\} = \{\Delta_{i-1}\} + \sum_{j=1}^{m_i} \{d\Delta_i^j\} \tag{3}$$

where m_i is the number of iterative steps required in the ith load increment. In each step j, the unknown displacements are found by solving the linear system of equations

$$[K_i^{j-1}][d\Delta_i^j] = \{dP_i^j\} + \{R_i^{j-1}\} \tag{4}$$

where $[K_i^{j-1}]$ is the stiffness evaluated using the deformed geometry and corresponding element forces up to and including the previous iteration, and $[R_i^{j-1}]$ represents the imbalance between the existing external and internal forces. This unbalanced load vector can be calculated according to

$$\{R_i^{j-1}\} = \{P_i^{j-1}\} - \{F_i^{j-1}\} \tag{5}$$

where $\{P_i^{j-1}\}$ is the total external force applied and $\{F_i^{j-1}\}$ is a vector of net internal forces produced by summing the existing element end forces at each global degree of freedom. Note that in the above equations, the subscript is used to indicate a particular increment and the superscript represents an iterative step.

From the above equations, it can be seen that the state of the domain at a certain step is only dependent on the state of the domain at the immediate previous step. This is applicable for both incremental single-step methods and some of the incremental-iterative methods (such as Newton-Raphson scheme). Based on this observation, a discrete storage strategy can be applied to nonlinear structural analysis. More specifically, instead of storing all the analysis results, the state information of a nonlinear analysis is saved at a specified interval (e.g. every 10 steps or other appropriate number of steps, instead of every step). The saved state information needs to be sufficient to restore the domain to that particular step. As discussed earlier, object serialization can be used to guarantee this requirement.

During the postprocessing phase, the data requests are forwarded from the remote client site to the analysis core. After receiving the requests, the analysis core will search the database to find the closest sampled point that is less than or equal to the queried step. The core then fetches the data from the database to obtain the necessary state information for that step. These fetched data will be sufficient to restore the domain to that sampled step. After the domain restores itself to the required step, the core can progress itself to reach the queried time or incremental step. The details of this process are illustrated in the pseudo code shown in Fig. 5. Once the state of the queried step is restored, the data queries regarding the domain at that step can be processed by calling the corresponding member functions of the restored domain objects. Since the domain state is only saved at the sampled steps, the total storage space is dramatically reduced as opposed to saving the domain state at all the steps. Compared with restarting the analysis from the original step, the processing time needed by using *SASI* (i.e. restarting the analysis from a sampled step) can potentially be reduced significantly. The same strategy can also be designed for other types of analyses (such as for time dependent problems).

5. Data representation

In the data access system of the collaborative framework, data are organized internally within the FEA analysis core using an object-oriented model. Data saved in the COTS databases are represented in three basic data types: Matrix, Vector, and ID. Project management and version control capabilities are also supported by the system. For external data representation, XML (eXtensible Markup Language) [33] is chosen as the standard format for representing data in a platform independent manner. Since the internal and external data representations are different, certain data translation mechanism is needed.

5.1. Data modeling

In the collaborative framework, a relational COTS database system is used as the backend data management system. A relational database can be perceived by the users to be a collection of tables, with operators allowing a user to generate new tables and retrieve the data from the tables. The term *schema* often refers to a description of the tables and fields along with their relationships in a relational database system. An *entity* is any distinguishable object to be represented in the database. While at the conceptual level a user may perceive the database as a collection of tables, this does not mean that the data in the database is stored internally in tabular form. At the internal level, the data management system (DBMS) can choose the most suitable data structures necessary to store the data. This allows the DBMS to look after issues such as disk seek time, disk rotational latency, transfer time, page size, and data placement so that the system can respond to user requests much more efficiently than if the users were to implement the database directly using the file system.

The typical approach in using relational databases for FEA is to create a table for each type of object that needs to be stored (for example, Yang [16]). This approach, while straightforward, would require that a table be created for each type of object in the domain. Furthermore, in nonlinear analysis, two tables would have to be created, one for the geometry and the other for the time step related state information. Since data structures grow with the incorporation of new element and material types for finite element programs, the static schema definition of most DBMS is incompatible with the evolutionary nature of FEA software. Because of their static structure, commercial DBMS have difficulties in coping with changes and modification in the evolving design – inconsistencies could be introduced into the database and they are expensive to eliminate.

Since OpenSees is designed and implemented using C++, the internal data structure is organized in an object-oriented fashion. The data structure cannot be easily mapped into a relational database. As discussed in the last section, object serialization can be employed efficiently as linear stream to represent the internal state of an object. The linear stream can simply be a byte stream, or it can be a combination of matrix-type data, namely ID (array of integers), Vector (array of real numbers), and Matrix. The byte stream can be stored in the database as a CLOB in order to achieve good performance for data storage and searching. A CLOB is a built-in type that stores a *Character Large Object Block* as an entity in a database table. Two methods (*sendObj()* and *recvObj()*) are provided in the interface of *DB_Datastore* for the storage and retrieval of byte streams. The matrix-type data (ID, Vector, and Matrix), on the other hand, can be directly stored in a relational database. The corresponding methods for accessing the matrix-type data are also provided in *DB_Datastore* interface (for details, see Fig. 3). In the current implementation of the data access system, we focus on using the matrix-type data to represent and store the state of an object.

By using matrix-type data for storing object states, the database schema can be defined statically. The advantage of this approach is that new classes can be introduced without the creation of new tables in the relational database. The layer of abstraction provided by *DB_Datastore* can alleviate the burden of the FEA software developers, who in this case are typically finite element analysts, for learning database technologies. As long as the new classes (new element, new material types, etc.) follow the protocols of *sendSelf()* and *recvSelf()*, they can communicate with the database through the *DB_Datastore* object. The disadvantage of this approach is that no information regarding the meaning of the data will exist within the database. Therefore, users cannot query the database directly to obtain analysis results, e.g. the maximum stress in a particular type of elements. However, as discussed earlier, the data can be retrieved from the database by the objects in the core that placed the data there; that is, the semantic information are embedded in the objects themselves.

5.2. Project-based data representation

As shown in Fig. 1, a database is provided as the backend data storage to facilitate online data access. Since potentially many users can access the core server to perform structural analysis and to query the analysis results, a project management scheme is needed. The basic premise is that most researchers and engineers typically work independently, while sharing information necessary for collaboration. More importantly, they wish to retain control over the information they make accessible to other members [34]. In the prototype online data access system, a mechanism to perform version control and access control in order to cope with project evolution is implemented. The overall database schema is depicted in Fig. 6. The schema includes a user table and a project table. A user is identified by name and a project is identified by both its name and version. We use a hierarchical tree structure to maintain the version set of the projects. To simplify the design, each project has a primary user associated with it. This super-user has the privilege to modify the access control of a project. Only the authorized users who have the 'write' permission of a project will be allowed to make changes on the project and to perform online simulation with the analysis model. Other registered users have only 'read' access, in that any manipulation of analysis data is to be done a posteriori (for example, using other external programs such as MATLAB).

For the storage of nonlinear dynamic simulation results of a typical project, a hybrid storage strategy is utilized. As mentioned earlier, the state information saved in the database follows the *SASI* strategy. This strategy is very convenient and efficient for servicing the queries related to a certain time step, e.g. the displacement of Node 24 at time step 462. For obtaining a response time history, however, using the state information alone to reconstruct the domain will not be efficient. This is because a response time history requires the results from all time steps, and the state information from all time steps needs to be reconstructed. The performance in this case could be as expensive as a complete re-analysis. To alleviate the performance issue, the data access system has an option to allow the users to specify the response time histories of interests in the input file. During the nonlinear dynamic simulation, these pre-defined response time histories will be saved in files together with certain description information. These response time histories can then be accessed directly during postprocessing phase without involving expensive re-computation.

For the storage of *Domain* state information at the specified intervals, three tables are needed to store the basic data types. They are ID, Vector, and Matrix. Fig. 6 shows the schema design of the database and the relations among different tables. For ID, Vector, and Matrix tables, the attribute *projTag* identifies the project that an entry belongs to; *dbTag* is tag generated internally to identify the data entry; and *commitTag* flags the time step. Together, the set of attributes (*projTag, dbTag, commiTag*) forms a key for these relations. This set of attributes is also used as an index for the database table. An index on a set of attributes of a relation table is a data structure that makes it efficient to find those tuples that have a fixed value for the set of attributes. When a relation table is very large, it becomes expensive to scan all the tuples of a relation to find those tuples that match a given condition. In this case, an index usually helps with queries in which their attribute is compared with a constant. This is the most frequently used case for the database usage.

5.3. Data Representation in XML

Software applications collaborate by exchanging information. For example, a finite element program needs to be able to obtain an analysis model from CAD programs and send the analysis results to design tools. The lack of a reliable, simple and universally deployed data exchange model has long impaired effective interoperations among heterogeneous software applications. The integration of scientific and engineering software is usually a complex programming task. Achieving data interoperability is often referred to as ‘legacy integration’ or ‘legacy wrapping’, which has typically been addressed by ad-hoc means. There are several problems associated with this approach. First, every connection between two systems will most likely require custom programming. If many systems are involved, a lot of programming effort will be needed. Furthermore, if there are logic or data changes in one system, the interface will probably need to change – again, more need for programming. Finally, these interfaces are fragile: if some data are corrupted or parameters do not exactly match, unpredictable results can occur. Error handling and recovery are quite difficult with this approach.

XML can alleviate many of these programming problems. XML is a textual language quickly gaining popularity for data representation and exchange on the Web [35]. XML is a meta-markup language that consists of a set of rules for creating semantic tags used to describe data. An XML element is made up of a start tag, an end tag, and content in between. The start and end tags describe the content within the tags, which is considered the value of the element. In addition to tags and values, attributes are provided to annotate elements. Thus, XML files contain both data and structural information. More detailed description of XML can be found in Hunter et. al. [33].

In the data access system, XML is adopted as the external data representation for exchanging data between collaborating applications. XML format is not adopted for the internal communication in the finite element analysis program. The internal data of OpenSees is organized as matrix-type data (Matrix, Vector, and ID) and basic-type data (integer, real, and string, etc.), and a mechanism to translate between internal data and external XML representation is needed. The translation is achieved by adding two services: matrix services and XML packaging services. The matrix services are responsible for converting matrix-type data into an XML element, while the XML services can package both XML elements and basic-type data into XML files. The relation of these two types of services is shown in Fig. 7.

The translation between matrix-type data and XML elements is achieved by adding two member functions to Matrix, Vector, and ID classes to perform data conversion:

```
void XMLToObj(char* inputXML);
char* ObjToXML();
```

The function *XMLToObj()* is used to populate a matrix-type object with an input XML stream; the function *ObjToXML()* is responsible for converting the object member data into XML representation. In order to represent data efficiently, matrix-type entity sets can be divided into two categories: sparse matrices and full matrices. Fig. 8 shows the XML representation of a full matrix (for example, the stiffness matrix of a 2D truss element) and a sparse matrix (for example, the lumped mass matrix of a 2D truss element). Since Vector and ID are normally not sparse, they can be represented in a similar way as full matrix.

After matrix-type data are converted into XML elements, the next step is packaging them with other related information. This can be achieved by adding a new class *XMLService* to OpenSees, which is responsible for formatting and building XML documents, as well as interpreting and parsing input XML documents.

Two data models have been used in the data access system for XML representation. The relational model is used with tabular information, while the list model is defined for matrix-type entity sets. The relational model is different in implementation from the list model, because of the mechanism involved in locating a record of information. The tabular data essentially has two parts, one is the metadata that is the schema definition and the other is the content. An example of the tabular data is the displacement time history response of a node in nonlinear analysis. The list model is essentially provided for packaging all the related information into a single XML file. An example of the list model is the description of an element. Fig. 9 shows the typical XML representations for both tabular data and list data.

Since XML is a text format, and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the XML developers. The advantages of a text format are evident, and the disadvantages can usually be compensated at a different level. Programs like zip and gzip can compress files very well and very fast. Those programs are available for nearly all platforms (and are usually free). In addition, communication protocols such as HTTP/1.1 (the core protocol of the web) can compress data on the fly, thus saving bandwidth as effectively as a binary format.

6. Data query processing

For finite element programs, the postprocessing functions need to allow the recovery of analysis results and provide extensive graphical and numerical tools for gaining an understanding of results. In this sense, querying the analysis results is an important aspect and query languages need to be constructed to retrieve the analysis results. In the online data access system, a data query processing system is provided to support the interaction with both humans and other application programs. A data query language is also defined to facilitate data retrieval as well as invoking postprocessing functionalities. Through the query language, users can have uniform access to the analysis results by using a web-browser or other application programs.

6.1. Data query language

The data access system supports both programmed procedures and high-level query languages for accessing domain models and analysis results. A query language can be used to define, manipulate, and retrieve information from a database. For instance, for retrieving some specific result of an analysis, a query can be formed in the high-level and declarative query language that satisfies the specified syntax and conditions. In the data access system, a query language is provided to query the analysis result. This DQL (data query language) is defined in an easy-to-use manner that resembles natural English language. The DQL is capable of querying the analysis results together with invoking certain postprocessing computation. Combining general query language constructs with domain-related representations provides a more problem-oriented communication [36]. The defined DQL and the programmed procedures have at least two features:

- It provides a unifying data query language. No matter what kind of form the data is presented (whether a relation or a matrix), the data is treated in the same way. It is also possible to make query on specific entries in a table or individual elements of a matrix.

- It provides language with the same syntax for both terminal users (from command lines) and for those using them from a programmed procedure. This leads to the ease of communication between client and server, and can save programming efforts when linking the data access system with other application programs.

As discussed earlier, a hybrid storage strategy is utilized for storing nonlinear dynamic simulation results. For different type of stored data (results regarding a certain time step or time history responses), different query commands are needed and different actions are taken. Several commonly used features of the DQL are illustrated below.

Queries related to a particular time step:

First, we will illustrate the queries related to a certain step. In order to query the data related to a certain time step, the state of the domain needs to be restored to that time step. For example, we can use command `RESTORE 462`, which will trigger the function `Domain::convertToState()` (shown in Fig. 5), to restore domain state to time step 462.

After the domain has been initialized to certain time step, queries can be issued for detailed information. As an example, we query the displacement from node number 4,

```
SELECT disp FROM node=4;
```

The analysis result can also be queried from Element, Constraint, and Load. For example,

```
SELECT tangentStiff FROM element=2;
```

returns the tangent stiffness matrix of element 2.

Besides the general queries, two wildcards are provided. One is the wildcard '*' that represents *all values*. For instance, if we want to obtain the displacement from all the nodes, we can use

```
SELECT disp FROM node=*;
```

The other wildcard '?' can be used on certain object to find out what kind of queries it can support. For example, the following query

`SELECT ? FROM node=1;` returns

```
Node 1:: numDOF crds disp vel accel load mass *
```

Another class of operations is called *aggregation*. By aggregation, we mean an operation that forms a single value from the list of values appearing in a column of a database table. In the current implementation, five operators are provided that apply to a column of the relation table and produce a summary or aggregation of that column. These operators are:

`SUM`, the sum of the values in the column;

`AVG`, the average of values in the column;

`MIN`, the least value in the column;

`MAX`, the greatest value in the column;

`COUNT`, the number of values.

Queries of time history responses:

The second type of queries is used to access the pre-defined analysis results, especially the time history responses. The queried time history responses can be saved into files in the client site. These files can then be processed to generate graphical representation. For instance, if we want to save the displacement time history of a particular node for a nonlinear analysis, the following query can be issued to the server

```
SELECT time disp FROM node=1 AND dof=1
SAVEAS node1.out;
```

If the data are pre-defined in the input file and saved during the analysis phase, the query can return the corresponding saved analysis results. Otherwise, a complete re-computation is triggered to generate the requested time history response.

6.2. Data Query Interfaces

The collaborative framework can offer users access to the analysis core, as well as the associated supporting services via the Internet. One of the supporting services is to query analysis results. Users can compile their query in the client site and then submit it to the central server. After the server finishes the processing, queried results will return to the users in a pre-defined XML format. It is up to the program in the client site to interpret the data and present in a specific format desirable to the users. In the prototype system, two types of data query interfaces at the client site are provided. They are web-based interface and MATLAB-based interface. This client-server computing environment forms a complete computing framework with a very distinct division of responsibilities. One benefit of this model is the transparency of software services. From a user's perspective, the user is dealing with a single service from a single point-of-entry – even though the actual data may be saved in the database or re-generated by the analysis core.

For the data access system, a standard World Wide Web browser is employed to provide the user interaction with the core platform. Although the use of a web browser is not mandatory for the functionalities of the data access system, using a standard browser interface leverages the most widely available Internet environment, as well as being a convenient means of quick prototyping. Fig. 10 depicts the interaction between the web-based client and the data access server. A typical data query transaction starts with the user supplying his/her data query intention in a web-based form. After the web-server receives the submitted form, it will extract the useful information and packaging it into a command that conforms to the syntax of the DQL. Then the command will be issued to the core analysis server to trigger the query of certain data from the database and to perform some re-computation by the analysis core. After the queried data is generated, it will be sent back to the client site and presented to the user as a web page.

The web-based client is convenient and straightforward for the cases when the volume of the queried data is small. When the data volume is big, especially if some postprocessing is needed on the data, the direct usage of web-based client can bear some inconvenience. All too often the queried analysis results need to be downloaded from the server as a file, and then put manually into another program to perform post processing, e.g. a spreadsheet. For example, if we want to plot a time history response of a certain node after a dynamic analysis, we might have to download the response in a data file and then use MATLAB, Excel, or other software packages to generate the graphical representation. It would be more convenient to directly utilize some popular application software packages to enhance the interaction between client and server. In our prototype system, a MATLAB-based user interface is available to take advantage of the flexibility and graphical processing power of MATLAB. In the implementation, some extra functions are added to the standard MATLAB in order to handle the network communication and data processing. Theses add-on functions can be directly invoked from either the MATLAB prompt or a MABLAB-based graphical user interface.

7. Examples

A prototype of the data access and project management system is implemented by using Sun workstations as the hardware platform. The experiments were performed on a Sun Ultra60 workstation with 256Mbytes of memory and two 450MHz Sun UltraSPARC processors. The finite element analysis core is based on OpenSees, and the employed database system is Oracle 8i. Apache HTTP server is served as the web server, and Apache Tomcat 4.0 is utilized as the Java Servlet server. MATLAB 6.0 and a standard web-browser are employed as the user interfaces for accessing the analysis results and project-related information.

7.1. Example 1: An Eighteen Story One Bay Frame

The first example is an eighteen story two-dimensional one bay frame. The story heights are all 12 feet and the span is 24 feet. A nonlinear dynamic analysis is performed on the model using the 1994 Northridge earthquake recorded at the Saticoy St. station, Van Nuys, California. Newton-Raphson algorithm is utilized for performing the structural analysis and SASI strategy is chosen to store the selected analysis results. In the example, the specified interval is ten time steps, which is specified in the input file. Details of the model and the analysis have been described in Peng and Law [3]. Besides the saved domain states, the time history displacement values of each node are saved in files by using the `recorder` command in the input file. The usage of the `recorder` command can be found in [21]. After the analysis, the results regarding a certain time step can be queried by using DQL commands. The following illustrate example usage of some of the DQL commands. We use **C:** for the command and **R:** for the queried results.

C: `RESTORE 462`

This command is used to restore the *Domain* state to the time step 462. The command first triggers the analysis core to fetch from the database the saved *Domain* state of the time step 460, which is the closest time step stored before the requested step. The core then progresses itself to reach the time step 462 by using Newton-Raphson algorithm. After the *Domain* has been initialized to the step of 462, the wildcard '?' can be used to find what kind of queries that node 1 (which is the left node on the 18th floor) can support

C: `SELECT ? FROM node=1;`

```
R: Node 1:: numDOF crds disp vel accel load trialDisp trialVel
   trialAccel mass *
```

The attribute information of node 1 can be queried, for example

```
C: SELECT disp FROM node=1;
R: Node 1:: disp= -7.42716  0.04044
```

The analysis result can also be queried from Element, Constraint, and Load. For instance, we can query the information related to element 19, which is the left column on the 18th floor.

```
C: SELECT ? FROM element=19;
R: ElasticBeam2D 19:: connectedNodes A E I L tangentStiff
   secantStiff mass damp
C: SELECT L E FROM element=19;
R: ElasticBeam2D 19:: L=144 E= 29000
```

As mentioned earlier, five *aggregation* operators are provided to produce summary or aggregation information. For instance, the following command produces the maximum displacement of nodes. Note that both positive and negative maximum values are presented.

```
C: SELECT MAX(disp) FROM node=*;
R: MAX(disp)::
   Node 1: -7.42716
   Node 21: 4.93986
```

We can also use a DQL command to query the time history response. For instance, if we want to save the displacement time history response of Node 1, the following query can be issued to the server

```
SELECT time disp FROM node=1 AND dof=1
SAVEAS node1.out;
```

After the execution of the command, the displacement time history response of Node 1 is saved in a file named node1.out.

7.2. Example 2: Humboldt Bay Middle Channel Bridge

The second example is an ongoing research effort within the Pacific Earthquake Engineering Research (PEER) center to investigate the seismic performance of the Humboldt Bay Middle Channel Bridge. The Humboldt Bay Middle Channel Bridge is located at Eureka in northern California, and the bridge is a 330 meters long, 9-span composite structure with precast and prestressed concrete I-girders and cast-in-place concrete slabs to provide continuity. It is supported on eight pile groups, each of which consists of 5 to 16 prestressed concrete piles. The river channel has an average slope from the banks to the center of about 7% (4 degrees). The foundation soil is mainly composed of dense fine-to-medium sand (SP/SM), organic silt (OL), and stiff clay layers. In addition, thin layers of loose and soft clay (OL/SM) are located near the ground surface. A two-dimensional nonlinear model of the Middle Channel Bridge, including the superstructure, piers, and supporting piles, was developed using OpenSees as shown in Fig. 11 [37]. The bridge piers are modeled using 2-D nonlinear material fiber beam-column elements and the abutment joints are modeled using zero-length elasto-plastic gap-hook elements. A four-node quadrilateral element is used to discretize the soil domain. The soil below the water table is modeled as an undrained material, and the soil above as dry.

To conduct probability analysis, approximately sixty ground motion records are to be applied to the bridge model for damage simulation. The ground motion records are divided into three hazardous levels based on their probability of occurrence, which are 2%, 10%, and 50% in fifty years respectively. Since the bridge will perform differently under each ground motion level, the projects can be grouped according to the applied ground motion records. Fig. 12 is a list of some of the Humboldt Bay Bridge projects. When using the project management developed in this work, the web page is a single point-of-entry for all the project related background information, documents, messages, and simulation results. The detailed information of a particular project can be accessed by clicking on the project name, which is a hyperlink to the project website.

We will use project X1 (see Fig. 12) as an illustration. The finite element model used in this project has 456 beam-column elements, 16 zero-length elements and 3459 quadrilateral elements. The ground motion applied to this project is a near-field strong motion, the 1994 Northridge earthquake recorded at the Rinaldi station (PGA = 0.89g, PGV = 1.85 m/sec, PGD = 0.60 m), with a probability of 2% occurrence in fifty years. A nonlinear dynamic analysis is conducted on the model with the input ground motion. The analysis was conduced under three different conditions: without any domain state storage, using Oracle database to save the domain states at every 20 time steps, and using file system to save the domain states at every 20 time steps. Table 1 shows the solution time for the nonlinear dynamic analysis. Since the model is fairly large and some *expensive* elements (fiber element) and materials (nonlinear) are used in the model, the nonlinear dynamic analysis requires extensive computation time. As shown in Table 1, the usage of Oracle database system and file system to store the selected domain states imposes some overhead and further reduces the performance.

Although there are performance penalties for using database or file system to save selected domain states during a nonlinear dynamic analysis, using the data storage can improve the performance of re-computation during the postprocessing phase. Table 2 lists some solution time for re-computation that involves restoring the domain to certain time steps. If there are no domain states archived during the analysis phase, we have to start the analysis again from the scratch. On the other hand, with the selected domain states saved in the database or file system, we can restore the domain to certain stored state and then progress the domain to the requested time step. For example, in order to restore the domain state to the time step 105, we can retrieve the domain state at time step 100 and then progress the domain to time step 105 by using Newton-Raphson scheme. The solution time shown in Table 2 clearly demonstrated that the usage of data storage dramatically improves the performance of re-computation. The storage space required for this analysis model is roughly 230MB for storing the domain state per time step. Because of the large amount of data to be stored, seperate file is created for the domain state of each saved time step. The experimental results also showed that the usage of Oracle database is generally more efficient than the direct usage of file systems for large structural models.

As mentioned earlier, a hybrid storage strategy is used to save some selected analysis results in the database during the nonlinear dynamic simulation. The saved analysis results include both Domain state information at sampled time steps and user pre-defined response time histories. To query results regarding a certain time step, the DQL commands are similar to what have been used for the previous example; and this process involves restoring the domain to that time step together with certain re-computation. For obtaining a pre-defined response time history, on the other hand, no re-computation is needed. Fig. 13 shows a typical session of using a web-based user interface, where all the pre-defined response time histories are listed, and certain response results can be searched and downloaded.

Besides the web-based user interface, a MATLAB-based user interface is also available to take advantage of the mathematical manipulation and graphic display capabilities of MATLAB. Some functions are added to the standard MATLAB for handling the network communication and data processing. These commands can be executed from either the standard MATLAB prompt or the MATLAB-based graphical interfaces. We can issue the command `submitmodel humboldtX1.tcl` to submit the input file to the analysis core server for performing the online simulation. After the analysis, the command `queryresult` can be issued to bring up an interactive window. The user then enters DQL commands in this window to query the analysis results related to a certain time step. To access the pre-defined response time histories, the command `listResults` can be used to generate the list of response time history files (shown in Fig. 14(a)). To generate a graphical representation of a particular response time history, two steps are needed: one for downloading the file and the other for plotting. For example, we can issue the command `getFile press1315_2.out` to download the response time history file and `res2Dplot(‘press1315_2.out’)` to invoke the plotting of the results. The plot is shown in Fig. 14(b).

8. Conclusions

Scientific and engineering database systems have several special requirements compared with their business counterparts. The dataflow of a finite element analysis program is tightly coupled with expensive numerical computation. This is one of the main reasons for the lack of a generic purpose data management system for finite element programs. When a number of programs are required for engineering simulation or design, the absence of standardization and the lack of coordination among software developers can result in difficulty in data communication from one program to another. The result is often manual transfer of data with laborious efforts, time delay, and potential error. In the effort of trying to alleviate some of the problems, we have introduced a data access system for a finite element structural analysis program. The main design principle of the system is to separate data access and data storage from data processing, so that each part of the system can be designed and implemented separately. By adopting a multi-tiered modular infrastructure, each component of the system can be added or updated without substantial amount of modification to the existing system. By storing abstract data types (ID, Vector, and Matrix) into the database, it is not necessary to re-implement low-level dedicated data structures or to re-define new database tables for each added new element or other components. By introducing a standard data representation using XML format, the data access system can easily exchange data with other application programs. The utilization of standard query language and popular query interfaces, as well as the deployment of the Internet for delivering data, is another factor that makes the system flexible and extendible. Although this work has focused on engineering data access system for a finite element program, the design principles and techniques can be applied to other similar types of engineering and scientific applications.

In the prototype data access system, the performance may be a concern. This is partly because of the unstable performance of the Internet, and partly due to the design decision of sacrificing certain degree of performance for better flexibility and extendibility. However, compared with the direct usage of file systems, using relational database systems normally improves the overall performance of the system. By utilizing the selective storage strategy, especially SASI, the amount of storage space in our system is substantially smaller than the storage requirement of simply dumping all the analysis data into files. Compared to the traditional way of redoing the entire analysis to obtain the results that are not pre-defined, the re-computation technique used in the data access system could be more efficient because only a small portion of the program is executed with the goal to fulfill the query request. While the performance issue does exist, it may be alleviated by efficient optimization of generated code, and by indexing techniques. The database indexing techniques have been used in the prototype data access system to improve the performance of data query.

For the prototype implementation and testing in this research work, we have utilized Oracle database system as the backend data server. The experimental results clearly demonstrated that a COTS database system can facilitate the data management for a FEA program, and improve the performance of certain types of queries including queries related to a particular time step of a nonlinear dynamic analysis. Because the design is flexible and general, other types of database systems, e.g. MySQL [38], BerkeleyDB [39], or Microsoft Access [40], can be easily employed to provide data storage for the data access system.

The project management scheme developed in the collaborative framework allows the usage of a database system to manage the information related to projects. The actual project data is stored in distributed machines. Certain access control and revision control capabilities are provided in the project management system. Further work needs to be done to improve the flexibility and power of the project management system. New development may include an automatic notification mechanism to acknowledge and notify project participants whenever changes are made to a particular project.

Acknowledgements

The authors would like to thank Dr. Frank McKenna and Prof. Gregory L. Fenves of UC, Berkeley for their collaboration and support of this research. The authors are also grateful to Dr. Zhaohui Yang, Mr. Yuyi Zhang, Prof. Ahmed Elgamal, and Prof. Joel Conte for providing the Humboldt Bay Bridge example and valuable suggestions.

This work has been supported in part by the Pacific Earthquake Engineering Research Center through the Earthquake Engineering Research Centers Program of the National Science Foundation under Award number EEC-9701568, and in part by NSF Grant Number CMS-

0084530. The *Technology for Education 2000* equipment grant from Intel Corporation provides the computers employed in this research.

References

[1] Mackie RI. Using objects to handle complexity in finite element software. Engineering with Computers 1997;13(2):99-111.

[2] van Engelen R, Gallivan K, Gupta G and Cybenko G. XML-RPC agents for distributed scientific computing, Proceedings of IMACS'2000: the IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, Lausanne, Switzerland, 2000.

[3] Peng J and Law KH. A prototype software framework for Internet-enabled collaborative development of a structural analysis program. Engineering with Computers 2002;18(1):38-49.

[4] Peng J and Law KH. Framework for collaborative structural analysis software development, Proceedings of the Structural Congress & Expositions ASCE, Philadelphia, PA, 2000.

[5] Peng J, McKenna F, Fenves GL and Law KH. An open collaborative model for development of finite element program, Proceedings of the Eighth International Conference on Computing in Civil and Building Engineering (ICCCBE-VIII), Palo Alto, CA, 2000;1309-16.

[6] McKenna F. Object-oriented finite element programming: frameworks for analysis, algorithm and parallel computing. PhD Thesis. Department of Civil and Environmental Engineering, University of California at Berkeley, 1997.

[7] Kyte T. Expert one on one: Oracle, Chicago, IL: Wrox Press, 2001.

[8] Tuel WG and Berry CJ. Data bases for geographic facility design and analysis, Proceedings of ASCE Conference on Computing in Civil Engineering, Atlanta, GA, 1978;449-64.

[9] Lopez LA, Dodds RH, Rehak DR and Urzua JL. Application of data management to structures, Proceedings of ASCE Conference on Computing in Civil Engineering, Atlanta, GA., 1978;477-88.

[10] Felippa CA. Database management in scientific computing -- I. general description. Computers and Structures 1979;10(1-2):53-61.

[11] Felippa CA. Database management in scientific computing -- II. data structures and program architecture. Computers and Structures 1980;12(1):131-46.

[12] Felippa CA. Database management in scientific computing -- III. implementation, Proceedings of Symposium on Trends and Advances in Structural Mechanics, Washington, DC, 1982.

[13] Blackburn CL, Storaasli OO and Fulton RE. The role and application of data base management in integrated computer-aided design. Journal of Aircraft 1983;20(8):717-25.

[14] Anumba CJ. Data structures and DBMS for computer-aided design systems. Advances in Engineering Software 1996;25(2-3):123-9.

[15] Rajan SD and Bhatti MA. SADDLE: A computer-aided structural analysis and dynamic design language -- part II. database management system. Computers and Structures 1986;22(2):205-12.

[16] Yang X. Database design method for finite element analysis. Computers and Structures 1992;44(4):911-4.

[17] Carey MJ, DeWitt DJ, Graefe G, Haight DM, Richardson JE, Schuh DT, Skekita EJ and Vandenberg SL. The EXODUS extensible DBMS project: an overview. In Zdonik SB and

Maier D, editors, Readings in object-oriented database systems, data management series, San Mateo, CA: Morgan Kaufmann, 1990.

[18] Nagel RN, Braithwaite WW and Kennicott PR. Initial graphics exchange specification IGES, version 1.0, Washington, DC: National Bureau of Standards, 1980.

[19] Fishwick PA and Blackburn CL. Managing engineering data bases: the relational approach. Computers in Mechanical Engineering 1983;1(3):8-16.

[20] Chandra S. Information extraction and qualitative descriptions from dynamic simulation data. Computers and Structures 1998;69(6):757-66.

[21] McKenna F and Fenves GL. The OpenSees command language manual. http://opensees.berkeley.edu/OpenSees/OpenSeesCommands.pdf, 2001.

[22] Breg F and Polychronopoulos CD. Java virtual machine support for object serialization, Proceedings of the ISCOPE Conference on ACM 2001 Java Grande, Palo Alto, CA, 2001;173-80.

[23] Slominski A, Govindaraju M, Gannon D and Bramley R. Design of an XML based interoperable RMI system: SoapRMI C++/Java 1.1, Proceedings of PDPTA'2001: the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 2001;1661-7.

[24] Diwan S. Open HPC++: An open programming environment for high-performance distributed applications. PhD thesis. Computer Science Department, Indiana University, 1999.

[25] Lunney TF and McCaughey AJ. Component based distributed systems - CORBA and EJB in context. Computer Physics Communications 2000;127(2):207-14.

[26] Forde BWR, Foschi RO and Stiemer SF. Object-oriented finite element analysis. Computers and Structures 1990;34(3):355-74.

[27] Miller GR. An object-oriented approach to structural analysis and design. Computers and Structures 1991;40(1):75-82.

[28] Rucki MD and Miller GR. Algorithmic framework for flexible finite element-based structural modeling. Computer Methods in Applied Mechanics and Engineering 1996;136(3-4):363-84.

[29] Mackie RI. Object-oriented methods -- finite element programming and engineering software design, Proceedings of the 6th International Conference on Computing in Civil and Building Engineering (ICCCBE-VI), Berlin, Germany, 1995;133-8.

[30] Archer GC. Object-oriented finite element analysis. PhD thesis. Department of Civil and Environmental Engineering, University of California at Berkeley, 1996.

[31] Zimmermann T, Dubois-Pelerin Y and Bomme P. Object-oriented finite element programming: I. governing principles. Computer Methods in Applied Mechanics and Engineering 1992;98(2):291-303.

[32] McGuire W, Gallagher RH and Ziemian RD. Matrix structural analysis, New York, NY: John Wiley & Sons, Inc., 2000.

[33] Hunter D, Rafter J, Pinnock J, Dix C, Cagle K and Kovack R. Beginning XML, Chicago, IL: Wrox Press Inc, 2001.

[34] Krishnamurthy K. A data management model for change control in collaborative design environments. PhD thesis. Department of Civil and Environmental Engineering, Stanford University, 1996.

[35] Goldman R, McHugh J and Widom J. From semistructured data to XML: Migrating the Lore data model and query language, Proceedings of WebDB '99: the Second International Workshop on the Web and Databases, Philadelphia, PA, 1999;25-30.

[36] Orsborn K. Applying next generation object-oriented DBMS for finite element analysis, Proceedings of the first International Conference on Applications of Databases (ADB'94), Vadstena, Sweden, 1994;215-33.

[37] Conte JP, Elgamal A, Yang Z, Zhang Y, Acero G and Seible F. Nonlinear seismic analysis of a bridge ground system, Proceedings of the 15th ASCE Engineering Mechanics Conference, New York, NY, 2002.

[38] DuBois P and Widenius M. MySQL, Indianapolis, IN: New Riders Publishing, 1999.

[39] Olson MA, Bostic K and Seltzer M. Berkeley DB, Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, Monterey, CA, 1999;183-92.

[40] Andersen V. Access 2000: The complete reference, Berkeley, CA: Osborne McGraw-Hill, 1999.