6 Internet-Enabled Software Platform

6.1 ABSTRACT

This chapter describes the development of an Internet-enabled open collaborative platform for OpenSees. The new platform is intended to serve as a common tool for which researchers can build, test and incorporate new developments. Furthermore, the new platform would allow users to have easy access to the nonlinear analysis platform via the Internet. Users can have direct access to OpenSees and the analysis results via a web-browser or other application programs. Last but not least, a database interface has also been developed to provide a persistent storage for the interim and selected analysis results.

6.2 INTRODUCTION

It is well recognized that a significant gap exists in the state-of-the-art computing methodologies and the state-of-practice in structural engineering analysis programs. Most prevailing structural analysis programs bundle all the procedures and program kernels into software packages that are developed by individual organizations. As technologies advance, these software packages need to be able to accommodate new advances in element and material formation, solution strategies, and computing environments. Extending and upgrading these programs to incorporate new developments is a difficult process and more importantly, there is no easy way to link customized components developed by users and researchers separately outside the organization.

The development of OpenSees is facing the same challenge. At the early stage of OpenSees, it was developed and tested within a small group at PEER center. The communication and cooperation among the researchers could be achieved rather easily. However, as the platform begins to gain acceptance and popularity, many people are involved in the development with different perspectives and different focuses. There are users whose main purpose is to use OpenSees as a structural analysis tool. There are core developers who are intending to

incorporate new developments and to update the analysis core. There are solution and analysis strategies developers as well as researchers whose main focus is developing new element and material technologies. Since the development is concurrent and incremental, traditional waterfall software development approach cannot meet the requirements. The management of developers and source code becomes important for the success of continuing development.

With the maturation of information and communication technologies, the concept of building collaborative systems to distribute the services over the Internet is becoming a reality (Han et al. 1999). Following this idea, we have developed an open collaborative framework for the continuing development of OpenSees (Peng and Law 2000a, Peng et al. 2000b). A collaborative system is one where multiple users or agents engage in a shared activity, usually from remote locations. Unlike the development of traditional packaged structural analysis programs, the collaborative framework can potentially reduce the overhead of continuous upgrade and extension. Users and engineers can select appropriate services and can easily replace a specific module if a superior module becomes available. Developers and researchers can concentrate on developing components and can easily integrate their component to the core through a *plug-and-play* environment. In some sense, the open collaborative system paradigm can be seen as analogous to the open source OS concept (i.e. Linux) that has resulted in some high quality software.

6.3 ARCHITECTURE OF COLLABORATIVE MODEL

The overall system architecture of the Internet-enabled collaborative framework is schematically depicted in Fig. 1. In this model, OpenSees is running in a central server as a finite element compute engine. A compute engine is a remote software program that takes tasks from clients, runs them and returns the result. In this part, parallel and distributed computing environment can also be deployed to perform large-scale engineering simulations (Santiago 1996, McKenna and Fenves 2000).

In this collaborative system, users play the role of clients to the central compute engine. They can have direct or remote access (one such avenue is the Internet (Han et al. 1999)) to the core program through a graphical user interface or an interpreter. The users can utilize advanced and appropriate developments (element types, efficient solution methods, and analysis strategies) contributed by other developers that were incorporated into the platform. For element technology developers, a standard interface/wrapper is defined for communicating the element(s) with the analysis core. The element(s) can be written in languages such as C, Fortran, C++ and/or Java. If the developer and the system administrator agree, the new element(s) can be migrated into the analysis core and become part of the static element library. Moreover, the developer can also choose to be an on-line element service provider. In this case, the element(s) can be registered to the core and can be accessed remotely over the network.

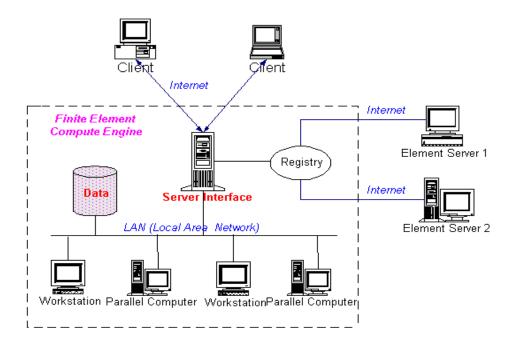


FIG. 1. Collaborative System Architecture for Finite Element Program

As shown in Fig. 2, the finite element compute engine is designed in a modular manner. The *Analysis Core* module is the part that most PEER researchers are working on. New element and material technologies are brought into this module to enhance the functionality of the core. The *User-Interaction Interface* module is deployed to provide web-based interface to the users and developers. The *Registration and Naming Service* is provided for on-line services to register to the core so that these services can be found during analysis. The *Distributed Element Service* module is provided for remote access to elements resided in different sites. The *Dynamic Linked Element Service* is implemented to provide a flexible way of dynamically binding elements to the core in real time. Last but not least, the *Database Interface* module can take advantage of a commercial database to provide efficient data access and to enable flexible post-processing. The details of each module will be further explained in the following sections.

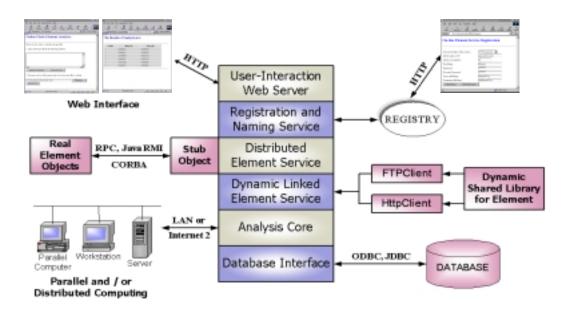


FIG. 2. Collaborative System Modules

Since the proposed open collaborative system relies on the aggregate behavior of loosely coupled subsystems, *component-based* design and modeling can be used to facilitate the concurrent development process. The Internet has introduced a transport mechanism that allows various disparate components to interact with each other to provide more complex, complete system behavior (Hopkins 2000). We can now look at software development from a higher level of abstraction, one that treats the individual components as the target platforms. The interactions between them form the dynamic behavior of the system. By utilizing a component-based approach, the application is easy to build and is easy to modify and extend.

The Internet-enabled collaborative model can provide greater flexibility and extensibility than traditional structural analysis programs, which are typically individually packaged. The mechanics of the collaborative model is illustrated in Fig. 3. In the framework, the users build their structural model by using a web-based model-building service on the client site. The model then can be sent to the analysis core by using the Internet as a communication channel. Upon receiving the analysis model, the core server performs the analysis in a collaborative manner. During the analysis, some elements can be obtained locally from the core static element library. In order to find other required elements that do not exist in the local element library, the *registry* will be queried to find the location of the on-line element services, which have already been pre-

the analysis core to perform the structural analysis. After the analysis is completed, the results will be returned to the user by generating a dynamic web page in the user's web browser.

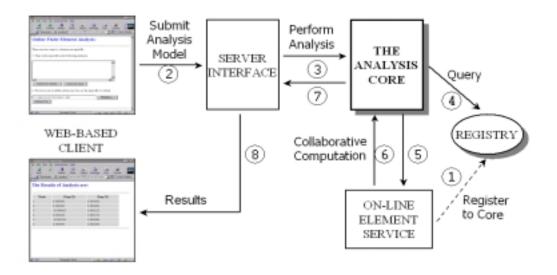


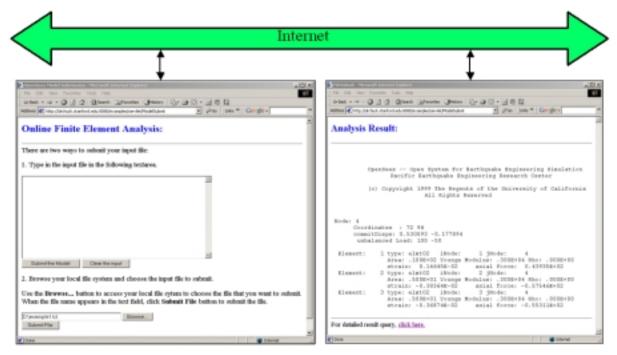
FIG. 3. Mechanics of the Collaborative Model

6.4 USER INTERACTION INTERFACE

As indicated in Fig. 3, the collaborative framework can offer access to OpenSees, as well as the associated support services, to users and researchers via the Internet. One benefit of this model is the transparency of software services. From a user's perspective, he or she is dealing with a single service from a single point of contact – even though the actual structural simulation can be performed in a distributed and collaborative manner. The other benefit is that this framework promises to widen the reach of OpenSees to researchers and practitioners. Users can easily access the software platform without the associated cost and maintenance challenges of personal computing.

A user who has a copy of OpenSees installed in his/her personal computer can conduct a simulation with a scripting language that has been extended to incorporate the features of OpenSees. The scripting language is named Tcl, and it has many features for dealing with variables, expressions, loops, data structures, and input/output, that are useful for doing a simulation. There are two ways to execute Tcl script. One is the interaction mode where users can enter commands at the Tcl prompt. The other is the batch mode where users can save their analysis model in a file and execute OpenSees with this file as input.

In our web-based user interface, the two modes of executing Tcl script are also provided. Users can directly interact with the server by submitting Tcl script or they can compile a Tcl script file by using their favorite text editor and then submit the input file to the central server. After the server performs the simulation, the results can be returned to the users by generating a dynamic web page in their browser. If the users prefer to have persistent simulation result, it can also be downloaded in output file format. Fig. 4 shows the web pages for both file submission and the simulation result generation.



File Submission for Analysis Model

Analysis Results

FIG. 4. Web Pages Generated in the Client Site

Besides the illustrated web-based user interface, a Matlab-based user interface is also provided to take advantage of the flexibility and graphical processing power of Matlab, so that Matlab can be utilized to perform the post-procssing of a model analysis. In our implementation, some extra functions are added to standard Matlab to handle the network communication and data processing. These add-on functions can be directly invocated from Matlab prompt. The usage of Matlab-based user interface will be shown in the *Examples* section

6.5 DISTRIBUTED ELEMENT SERVICE

Most people interacting with the central server are end users whose primary concern is doing finite element simulation. However, there are also some developers who are interested in bringing new element and material technologies into the core server. For the new element developers, a standard interface/wrapper is defined for communicating the element with the object-oriented analysis core. To introduce the new element into the analysis core generally composes of creating subclasses of *Element* class. The common interface for *Element* super-class is defined in OpenSees kernel. After the development process is finished, the new element can be shifted into the core platform and become part of the static element library.

In addition to contributing the element to the analysis core directly, the developer can also choose to be an on-line element service provider. In this case, the actual computation code resides in the service provider's site and it runs as a compute engine, as shown in Fig. 5. Whenever the core needs certain element data, for example the stiffness matrices, the core will request the service provider through a sequence of remote procedure calls. The meaningful computation is performed at the service provider's site and the requested data can be sent back to the server after the computation.

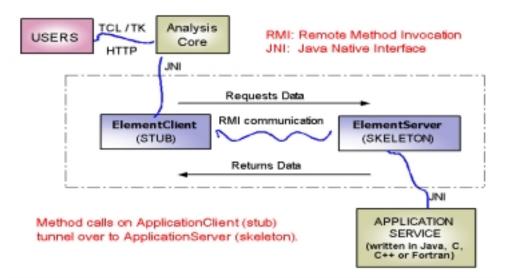


FIG. 5. Distributed Element Service

The essential requirements in a distributed object system are the ability to create or invoke objects on a remote host or process, and interact with them as if they were objects within our own process. It seems logical that we would need some kind of message protocol for sending requests to remote agents to create new objects, to invoke methods on these objects, and to delete the objects when we are done with them. In the prototype implementation, Java's *Remote Method Invocation (RMI)* is used to handle the network communication. RMI enables a program in one Java Virtual Machine (JVM) to make method calls on an object located on a remote server machine. RMI gives the programmer the ability to distribute computing across a networked environment and thus allows a task to be performed on the machine most appropriate for the task. The *skeleton*, which is the object at the server site, receives method invocation requests from the client. It then makes a call to the actual object implemented on the server. The *stub* is the client's proxy representing the remote object. *Stubs* define all of the interfaces that the remote object supports. Fig.6 shows the interaction diagram for a typical distributed element service.

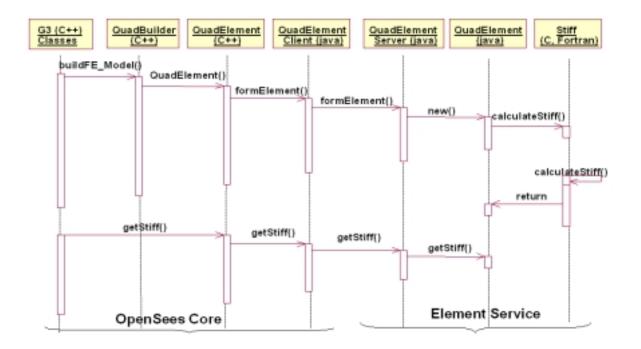


FIG. 6. Interaction Diagram for Distributed Element Service

To standardize the implementation of a new element, a common interface named *ElementRemote* is provided, as presented in Fig. 7. Element developers need to implement an *ElementServer*, which is the subclass of *ElementRemote*. The *ElementRemote* interface is almost the same as the standard element interface. The only difference lies in the fact that two new methods are introduced in this interface. One is *formElement()* that is used by the client to send

the input data (geometry, nodes coordinates, etc.) to the actual element. The other is *clearElements()*, which will be called to do the "house-cleaning" after the analysis is finished. During the analysis, the output data (stiffness matrix, mass matrix, etc.) of each element can be obtained by calling the corresponding member functions. It should be noted that all the methods of the *ElementRemote* class also perform exception processing; they are ignored in Fig. 7 for clarity.

```
public class ElementRemote extends Remote {
    // This is the service name for publishing.
   public static final String SERVICE = "ElementService";
    // This is the port number, could be changed as needed.
   public static final int PORT = 12345;
    // This function is used to send the element data to server.
   public void formElement(int tag, Identity src, String input);
    // This function is used to perform house cleaning.
   Public void clearElements(Identity src);
   public int commitState(int tag, Identity src);
   public int revertToLastCommit(int tag, Identity src);
   public int revertToStart(int tag, Identity src);
   // Form element stiffness, damping and mass matrix.
   public MyMatrix getTangentStiff(int tag, Identity src);
   public MyMatrix getSecantStiff(int tag, Identity src);
   public MyMatrix getDamp(int tag, Identity src);
   public MyMatrix getMass(int tag, Identity src);
   public void zeroLoad(int tag, Identity src);
   public MyVector getResistingForce(int tag, Identity src);
   public MyVector getTestingForceIncInertia(int tag, Identity src);
}
```

FIG. 7. Class Interface of ElementRemote

6.6 DYNAMIC SHARED LIBRARY ELEMENT SERVICE

The distributed element model is flexible and convenient, however, the drawback here is performance. The initiation of remote procedure calls is quite expensive considering that it has to be done for every element and a typical structural model can have more than several thousands of elements. To improve performance without losing flexibility, the dynamic shared library service can be used. The mechanics of the *Dynamic Shared Library Element Service* is depicted in Fig. 8. In this model, the computation code of element is built in the form of dynamic shared library conforming to a standard interface. The shared library then can be put into an ftp server or an http server in the on-line service provider's site. When the core needs this element, the

dynamic shared library will be automatically downloaded from the service provider's site. The downloaded shared library can dynamically link with the analysis core during the run-time to perform structural simulation.

The mechanics of the dynamic shared library is different from those of the static library. With static libraries, objects within the library are linked into the program's executable file at link time. With dynamic shared libraries, objects within the library are not linked into the program's executable file, but rather the linker notes in the executable that the program depends on the library. When the program is executed, the system loads the dynamic libraries that the program requires. The advantage of linking dynamically with shared libraries over linking statically with static libraries is that the shared library can be loaded at runtime, so that different services can be replaced at runtime without relinking with the application.

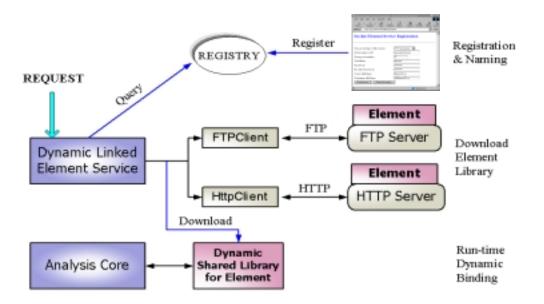


FIG. 8. Dynamic Shared Library Element Service

In summary, the new element can be developed in the following forms: static element library, distributed element service, and dynamic shared element library. Which form will be used for new element development is decided by the developers for their convenience. As long as the new element conforms to the standard interface, it should be able to communicate with the analysis core. After the development is completed, the new element service can be registered to the *Registration and Naming Service* by telling the server its name, location, form and other pertinent information. Users don't need to worry about what kind of element service to choose

and where to find the service. Although there are three representations of elements, the choosing and binding of element services are automated and totally transparent to users.

6.7 REGISTRATION AND NAMING SERVICE

Since multiple participants are engaged in the collaborative system, there is a need to uniquely identify them so that the tasks can be assigned. Also, if access to the system or to certain resources associated with the system needs to be restricted, then participant's identities will need to be authenticated. In the simple prototype implementation, a Java class is defined to record the service identity. The service is identified by a *name* property and an *id* property. The *name* property is a descriptive name that can be used to specify the service. The integer *id* is an internal identifier used to tag each service uniquely. We have designed the *Identity* class to implement the *Serializable* interface, so that *Identity* objects can be passed back and forth on the network. One important method of Identity is *equals()*, which can be used to justify if two identities are the same.

Another issue associated with the distributed environment is that the core server should be able to find on-line services. In the case of dynamic object activation, a naming service is also needed to find the proper dynamic shared element library. An elegant way to handle finding services would be to create a naming service for objects, where an agent of a certain service could register its service with the naming service and generate a unique name/address for the object. The naming service would be responsible for mapping named services to where they actual live. Users of the service could find the service with one name. The naming service permits clients to obtain references to objects they wish to use. It allows names to be associated with object references, via bindings. Clients may query a naming service using a predetermined name to obtain the associated object reference.

As shown in Fig. 9, a web-based form is used to gather the information of an on-line service. Some of the necessary data are the type of the service, the IP and port number of the service provider's site, developer's identity, and the description of the service. Based on the user input, the naming service can generate a unique *id* for each service. This *id* is used to find the service and to handle the binding.

On-line Element Service Registration - I	ficrosoft Internet Explorer
Elle Edit View Go Fgvoites Help	ter en la companya de la companya d
Back Forward Stop Refres	h Home Search Favoriles History Charr
Address http://eig.stanford.edu/8080/g3/reg	gister 💽 Linke
On-line Element Service Registration	
Choose the type of the service:	FTP download
Server name or IP:	eig stanford.edu
Server port number:	21
UserName:	theuser
Password:	
Re-enter Password	******
Source fileName:	Beam2D.so
Destination fileName:	lib/Beam2D.so
Submit form Clear the inputs	
8)	E Internet zone

FIG. 9. On-line Service Registration Form

6.8 DATA ACCESS SYSTEM

Traditionally, there are two ways of obtaining the requested analysis results from structural analysis programs. One way is pre-defining all the wanted data before analysis and saving the analyzed data during analysis. A problem associated with this method is that if the users want certain data other than pre-defined ones, a complete re-analysis has to be performed to generate requested results. For nonlinear dynamic analysis of a large structural model, the re-analysis can be very expensive in terms of both processing time and storage requirement. The other way of obtaining analysis results is simply putting all the interim and final analysis data into files. These files can be searched for the post-processing phase of structural analysis. Obvious drawbacks of this method are substantial amount of storage space and bad performance because of the expensive searching.

To overcome the disadvantages of the traditional ways of post-processing, we propose an on-line data access system for OpenSees. A commercial database system – Oracle 8i is used in our implementation – is employed for the efficient access of large structured data, like matrixes and vectors (Oracle 2000). In the proposed system, only *selected* results, not all results, will be stored into the database during the analysis. When the user wants to access the results, the

request will be forwarded to the analysis core. If the data that the user wants is already stored in the database, the action is no more than a database query; otherwise, the program will automatically instantiate the required new objects to re-compute the required data. Compared to the traditional way of redoing the whole analysis to obtain certain data, the re-computation should be more efficient and only involves a small portion of the program.

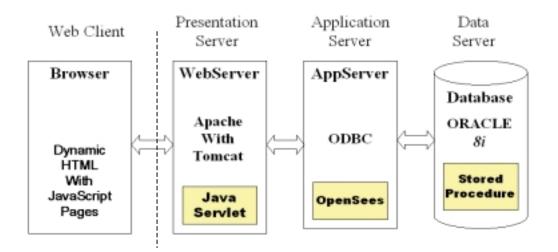


FIG. 10. Data Access System Architecture

Fig. 10 depicts the architecture of our proposed data access system. In this system, users interact with the server through web-based interfaces. Using dynamic HTML pages together with JavaScript code, the server will be able to generate rich and flexible content in users' browsers. The server side is divided into three layers. *Database*, which is used for the storage and retrieval of selected analysis results, is the back-end of the system. Java Servlet enabled *Web Server* is employed to receive users' requests and hand them to the *Application Server*. The *Application Server* is the middle layer for handling communication between *Web Server* and *Database* as well as doing analysis and generating results. As shown in Fig. 10, some extra functionality can be added to OpenSees in order to modify it into the *Application Server*.

6.9 DATA QUERY LANGUAGE

The most commonly used relational database systems query and modify the database through a language called SQL (structured query language). In our data access system, an SQL-like query language is defined to query the analysis result. This DQL (data query language) uses the same

syntax as SQL and is able to query the result together with some post-processing capabilities. Some commonly used features of the DQL are illustrated below.

As an example, we query the displacement from dof 1 of node number 4,

```
SELECT disp FROM node=4 AND dof=1;
```

The analysis result can also be queried from element, for example,

SELECT stiff FROM element=2;

returns the stiffness matrix of element 2.

If we want to obtain the displacement from all the nodes, the wildcard * can be used.

SELECT disp FROM node=*;

In order to have persistent storage of the analysis result, the queried data can be saved into files. These files can then be processed to generate graphical representation. For instance, if we wan to save the displacement time history of a particular node for a nonlinear analysis, the following query can be used.

```
SELECT time disp FROM node=19 AND dof=1
SAVEAS node19_1.out;
```

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 the column. In the current implementation, five operators are provided that apply to a column and produce some summary or aggregation of that column. These operators are:

- 1. SUM, the sum of the values in this column.
- 2. AVG, the average of values in this column.
- 3. MIN, the least value in the column.
- 4. MAX, the greatest value in the column
- 5. COUNT, the number of values.

For example, the following query finds the maximum nodal displacement.

SELECT MAX(disp) FROM node=*;

6.10 EXAMPLES

To illustrate the collaborative model, this section presents two sample examples. The first example is a simple linear-elastic three bar truss structure, as shown in Fig. 11, subject to static

loads. The Tcl script input file for this model can be submitted to the server in a web-based environment and the result will be returned to the browser, as shown in Fig. 4.

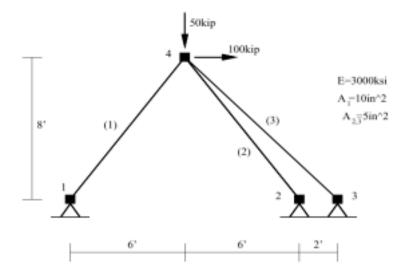


FIG. 11. Simple Truss Example

After the analysis, the user can also interact with the server to query the result. As we discussed earlier, there are two types of user interfaces: one is web-based, and the other is Matlab-based. Fig. 12 illustrates a simple web-based analysis request form. The analysis result can be either downloaded as a file (as *example1.out* in Fig. 12) or be interactively queried. Fig. 12 shows the process of requesting displacement at Node 4.

The second example we will illustrate is a three-dimensional rigid frame model. The model consists of three stories and one bay in each direction. Rigid diaphragm multi-point constraints are used to enforce the rigid in-plane stiffness assumption for the floors. Gravity loads are applied to the structure and the 1978 Tabas acceleration records are the uniform earthquake excitation that has been applied to both X and Y directions.

Nonlinear beam column elements are used for all members in the structure. The beam sections are elastic while the column sections are discretized by fibers of concrete and steel. A solution algorithm of type Newton is used for the nonlinear problem. 1000 steps are performed with a time step of 0.01.

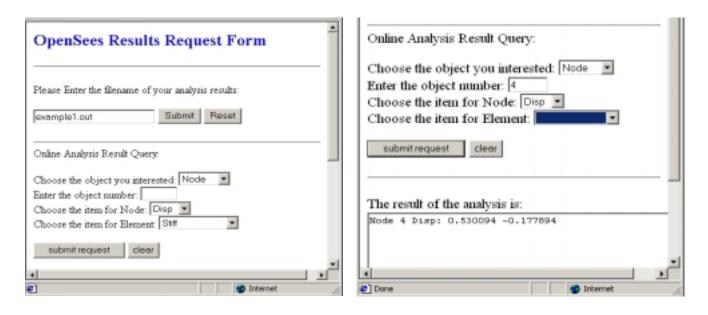


FIG. 12. Analysis Results Request Form

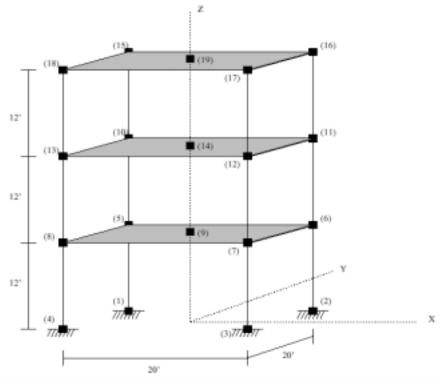


FIG. 13. Three-Dimensional Rigid Frame

For this example, we will use the Matlab-based interface to interact with the server. The first step is still editing the input script file. Then we can invoke our pre-defined Matlab add-on

functions from the Matlab command prompt. The input analysis model can be submitted by entering the following command:

>>submitmodel RigidFrame3D.tcl

After the analysis, the model can be plotted in Matlab by invoking,

>>modelplot

the plotted model is shown in Fig. 13.

If we want to interactively query the analysis result, the *queryresult* command can be used to enter the DQL (Data Query Language) environment. Then we can use the query language to get the wanted data.

>>queryresult
Please enter your query:
SELECT time disp FROM node=19 AND dof=1
SAVEAS node19_1.out;

The result consists of the file node19_1.out, which contains a line for every time step. Each line contains the time and the horizontal displacement at the diaphragm master node 19. To obtain a graphical representation of this displacement time history, we can use the command

>>res2Dplot node19_1.out

to take advantage of Matlab's graphical processing power. The result is shown in Fig. 14.

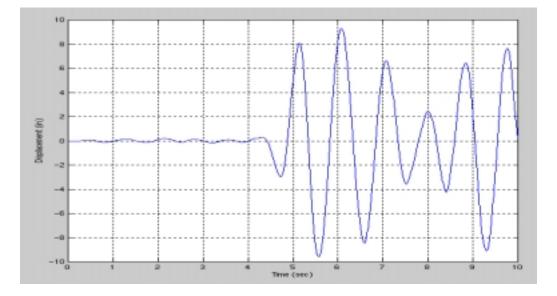


FIG. 14. Node 19 Displacement Time History

6.11 SUMMARY

A prototype of the Internet-enabled collaborative platform has been implemented and tested. The collaborative framework has been shown to have greater flexibility and extensibility than the current engineering approaches. A diverse group of users and developers can easily access the platform and attach their own developments to the core. Some avenues to improve the current platform will be further investigated and the implementation will continue to enhance robustness and flexibility.

REFERENCES

- Han, C. S., Kunz, J. C. and Law, K. H. (1999). "Building Design Services in a Distributed Architecture." *Journal of Computing in Civil Engineering*, 13(1), 12-22.
- Hopkins, J. (2000). "Component Primer." Communication of the ACM, 43(10), 27-30.
- McKenna, F. (1997). "Object Oriented Finite Element Analysis: Frameworks for Analysis Algorithms and Parallel Computing." *Ph.D. Thesis, Department of Civil Engineering, University of California, Berkeley*, CA.
- McKenna, F. and Fenves, G. L. (2000), "An Object-Oriented Software Design for Parallel Structural Analysis." *Structural Congress & Expositions ASCE*. May 2000, Philadelphia, PA.
- Oracle Corporation (2000). "Building JSP Internet Application with Oracle JDeveloper." An Oracle Technical White Paper. April 28, 2000.
- Peng, J. and Law, K. H. (2000a), "Framework for Collaborative Structural Analysis Software Development." *Structural Congress & Expositions ASCE*. May 2000, Philadelphia, PA.
- Peng, J., McKenna, F., Fenves, G. L. and Law, K. H. (2000b), "An Open Source Model for Collaborative Development of Finite Element Program." *International Conference on Computing in Civil and Building Engineering*, August 2000, Palo Alto, CA.
- Santiago, E. D. (1996). "A Distributed Implementation of The Finite Element Method for Coupled Fluid Structure Problems." *Ph.D. Thesis, Department of Civil Engineering, Stanford University, Stanford, CA.*