Engineering with Computers (2002) 18: 38–49
Ownership and Copyright
© 2002 Springer-Verlag London Limited

Engineering with Computers

A Prototype Software Framework for Internet-Enabled Collaborative Development of a Structural Analysis Program

J. Peng and K. H. Law

Department of Civil and Environmental Engineering, Stanford University, Stanford, CA, USA

Abstract. *This paper describes a prototype implementation of a software platform that enables the utilization and collaborative development of structural analysis programs, taking advantage of the Internet and web-based technologies. The new platform would allow users to have easy access to the structural analysis platform via the Internet. Users can have direct access to the analysis program and the analysis results by using a web-browser or other application programs, such as MATLAB. Furthermore, the platform is intended to serve as a common finite element structural analysis tool for which researchers and users can build, test, and incorporate new developments. Researchers and developers can link the analysis platform with their developments by utilizing pre-defined Internet-enabled communication interfaces. The prototype system is applied to perform a nonlinear dynamic analysis for a two-dimensional frame structure. It is shown that the Internet-enabled collaborative paradigm can potentially provide greater flexibility and extendibility than traditional structural analysis programs, which are typically packaged individually.*

Keywords. Collaborative software development; Communication protocol; Component-based framework; Distributed element service; Finite element program; Internet service

1. Introduction

It is well recognized that a significant gap exists between the advances in computing and the state-of-practice in structural engineering software development. Practicing engineers today typically perform finite element structural analysis on a dedicated computer, using the elements and analysis algorithms that are provided by finite element analysis programs. These finite element analysis programs usually bundle all the procedures and program kernels into software packages that are developed by individual organizations. As technologies and structural theories advance, structural analysis software packages need to be able to accommodate new developments in element formulation, material relations, analysis strategies, solution strategies, as well as computing environments. For the current state of finite element software packages, modifying or extending the code requires that the users have intimate knowledge of the data structures and what procedures affect what portions of the code. The ability to reuse code from other sources is also limited. This is because data structures vary widely between programs. As a consequence, introducing code from other sources often requires that the code be modified to suit the data structure used in the finite element program. The current trend sees the utilization of object-oriented programming in finite element software development in order to support better data encapsulation and to facilitate code reuse [1–10]. Object-oriented finite element packages, particularly those written in C++, have been shown [11,7,8] to have comparable performance to their procedural-based counterparts and provide certain maintainability and extendibility to modern day software packages. Although these programs use intrinsic object-oriented support of abstraction, encapsulation, inheritance, and polymorphism, the codes are usually not designed in a very flexible and extendible way to allow the analysts to easily experiment with their own elements or analysis algorithms. Extending and upgrading these programs to incorporate new developments remains 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 current development of Open System for Earthquake Engineering Simulation (OpenSees) by the Pacific Earthquake Engineering Research (PEER)

Correspondence and offprint requests to: K. H. Law, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305, USA. E-mail: law@cive.stanford.edu

Center is facing the same challenge. The goal of the OpenSees development is to improve the modeling and computational simulation in earthquake engineering through open-source development. The analysis core of OpenSees is based on an object-oriented framework for nonlinear dynamic analysis of structural and geotechnical systems [8]. The focus of OpenSees is to support the communication and cooperation of users and researchers, and to facilitate the incorporation of their research developments into the framework. Many participants from different universities and organizations are involved in the utilization and development of OpenSees with different perspectives. There are users whose main purpose is to use OpenSees solely as a structural and geotechnical analysis tool. There are core developers who are working to incorporate new developments and to expand the analysis core. There are also analysts and researchers whose focus is to develop new element and material technologies. Since the development is concurrent and incremental and is not controlled by a monolithic organization, traditional software engineering paradigms and waterfall software development approaches are not appropriate for the open collaborative effort.

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 [12]. Following this idea, we have designed and prototyped an Internet-enabled collaborative framework [13,14] for the continuing usage and development of the finite element structural analysis program, OpenSees. A collaborative system is one where multiple users or agents engage in a shared activity, usually from remote locations. By using the Internet as a communication avenue, the framework can make the structural analysis program more easily accessible by the end users. Unlike the development of traditional packaged structural analysis programs, the collaborative framework can potentially reduce the overhead of continuous upgrade and extension. Developers and researchers can concentrate on developing components and then easily integrate their components to the core through a *plug-and-play* environment.

2. Architecture of the Collaborative Framework

When integrating multiple computer applications or systems, a suitable architecture is needed to define how the components (applications or systems) are connected (linked) within the constraints [15]. The overall system architecture of the Internet-enabled collaborative framework is schematically depicted in Fig. 1. The architecture defines the dependency and the interaction among the participants:

- In this framework, the structural analysis core program is running on a central server as a compute engine. At the heart of the compute engine is a protocol that allows jobs to be submitted to the compute engine, the compute engine to run those jobs, and the results of the job to be returned to the client. This protocol is expressed in interfaces supported by the compute engine and by the objects that are submitted to the compute engine.
- In this collaborative system, users play the role of clients to the central finite element compute engine. The users can have direct or remote access (one such avenue is the Internet) to the core program through a web-based user interface or other application programs, such as MATLAB. The users can specify desirable features and methods (element types, efficient solution methods, and analysis strategies) contributed by other developers that have been tested and incorporated into the core platform.
- For element developers, a standard interface/ wrapper is defined for communicating the element(s) with the analysis core. The element code can be written in languages such as Fortran, C, C++ and/or Java as long as it conforms to the standard interface, which is a set of predefined protocols to bridge element code with the central server. If the developer and the system administrator agree, the new element can be merged 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 devel-

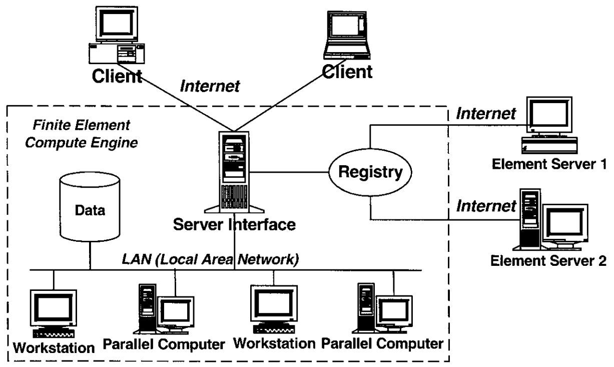


Fig. 1. System architecture for a collaborative finite element program.

oper needs to register the element code and its location to the core and the element service can then be accessed remotely over the Internet.

Since the 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 a more complex and complete system behavior [16]. 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.

Usually, a component can be viewed as a black-box entity that provides and/or requires a set of services (via interfaces) [17]. For a finite element program, element code can be treated as a component. Since there are continuing new developments in element technologies, building an element as a separate component can facilitate the concurrent development and the eventual incorporation of the new element into the core. In the prototype framework, a database is used for efficient data storage and flexible post-processing. Since the database module is loosely coupled with the core program, it is also a good candidate for building as a component. As presented in Fig. 2, the Internet-enabled structural analysis platform consists of six distinct modules:

- The *Analysis Core* module is the part that consists of a set of basic functional units of a finite element structural analysis program. Element and material models, solvers, as well as solution strategies, are brought into this module to provide the basic functionality of the core.
- The *User-Interaction Interface* module provides an interface to facilitate the access to the software platform by the users and developers. The platform can be accessed from either a web-based interface or other application programs.
- The *Registration and Naming Service* is provided for on-line services to register to the core so that these services can be found and accessed during analysis.
- Two approaches are provided for remote access to element services residing in different locations. The *Distributed Element Service* is intended to provide a communication link to remote element services where the element code is executed. The *Dynamic Linked Element Service* is implemented to dynamically load the element code from a remote element service and to link and bind the code with the core at runtime.
- The *Database Interface* module is built to provide efficient data access and to enable post-processing tasks.

The mechanics of the collaborative model is illustrated in Fig. 3. First, users build the structural model in the client site and submit it to the analysis core via a web-browser or an application program using the Internet as a communication channel. Upon receiving the model, the core server performs an analysis in a distributed and collaborative manner. During the analysis, elements that are available in the core can be accessed locally from the static element library (this is the case for most prevailing finite element packages), whereas other elements are obtained from on-line element services. To find the 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 been previously registered with the core platform. The on-line element services are dynamically linked with the analysis core to perform the analysis. After the analysis is completed, part of the results

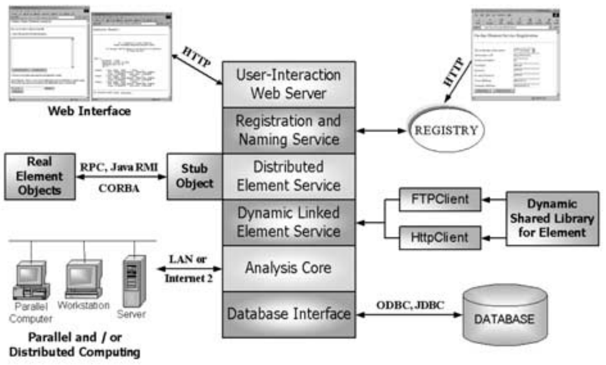


Fig. 2. Collaborative system modules.

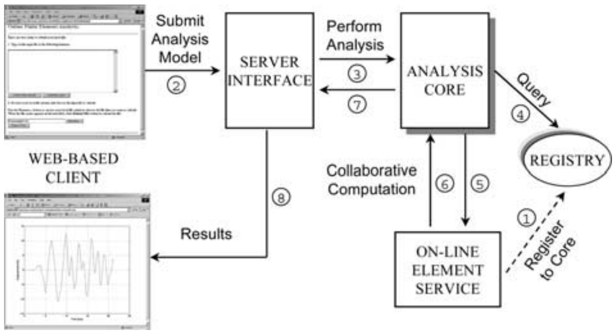


Fig. 3. Mechanics of the collaborative model.

will be returned to the user by generating a dynamic web page in the user's web browser. The user can also query and view the analysis results using a web browser or other application programs.

A prototype of the Internet-enabled collaborative system has been implemented using a Sun Ultra 60 workstation as the hardware platform. Apache HTTP server is used as the web server for handling users' requests. Apache Tomcat, which is an implementation of the Java Servlet 2.2 technologies, is deployed as the Java Servlet server. For persistent storage of analysis results, the Database Management System (DBMS) is Oracle 8i with the Open Database Connectivity (ODBC).

All six modules shown in Fig. 2 have been implemented in the prototype system. As we mentioned early, they are developed in a loosely coupled manner by using component-based design and modeling. This paper will focus on describing four of the implemented modules, which are *the User-Interaction Module, the Registration and Naming Service, the Distributed Element Service*, and *the Dynamic Linked Element Service*. To illustrate the mechanics of the prototype collaborative system, a simple linear-elastic three bar truss structure subject to static loads, as shown in Fig. 4, is employed. The model consists of four nodes, three truss elements, nodal loads acting at node number 4, and the fixed constraints at the three support nodes.

3. User Interaction

As indicated in Fig. 1, the collaborative framework can offer users access to the analysis core, as well as the associated support services via the Internet. This client/server computing environment consists of two logical parts: a server that provides services and a client that requests services of the server.

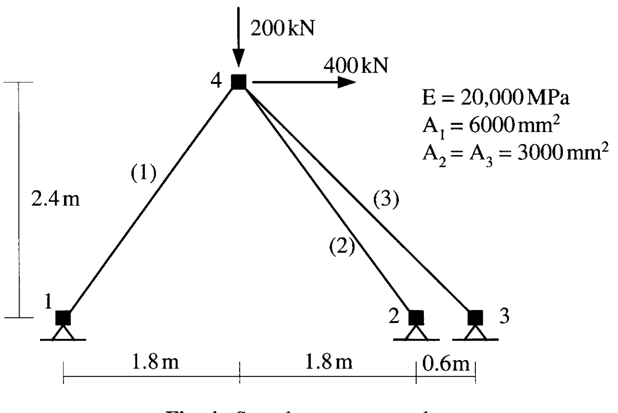


Fig. 4. Sample truss example.

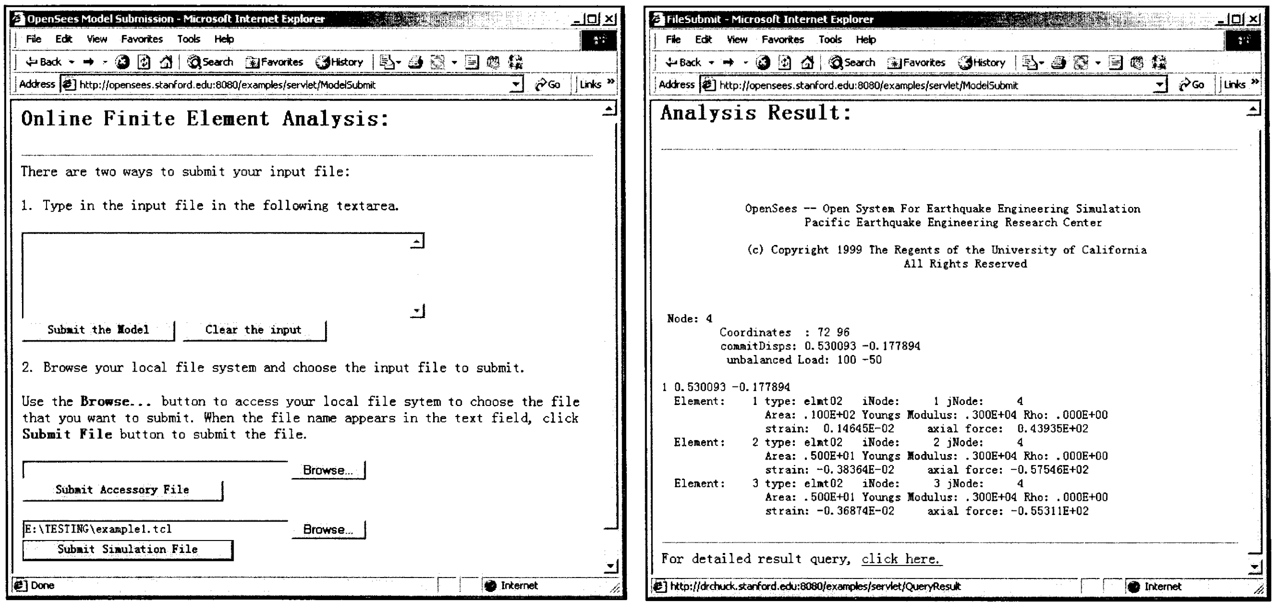
Together, the two parts form a complete computing framework with a very distinct division of responsibility [18]. One benefit of this model is the transparency of software services. From a user's perspective, the user is dealing with a single service from a single point of contact – even though the actual structural analysis may be performed in a distributed and collaborative manner. The other benefit is that this framework can widen the reach of the analysis core to the users and external developers. The core platform offering the finite element analysis service stays at the provider's site, where the software core is developed, kept securely, operated, maintained and updated. Users can easily access the software platform without the associated cost and maintenance challenges.

3.1. Web-Based User Interface

For the collaborative system, a standard World Wide Web browser is employed to provide the user interaction with the core server. Although the use of a web browser is not mandatory for the functionalities of the collaborative framework, using a standard browser interface leverages the most widely available Internet environment, as well as being a convenient means of quick prototyping.

To conduct an analysis using the current version of OpenSees, most users need to set up an input file using the scripting language Tcl (Tool Command Language) [19]. In the web-based user interface, two modes of inputting a Tcl script are accepted. Users can directly submit Tcl command lines to the server; or they can first edit a Tcl script file and then submit the input file to the core server. After the server performs the analysis, the results can be returned to the users by dynamically generating a web page in their browser. As an alternative, the users can also receive the analysis results recorded in an output file. Figure 5 presents the web pages of the truss example for the submission of an input file and for the analysis results reported.

For the server to process the HTTP requests from the user, Apache Tomcat, which is built on Java Servlet based technologies, is deployed as the entry point of server's process. Whenever Apache Tomcat receives a request for an analysis, it will start a new process to run OpenSees. Because Java provides simple methods for incorporating external processes, the integration of OpenSees – which is a C++ application – with the Java Servelet server is fairly straightforward. After the analysis results are returned from OpenSees to Tomcat, the web server


OpenSees Model Submission - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://opensees.stanford.edu:8080/examples/servlet/ModelSubmit
Go Links
Submit the Model
Clear the input
Browse...
Submit Accessory File
Browse...
Submit Simulation File
Done
Internet
(a)
FileSubmit - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://opensees.stanford.edu:8080/examples/servlet/ModelSubmit
Go Links
http://drchuck.stanford.edu:8080/examples/servlet/QueryResult
Internet
(b)

```
Online Finite Element Analysis:

There are two ways to submit your input file:

1. Type in the input file in the following textarea.

2. Browse your local file system and choose the input file to submit.

Use the Browse... button to access your local file sytem to choose the file
that you want to submit. When the file name appears in the text field, click
Submit File button to submit the file.

E:\TESTING\example1.tcl
```

```
Analysis Result:

          OpenSees -- Open System For Earthquake Engineering Simulation
                  Pacific Earthquake Engineering Research Center

          (c) Copyright 1999 The Regents of the University of California
                                All Rights Reserved

 Node: 4
        Coordinates  : 72 96
        commitDisps: 0.530093 -0.177894
         unbalanced Load: 100 -50

1 0.530093 -0.177894
  Element:    1 type: elmt02   iNode:     1 jNode:     4
                Area: .100E+02 Youngs Modulus: .300E+04 Rho: .000E+00
                strain:  0.14645E-02     axial force:  0.43935E+02
  Element:    2 type: elmt02   iNode:     2 jNode:     4
                Area: .500E+01 Youngs Modulus: .300E+04 Rho: .000E+00
                strain: -0.38364E-02     axial force: -0.57546E+02
  Element:    3 type: elmt02   iNode:     3 jNode:     4
                Area: .500E+01 Youngs Modulus: .300E+04 Rho: .000E+00
                strain: -0.36874E-02     axial force: -0.55311E+02

For detailed result query, click here.
```

Fig. 5. Web pages generated in the client site. (a) File submission for analysis model, (b) analysis results.

will package the results in a properly generated web page and send the web page back to the user's web browser. One feature of this model is that Java Servlet supports multithreading, so that several users can send requests for analysis simultaneously and the server is still able to handle them without severe performance degradation.

3.2. MATLAB-based User Interface

For web-based services, all too often analysis result is downloaded from the computational server as a file, and then put manually (cut and paste plus maybe some cumbersome conversions) 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 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 user interaction with the core. In our prototype system, a MATLAB-based user interface is available to take advantage of the flexibility and graphical processing power of MATLAB. The user can also customize the post-processing of the results. In the implementation, some extra functions are added to the standard MATLAB in order to handle the network communication and data processing. These add-on functions can be directly invoked from either the MATLAB prompt or a MATLAB-based graphical user interface. Similar network communication protocols can be built for other application programs.

For the truss example in Fig. 4, the command, `submitmodel truss.tcl`, can be issued to submit the input Tcl file to the analysis server. Some pre-defined commands can be invoked to generate graphical representations of the model. Figure 6 presents the plots of the truss model and its deformed shape using the commands `modelplot` and `deformedplot 20`, where `20` is an amplification factor.

MATLAB can also be employed directly for post-processing purposes, i.e. the analysis results from the core server can be directly queried from MATLAB by using a data query language (DQL).

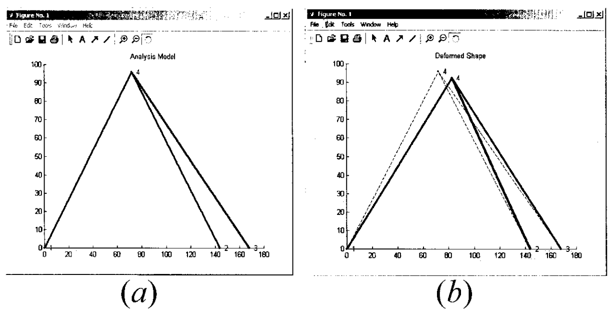


Fig. 6. Sample MATLAB-based user interface. (a) modelplot, (b) deformedplot 20.

The DQL is defined in a systematic way and it is capable of querying the analysis results. As an example, we can query the displacement from degree of freedom (dof) 1 of node 4 by issuing

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

We can also query the information from an element, for example,

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

returns the 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 ? may be used on certain object to find out what kind of queries it can support. For example, the following query

`SELECT ? FROM node=4` returns

```
  Node 4:: NumDOF Crds Disp Vel Accel
Load TrialDisp TrialVel TrialAccel *
```

If the user desires, the results returned can then be post-processed directly using MATLAB.

4. On-line Element Services

One of the salient features of the collaborative finite element software platform is to support analysts to integrate new element technologies and material models with the core server. In this collaborative framework, a standard interface/wrapper is defined for communicating the element with the object-oriented analysis core. Using a standard interface facilitates the concurrent development of new elements. It allows the replacement of an existing element code if a superior one becomes available. The encapsulation and inheritance features of object-oriented programming are utilized to define the standard interface for the element. In OpenSees, a super-class *Element* is defined in the analysis kernel (for details, see McKenna [8]). To introduce a new element into the analysis core requires simply creating a subclass of the *Element* class. The new element code, once tested and approved for adoption, may become part of the core's static element library.

In addition to contributing a new element to the analysis core directly, the developer can also choose to be an on-line element service provider. Two forms of on-line element services, namely *distributed element service* and *dynamic shared element service*, are introduced in the prototype system. Which form to be used for linking the element services with the core is up to the developers for their convenience and other considerations. As long as the new element conforms to the standard interface, it will be able to communicate with the analysis core. As opposed to the traditional statically linked element library, the on-line element services will not expose the source code to the core. The collaborative platform allows the building of proprietary element services and the linking of legacy applications.

The on-line element service can be released to public use by registering itself to the *Registration and Naming Service* with its name, location, service type (whether a distributed service or a dynamic shared library service) and other pertinent information. During a structural analysis, the *Registration and Naming Service* can be queried to find the appropriate type and location of element service. Although there are three types of element services (static element library and two forms of on-line element services), the selection and binding of element services are automatic and completely transparent to the users. The end users do not need to know the type of an element service to choose and the location of the service.

4.1. Registration and Naming Service

To support distributed services with many participants, the core server must be able to locate appropriate services for specific tasks. One approach is to create a naming service for objects, where an agent of a certain service could register its service to the naming service and generate a unique name and address for the object. The naming service would be responsible for mapping the named services to the physical locations. With the naming service, the users (clients) can obtain references to the objects (services) they wish to use. The service allows names to be associated with object references. Clients may query the naming service to obtain the associated object reference and the description of the service.

In the prototype implementation, a Java class *Identity* is defined to record the service identity. The service is identified by a *name* property and an *id* property. The string *name* property is a descriptive name that can be used to specify the service. The integer *id* is an internal identifier generated to uniquely tag each service. The *Registration and Naming Service* stores all the *Identity* objects in a hash-table keyed by the *name* property. 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 the *Identity* class is *equals()*, which can be used to identify if two identities are the same.

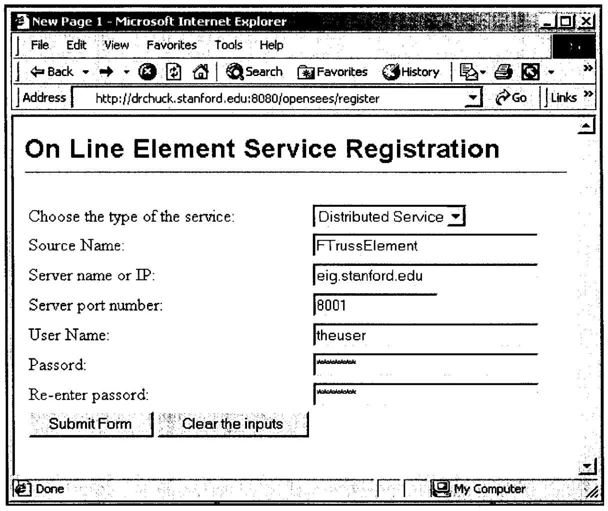
As an example, the truss element in the problem shown in Fig. 4 is built in the form of a distributed element service, which resides on a separate computer than the core server. The service is pre-registered with the central server using the web-based registration form as presented in Fig. 7. The information required for the registration service is the type of the service, the name of the service, the IP and port number of the service provider's site, developer's identity and password, and a description of the service. If the input name is already existed in the service list, the registration server will inform the developer to choose a different name. Based on the input data, the naming service can generate a unique *Identity* object for the service. This *Identity* is used later to find the service and to handle dynamic binding of the element service with the core server.

4.2. Distributed Element Service

For the distributed element service, the actual code resides in the service provider's site. As shown in Fig. 8, whenever the core needs certain element data, for example a stiffness matrix, the core will request the service provider through a sequence of remote method invocations. The computation (i.e. the forming of the stiffness matrix of an element) is performed at the service provider's site and the results are then returned as the return of remote method calls.

The essential requirements in a distributed object system are the ability to create and invoke objects on a remote host or process, and interact with them as if they were objects within the same process. To do so, some kind of message protocol is needed for sending requests to remote agents to create new objects, to invoke methods on these objects, and to delete the objects when they are done. Assorted tools and standards for assembling distributed computing applications have been developed over the years. They started as low-level data transmission APIs and protocols, such as TCP/IP and RPC, and have recently begun to evolve into object-oriented distribution schemes, such as CORBA, DCOM, Java RMI, and OpenDoc. These programming tools essentially provide a protocol for transmitting structured data (and, in some case, actual running code) over a network connection.

In the prototype implementation, Java's Remote Method Invocation (RMI) is chosen 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 allows distributing computational tasks across a networked environment and thus enables a task to be performed on the machine most appropriate for the task [20]. RMI can be implemented efficiently and it can be beneficial for scientific and engineering computing [21,22]. The *skeleton*, which is the object at the server site,


New Page 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://drchuck.stanford.edu:8080/opensees/register
Go Links
Done
My Computer

On Line Element Service Registration

Choose the type of the service:	Distributed Service
Source Name:	FTrussElement
Server name or IP:	eig.stanford.edu
Server port number:	8001
User Name:	theuser
Passord:	******
Re-enter passord:	******

Submit Form | Clear the inputs

Fig. 7. Registration and naming service interface.

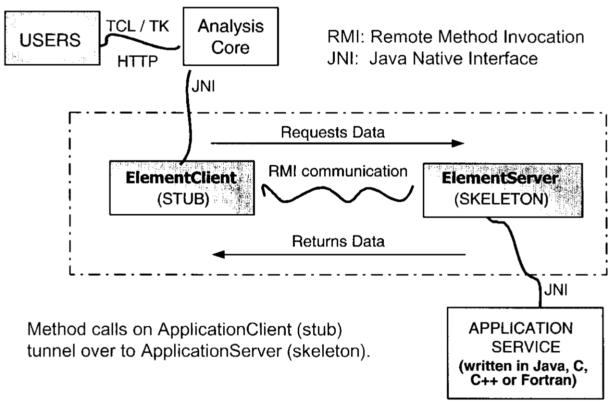

USERS
TCL / TK
HTTP
Analysis Core
RMI: Remote Method Invocation
JNI: Java Native Interface
JNI
Requests Data
ElementClient (STUB)
RMI communication
ElementServer (SKELETON)
Returns Data
JNI
Method calls on ApplicationClient (stub) tunnel over to ApplicationServer (skeleton).
APPLICATION SERVICE (written in Java, C, C++ or Fortran)

Fig. 8. Mechanics of the distributed element service.

receives method invocation requests from the client. The skeleton then makes a call to the actual object implemented on the server. The *stub* is the client's proxy representing the remote object and defines all the interfaces that the remote object supports. To illustrate the interaction between a distributed element service and the core server, Fig. 9 shows the interaction diagram of a typical distributed element service. The *TrussElementClient* object plays the role of a *stub* that forwards the core server's requests (*formElement()* and *getStiff()*) to the element service. The *TrussElementServer* object is a *skeleton* that defines the entry point of this *TrussElement* service. The actual element service itself is an example of a legacy application, which is, in this case, written partly in C and partly in Fortran. Once wrapped and conformed to the protocol, the legacy application can be integrated into the distributed service architecture.

The key item in the open collaborative system presented earlier is interfaces. A set of interfaces should be fully defined, available to the public, and appropriately maintained. To standardize the implementation of a new element, a common interface named *ElementRemote* is provided, as presented in Fig. 10. Following the object-oriented paradigm, the 'exposed' methods are the points of entry into the element services, but the actual implementation of these methods is dependent on the individual services. Element developers need to implement an

```
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 = 5432;

    // This function is used to send the element data to server.
    public int formElement(int tag, Identity src, char[] input);
    // This function is used to perform house cleaning.
    Public int 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. 10. Class interface of *ElementRemote*.

ElementServer, which is a subclass of *ElementRemote*. The *ElementRemote* interface is almost the same as the standard element interface that is provided by the core OpenSees program. The only difference lies in the fact that two additional methods are introduced in this interface. One is *formElement()* that is used by the client to send the input data (geometry, nodal coordinates, etc.) to the actual element. The other is *clearElements()*, which can be called to perform the 'house-cleaning' task once the analysis is complete. During the analysis, the output data (stiffness matrix, mass matrix, etc.) of each element can be obtained by calling the

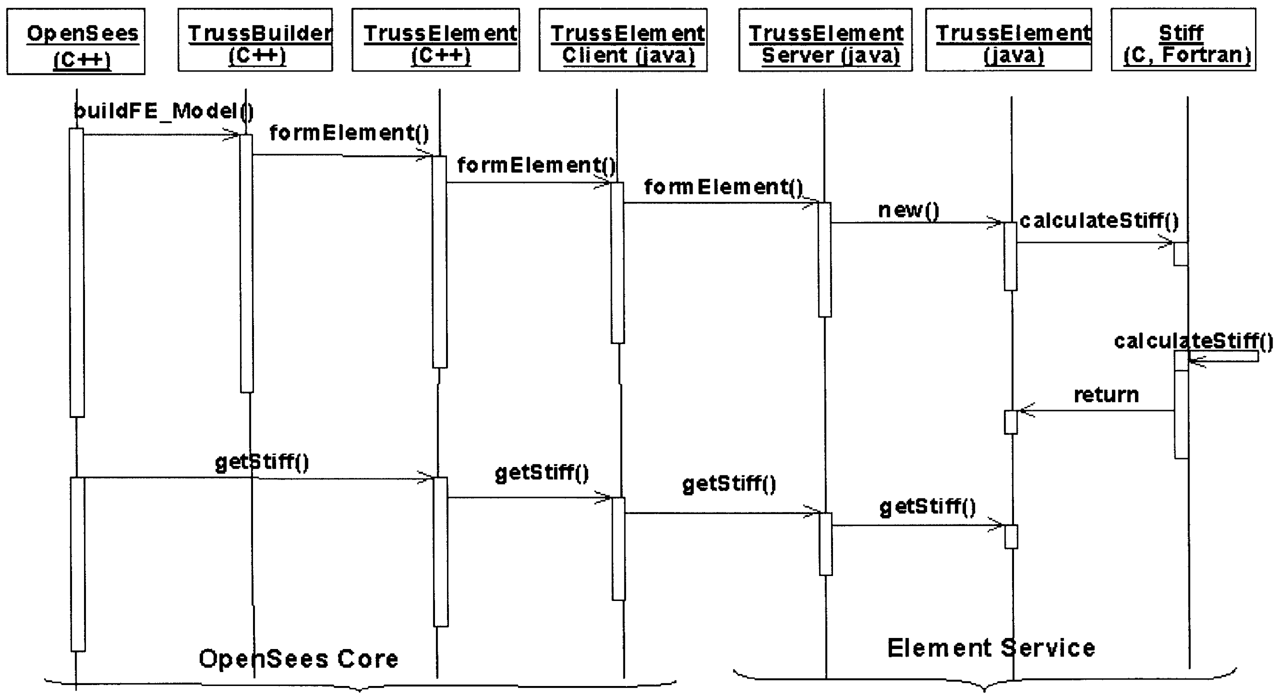


Fig. 9. Interaction diagram of the distributed truss element service.

corresponding member methods. Although it is not shown in Fig. 10 for the purpose of clarity, exception handling is included in the implementation for all the methods.

When the analysis core needs to use the distributed element, it instantiates and makes method calls to the element in the same way as it treats a local element object. The actual method calls on the *ElementClient* and tunnels over to the *ElementServer*. Figure 11 shows partial sample code that illustrates the usage of the two methods: *formElement()* and *getStiff()*. Upon receiving a *formElement()* request from the *ElementClient*, the *ElementServer* will instantiate a new *Element* object and start a new thread to compute the element stiffness matrix. After the computation is complete, the stiffness matrix is saved in a hash-table. The next time when the *Client* issues a call *getStiff()* for the stiffness matrix, the hash-table will be searched and the data can be sent to the caller as a function return.

Let's consider the truss example in Fig. 4, where the truss element is built in the form of a distributed element service. When the server receives the input model (Tcl file), the core program will first consult the *Naming and Registration Service* to find the information of the distributed truss element. The *TrussElementClient* will then use the queried information to initiate the communication with the *TrussElementServer*. After the communication is initiated, the core analysis program treats the truss element in the same way as a local element. Again, the searching and the binding of an on-line element service are automated – the user is not aware of the difference between a local element and an online element.

```
public class TrussElementClient extends ElementClient {
   // based on the server name and port number, creates stub object.
   public TrussElementClient(String server)
   {
      System.setSecurityManager(new RMISecurityManager());
      String name = "//" + server + ":" + ElementRemote.PORT + "/" +
                    ElementRemote.SERVICE;
      theStub = (ElementRemote)Naming.lookup(name);
   }

   // calls on the server are just method calls to stub.
   public void formElement(int tag, Identity src, char[] input)
   {
      theStub.formElement(tag, src, input);
   }

   public MyMatrix getTangentStiff(int tag, Identity src)
   {
      MyMatrix result = new MyMatrix(4, 4);
      result = theStub.getTangentStiff(tag, src);
      return result;
   }
}
```

```
public class TrussElementServer extends UnicastRemoteObject
                           implements ElementRemote {
   // A hash-table is used to store all the elements
   private Hashtable allElements = new Hashtable();

   public void formElement(int tag, Identity src, char[] input)
   {
       TrussElement newElement = new TrussElement(tag, input);
       allElements.put(tag, newElement);
   }

   public MyMatrix getTangentStiff(String tag, Identity src)
   {
       TrussElement oneElement = (TrussElement)allElements.get(tag);
       result = oneElement.getTangentStiff();
       return result;
   }
}
```

Fig. 11. Sample *ElementClient* and sample *ElementServer*.

4.3. Dynamic Shared Library Element Service

The distributed element service model described previously is flexible and convenient; however, the approach does carry some overhead on remote method invocation, which may be quite expensive considering that it has to be done for every call to an element. A dynamic shared library element service is designed to alleviate the problem and to improve performance without losing flexibility. The mechanics of *the Dynamic Shared Library Element Service* is depicted in Fig. 12. In this approach, the element code is built in the form of a dynamic shared library conforming to a standard interface. The shared library is placed on an FTP server or an HTTP server in the on-line service provider's site. When the core calls for the element, the dynamic shared library will be downloaded from the service provider's site. The downloaded shared library can then be dynamically linked with the analysis core during run-time.

The mechanics of a dynamic shared library differs from a static library. With a static library, objects within the library are linked into the program's executable file at compilation time. For a dynamic shared library, 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. One advantage of linking

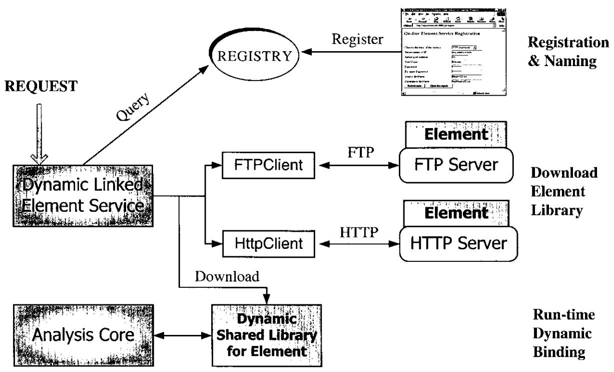
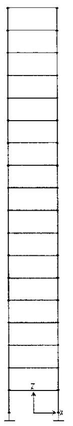


Fig. 12. Mechanics of the dynamic shared library element service.

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 re-compilation and/or re-linking with the application. Another benefit of using dynamic shared library is that the shared library is in binary format. This guarantees that the source code of the element will not be exposed to the core server. This also implies that the element developer controls the maintenance, quality, and upgrade of the source code. However, the dynamic shared library service bears some disadvantages. The most prominent one is platform dependency. To support dynamic loading and binding, the shared library must be built on the same platform as the core server. Other disadvantages may include potential security problem and minor performance overhead due to network download and dynamic binding.

5. Example

This section presents a nonlinear dynamic analysis to illustrate the usage of the collaborative software framework. Figure 13 shows the sketch of a structural model and part of the input Tcl file for the model. The structural model is an 18 story two-dimensional one bay frame. The model is finetuned so that beam hinging occurs simultaneously at the ends of the beams and at the bottom of the first story columns. The story heights are all 12 feet and the span is 24 feet. The beams and columns are modeled by using *ElasticBeamColumn* element and the hinging is modeled with zero-length elasto-plastic rotational element. A nonlinear dynamic analysis is performed using the 1994 Northridge earthquake recorded at the Saticoy St. station.



```
# Create ModelBuilder (with two-dimensions and 3 DOF/node)
model basic -ndm 2 -ndf 3

# build the model
#    tag    X       Y
node  1     0.0  2592.0 -mass $mass 0.0 0.0
. . .
# add material
#                           tag    Ke       rotp
uniaxialMaterial ElasticPP  1   239781.1  0.008674579
. . .
# add beam-column elements
#                           tag ndI ndJ  A    E    Iz  transfTag
element elasticBeamColumn   1  39  40 1e06  3e04  1e06   1
. . .
# add zero-length hinge elements
#           eleID nodeR nodeC matID
rotSpring2D  55     1    39     1
. . .
# Fix supports at base of columns (hinged columns)
#    tag   DX   DY   RZ
fix  75     1    1    1

# Create the solution algorithm, a Newton-Raphson algorithm
algorithm Newton

set accelSeries "Path -filePath NR94stc.dat -dt 0.01 -factor 386.4"

# perform the analysis
analyze Transient
analyze 2500 0.01
```

Fig. 13. The model and part of the input Tcl file.

Figure 14 illustrates the interaction of the distributed services during the simulation of the model. The analysis core, OpenSees, is running in a central server computer called opensees.stanford.edu. The *ElasticBeamColumn* element is built in the form of a distributed element service, which is running on a computer named galerkin.standford.edu. In order to use the element in the analysis, it has to be registered with the central server, using a web interface similar to the one shown in Fig. 7, before it can be utilized to provide the service. As we indicated before, users only need to know the location of the central server without the awareness of the underlying distributed framework; the users can communicate with the server via a web-based interface. The input Tcl file is submitted to the server using a web-based interface (as illustrated earlier in Fig. 5(a)). Upon receiving the request, the central server starts a new process to perform the nonlinear dynamic analysis. When the analysis is in need of certain type of element (in this case, *ElasticBeamColumn* element), the *Registration and Naming Service* will be consulted to query the element service. Once the communication between element service and the central server is established, the analysis can continue as if the element resides in the central server.

During the simulation, some selected analysis results will be saved in the Database, and certain information will be returned to the user's browser to inform the progress of the simulation (similar to Fig. 5(b)). After the simulation is complete, typical analysis results can be queried, or the results can be downloaded from the server and saved in a data file. To facilitate plotting of the analysis result in a user's web browser, a MATLAB-based distributed post-processing service is deployed in this example. For example, if the user wants to plot response time history of node 1 (which is the left node on the 18th floor in the structural model), the central server will forward the time history response data to the MATLAB-based service running on a separate com-

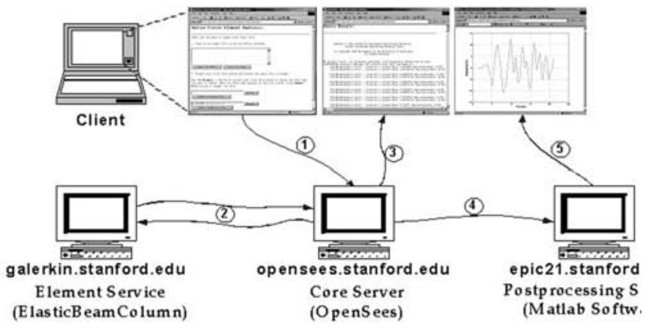


Fig. 14. Interaction of distributed services.

puter (in this case, epic21.standford.edu). Once the post-processing service receives the request, it automatically starts a MATLAB process to plot the time history response and then save it in a file of PNG (Portable Network Graphics) format. This file can later on be sent to the client and be plotted in the user's browser, as shown in Fig. 15.

In this example, the simulation was the result of the collaboration among four computers and several services running on these computers. The services are distributed on different computers on the Internet, residing within their own address space outside of the central server, and yet they appear as though they were local to the client.

6. Conclusions

This paper has described an Internet-enabled distributed service architecture that allows new services to be incorporated into a modular nonlinear dynamic analysis platform. The main design principle of this collaborative framework is to keep the kernel flexible and extendible. A diverse group of users and developers can easily access the platform and attach their own developments to the core. By providing a modular infrastructure, services can be added or updated without recompilation or reinitialization of the existing services. Although this work has focused on new element services, other analysis services (e.g. material services, solution strategies services, analysis strategies services, etc.) can be implemented and linked to the modular server in the similar fashion.

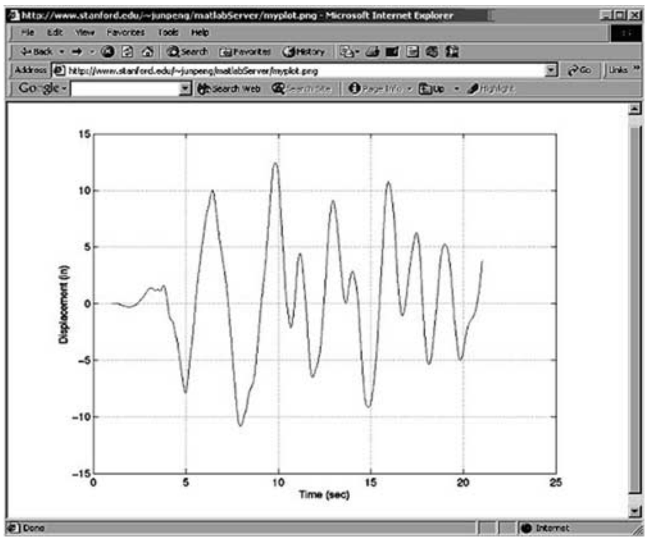

http://www.stanford.edu/~junpeng/matlabServer/myplot.png - Microsoft Internet Explorer
Displacement (in)
Time (sec)
15
10
5
0
−5
−10
−15
0
5
10
15
20
25

Fig. 15. Graphical response time history of Node 1.

This research has illustrated the many possibilities of the Internet for enhancing the distributive and collaborative software development and utilization. The Internet-enabled collaborative system implementation of engineering software for analysis and simulation has at least three benefits. First, the platform provides a means of distributing services in a modular and systematic way. Users can select appropriate services and can easily replace a service by another one, without having to recompile the existing services being used. Secondly, it provides a means to integrate legacy code as one of the modular services in the infrastructure. Thirdly, the framework alleviates the burden of managing a group of developers and their source code. Once a common communication protocol is defined, participants can write the code based on the protocol and there is no need to constantly merge the code written by different participants.

The collaborative prototype described in this paper does not address issues of authentication and security. These security issues could be addressed in the network level, especially by utilizing the Public Key Infrastructure (PKI) that supports digital signatures and other public key-enabled security services [23]. Another issue of the framework is scalability. The current implementation relies on Java's multithreading feature to handle simultaneous requests. Our test result shows that the performance will be substantially degraded when more than a dozen clients access the server simultaneously. This scalability problem could be tackled by providing multiple core servers, utilizing more powerful computers, and deploying parallel and distributed computing environment [24,25].

Acknowledgements

The authors would like to thank Dr. Frank McKenna and Prof. Gregory L. Fenves of UC, Berkeley for their collaboration and enthusiastic support of this research. This work was 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 authors would also like to acknowledge the *Technology for Education 2000* equipment grant from Intel Corporation.

References

1. Forde, B. W. R., Foschi, R. O., Stiemer, S. F. (1990) Object-oriented finite element analysis. Computers and Structures, 34(3), 355–374

2. Miller, G. R. (1991) An object-oriented approach to structural analysis and design. Computers and Structures, 40(1), 75–82
3. Mackie, R. I. (1992) Object-oriented programming of the finite element method. International Journal for Numerical Methods in Engineering, 35(2), 425–436
4. Zimmermann, T., Dubois Pelerin, Y., Bomme, P. (1992) Object-oriented finite element programming: I. Governing principles. Computer Methods in Applied Mechanics and Engineering, 98(2), 291–303
5. Kong, X. A., Chen, D. P. (1995) An object-oriented design of FEM programs. Computers and Structures, 57(1), 157–166
6. Archer, G. (1996) Object-oriented nonlinear dynamic finite element analysis. PhD thesis, Department of Civil and Environmental Engineering, University of California, Berkeley, CA
7. Rucki, M. D., Miller, G. R. (1996) Algorithmic framework for flexible finite element-based structural modeling. Computer Methods in Applied Mechanics and Engineering, 136(3–4), 363–384
8. McKenna, F. (1997) Object-oriented finite element programming: Frameworks for analysis, algorithm and parallel computing. PhD thesis, Department of Civil and Environmental Engineering, University of California, Berkeley, CA
9. Ones, S. R., De Santiago, E. (2000) An object based application of distributed programming for turbulent flow problems. Fourteenth Engineering Mechanics Conference ASCE, Austin, TX
10. Commend, S., Zimmermann, T. (2001) Object-oriented nonlinear finite element programming: A primer. Advances in Engineering Software, 32(8), 611–628
11. Dubois-Pelerin, Y., Zimmermann, T. (1993) Object-oriented finite element programming: III. An efficient implementation in C++. Computer Methods in Applied Mechanics and Engineering, 108(1–2), 165–183
12. Han, C. S., Kunz, J. C., Law, K. H. (1999) Building design services in a distributed architecture. Journal of Computing in Civil Engineering, 13(1), 12–22
13. Peng, J., Law, K. H. (2000) Framework for collaborative structural analysis software development. Structural Congress & Expositions ASCE, Philadelphia, PA
14. Peng, J., McKenna, F., Fenves, G. L., Law, K. H. (2000) An open collaborative model for development of finite element program. The 8th International Conference on Computing in Civil and Building Engineering (ICCCBE-VIII), Palo Alto, CA, 1309–1316
15. Smith, B. L., Scherer, W. T. (1999) Developing complex integrated computer applications and systems. Journal of Computing in Civil Engineering, 13(4), 238–245
16. Hopkins, J. (2000) Component primer. Communications of the ACM, 43(10), 27–30
17. Plasil, F., Visnovsky, S., Besta, M., (1999) Bounding component behavior via protocols. TOOLS USA 1999: 30th International Conference & Exhibition, Santa Barbara, CA, 387–398
18. Lewandowski, S. M. (1998) Frameworks for component-based client/server computing. ACM Computing Surveys, 30(1), 3–27
19. Ousterhout, J. K. (1994) Tcl and the Tk Toolkit, 1st Ed. Addison-Wesley
20. Farley, J. (1998) Java Distributed Computing, O'Reilly & Associates, Sebastopol, CA
21. Phillipsen, M., Haumacher, B., Nester, C. (2000) More efficient serialization and RMI for Java. Concurrency: Practice and Experience, 12(7), 495–518
22. Govindaraju, M., Slominski, A., Choppella, V., Bramley, R., Gannon, D. (2000) Requirements for and evaluation of RMI protocols for scientific computing. High Performance Networking and Computing Conference (SC 2000), Dallas, TX
23. Stallings, W. (1998) Cryptography and Network Security: Principles and practice, 2nd Ed. Prentice Hall, NJ
24. Law, K. H., Mackay, D. R. (1996) A parallel implementation of a generalized Lanczos procedure for structural dynamic analysis. International Journal of High Speed Computing, 8(2), 171–204
25. De Santiago, E., Law, K. H. (2000) A distributed implementation of an adaptive finite element method for fluid problems. Computers and Structures, 74, 97–119