An Open Collaborative Model for Development of Finite Element Program

Jun Peng[1], Frank McKenna[2], Gregory L. Fenves[3] and Kincho H. Law[4]

Abstract

An open collaborative model for development of structural analysis program is presented in this paper. The main design principle of this framework is to keep the kernel flexible and extendible so that the researchers and engineers can easily incorporate new element technologies and solution strategies. The distributed element service over the Internet shows that the open collaborative model could be a powerful tool for incorporating legacy code as well as new developments.

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. In current engineering practice, finite element packages need to be able to accommodate new advances in element and material formulations, solution strategies and computing environments. However, most existing structural analysis programs bundle all the procedures and program kernels into a software package that are developed by individual organizations. Extending these programs to incorporate new developments is a difficult process and more importantly, there is no easy way to link custom components developed by users and researchers separately outside the organization.

---

[1]Graduate Student, Dept. of Civil and Envir. Engrg., Stanford Univ., Stanford, CA 94305, junpeng@leland.stanford.edu

[2]Postdoctoral Research Fellow, Dept. of Civil and Envir. Engrg., Univ. of California, Berkeley, CA 94720, fmckenna@ce.berkeley.edu

[3]Professor, Dept. of Civil and Envir. Engrg., Univ. of California, Berkeley, CA 94720, fenves@ce.berkeley.edu

[4]Professor, Dept. of Civil and Envir. Engrg., Stanford Univ., Stanford, CA 94305, law@cive.stanford.edu

With the maturation of information and communication technology, the concept of building collaborative systems to distribute the services over the Internet is becoming a reality. The open source software development process reflects a powerful global trend toward networked collaboration (O'Reilly 1999). Structural analysis programs can also be developed in an open collaborative, highly user-involved fashion.

Unlike traditional packaged structural analysis programs, the collaborative model could potentially reduce the overhead of continuous upgrade and extension. For the users, they can select appropriate services and can easily replace a specific module if superior module becomes available. For the developers, they can concentrate on developing the components and can easily integrate the components to the core. This paper focuses on a prototype implementation of distributed element service, which illustrates an open collaborative model for development of structural analysis software over the Internet.

The Architecture of Collaborative Model

Figure 1 shows the system architecture of the collaborative framework for developing structural analysis software. The mechanics of the collaborative system is depicted in Figure 2. In this model, object-oriented kernel has been used to provide the maintainability and extendibility essential for software packages (McKenna 1997). This is due to the support provided in object-oriented languages for abstraction, encapsulation, modularity and code reuse.

In this framework, the users build their structural model by using model-building services on the client site. The finished model can be sent to the analysis core by using the Internet or other types of computer networks. Upon receiving the analysis model, the core server then performs the analysis in a collaborative environment. The server can take advantage of distributed and/or parallel computing platforms and thus allow the solution of large sized problems to be completed within a reasonable amount of time (Santiago 1996, McKenna 1997). A stand-alone database is used for efficient data accessing and for post-processing. During the analysis, some elements can be accessed locally from the core element library and some elements can be obtained from on-line element service providers. The reason that these element services can be found by the core server is that before the analysis is performed, the element services can be pre-registered with the core server. When the server needs to use these elements, the registry will be looked up and the requests can be forwarded to appropriate site.

This paper focuses on the components to support two types of usages: (1) routine users who are interested in performing linear, nonlinear and dynamic analyses, and (2) element technology developers.

1. Users 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 incorporated into the platform.

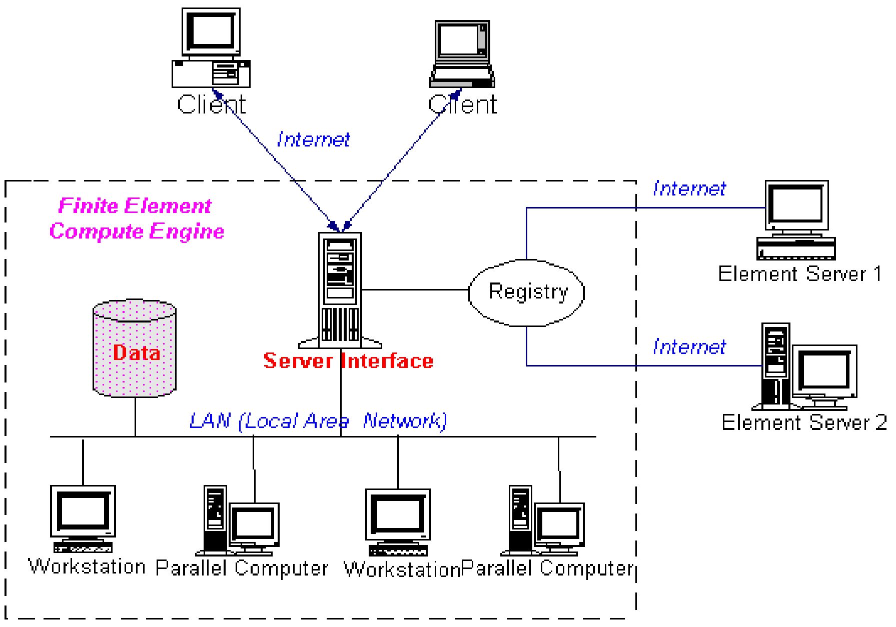


Figure 1: Collaborative System Architecture for Finite Element Software

2. For element technology developers, a standard interface/wrapper will be 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 element library. However, 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.

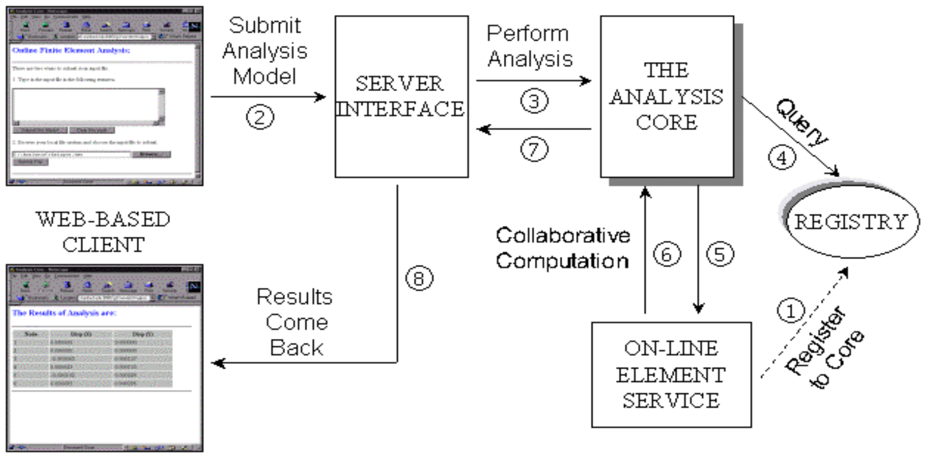


Figure 2: Mechanics of the Collaborative Model

The Object-Oriented Analysis Core

The Analysis core as shown in Figure 2 has an object-oriented modular architecture, which allows users to easily incorporate their developments and extend the system without having to change the existing core functionality. The main classes in the core can be divided into five groups:

1. Modeling: classes to create the finite element model for a given problem. The analyst interacts with a *ModelBuilder* object to create the *Elements*, *Nodes*, *Loads* and *Constraints* objects that define the model.
2. Finite Element Model: classes that describe the finite element model and to store the results of an analysis on this model. Typically a *Domain* object is used as a container object to store the *Node*, *Element*, *Load* and *Constraint* objects created by the *ModelBuilder*.
3. Analysis: classes that perform the analysis of the finite element model, i.e. form and solve the governing equations. In the core, an *Analysis* object is an aggregation of objects of the following types: *SolutionAlgorithm*, *Integrator*, *ConstraintHandler*, *DOF_Numberer*, *SystemOfEqn*, and *AnalysisModel*. The *AnalysisModel* is a container for the *FE_Element* and *DOF_Group* objects that created by the *ConstraintHandler*.
4. Numerical: classes that pass information between objects and to handle the numerical operations in the solution procedure. The numerical classes provided by the G3 core include *Matrix*, *Vector*, *SystemOfEquations* and *Solver*.
5. Monitor: classes that monitor the solution progress and results of an analysis. These include *Recorder*, *Database* and *Renderer* classes.

Distributed Element Service

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 the object-oriented finite element analysis kernel. After the development process is finished, the new element can be migrated into the core platform and become part of the element library.

In addition to the element library provided by the analysis core, 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. Whenever the user wants to use this element, the request will be forwarded to the service provider and the meaningful computation is performed at the service provider's site. This is more or less like a web server; the only difference in this instance is that the web presents primarily static information (typically HTML files), while the compute engine has programmed functionality that can generate dynamic responses for different requests.

To standardize the implementation of a new element, a common interface named *ElementRemote* is provided, as shown in Figure 3. 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 Figure 3 for clarity.

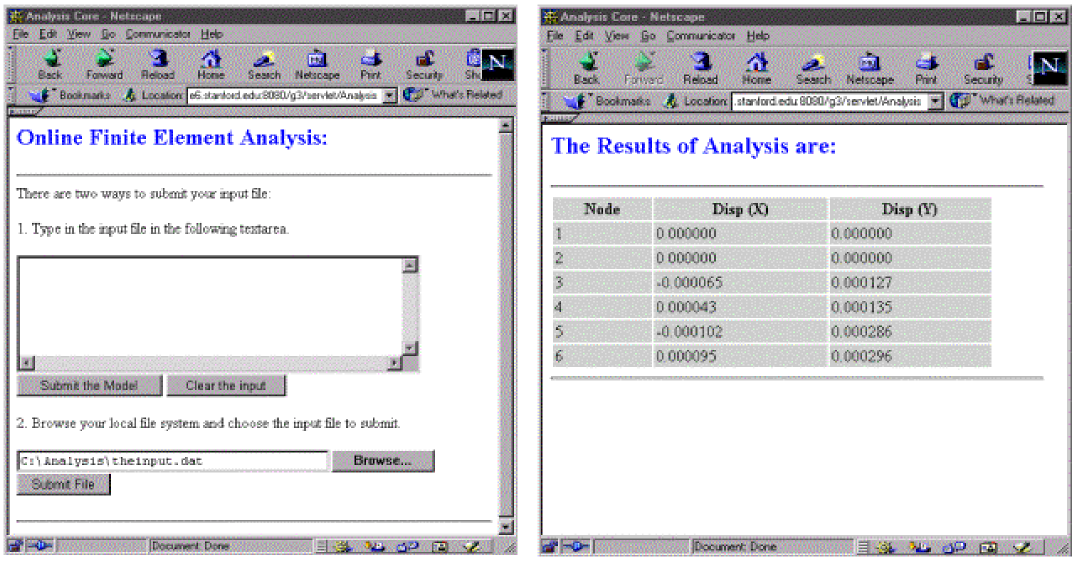
```
public class ElementRemote extends Remote {
    // This is the service name for publishing.
    public static final String SERVICE = "ElementService";
    // This is the port number, could be changed as needed.
    public static final int PORT = 12345;

    // 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);
}
```

Figure 3: Class Interface of ElementRemote

The Prototype Implementation

To illustrate the collaborative model, this section presents a simple prototype implementation. In the prototype, *http* is used for the communication between end users and analysis core. The finite element model is represented in a text file and this file can be submitted to analysis core through a web interface. After the server finishes the analysis, the results return to the client in the form of a generated web page. Figure 4 shows the web interface for file submission and the sample web page generated by the core server.

In the prototype implementation as depicted in figure 5, Java's *Remote Method Invocation (RMI)* is used to provide the network communication between the analysis core and the online element service. RMI enables a program in one Java Virtual Machine (JVM) to make method calls on an object located on a remote server machine. 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.


Analysis Core - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks Location: e6.stanford.edu:8080/g3/servlet/Analysis What's Related
Submit the Model
Clear the input
Browse...
Submit File
Document: Done

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.

```
C:\Analysis\theinput.dat
```

The Results of Analysis are:

Node	Disp (X)	Disp (Y)
1	0.000000	0.000000
2	0.000000	0.000000
3	-0.000065	0.000127
4	0.000043	0.000135
5	-0.000102	0.000286
6	0.000095	0.000296

Figure 4: Web Pages Generated in the Client Site

Figure 5 shows how the element service works by using Java RMI as the communication layer and using Java Native Interface (JNI) to link the legacy code. The *Analysis Core* is connected with *ElementServer* through *ElementClient*. The computation code that generates the element stiffness matrix, the mass matrix etc. is wrapped by *ElementServer*.

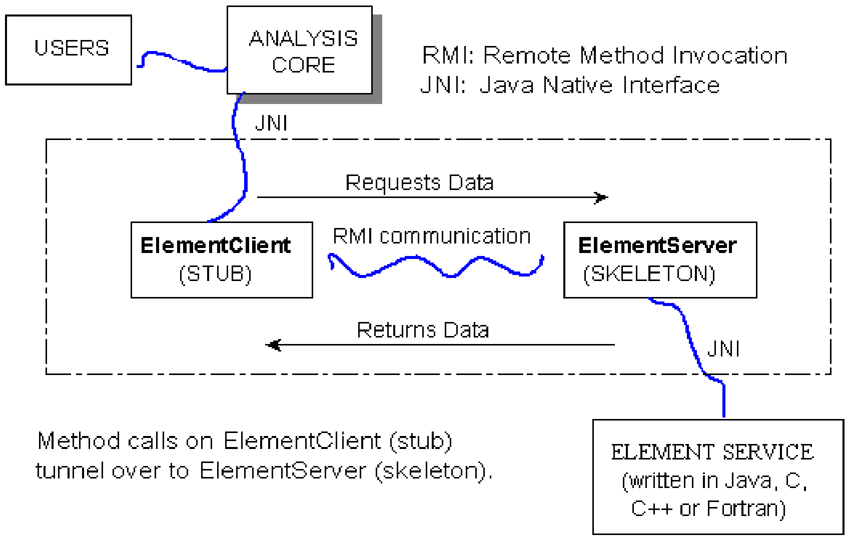

USERS
ANALYSIS CORE
RMI: Remote Method Invocation
JNI: Java Native Interface
JNI
Requests Data
ElementClient (STUB)
RMI communication
ElementServer (SKELETON)
Returns Data
JNI
Method calls on ElementClient (stub) tunnel over to ElementServer (skeleton).
ELEMENT SERVICE (written in Java, C, C++ or Fortran)

Figure 5: Distributed Element Service by using Java RMI

When the users want to use this new element, they can instantiate and make method calls to this element in the same way they treat the local element class. The actual method calls will call on *ElementClient* and tunnel over to *ElementServer*. Figure 6 shows some sample code that illustrates the usage of 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 calculate the element stiffness matrix. After the compute engine finishes the computation, the stiffness matrix is saved in a hashtable. The next time when the *Client* asks for the stiffness matrix by using *getStiff()* method, the hashtable will be looked up and the data can be returned to the caller immediately.

```
public class QuadElementClient extends ElementClient {
   /* based on the server name and port number, create stub object. */
   public QuadElementClient(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(String tag, String input)
   {
      theStub.formElement(tag, input);
   }

   public MyMatrix getStiff(String tag)
   {
      MyMatrix result = new MyMatrix(8, 8);
      result = theStub.getStiff(tag);
      return result;
   }
}
```

```
public class QuadElementServer extends UnicastRemoteObject
             implements ElementRemote {
   /* use the hashtable to hold all the elements */
   private Hashtable allElements = new Hashtable();
   public void formElement(String tag, String input)
   {
       QuadElement newElement = new QuadElement(tag, input);
       allElements.put(tag, newElement);
   }
   public MyMatrix getStiff(String tag)
   {
       QuadElement oneElement = (QuadElement)allElements.get(tag);
       result = oneElement.getStiff();
       return result;
   }
}
```

Figure 6: Sample ElementClient and Sample ElementServer

Conclusions

This paper focuses on one domain (in this case, the distributed element service) of the networked collaborative development of structural analysis program. The principle of distributed services can be applied to other domains of structural analysis program, for example, distributed solution strategy, distributed material etc. The collaborative model 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.

The collaborative system implementation of a structural analysis program has at least three benefits. First, the framework provides a means of distributing services in a modular and systematic way. Users can select appropriate services and can easily replace a service if a superior service becomes available, without having to recompile the existing services being used. Second, it provides a means to integrate legacy code as one of the modular services in the infrastructure. Third, 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 their code based on the protocol and there is no need to constantly merge the code written by different participants.

Acknowledgements

This research is partially sponsored by the Pacific Earthquake Engineering Research (PEER) Center. The authors would also like to acknowledge the "Technology for Education 2000" equipment from Intel Corporation which is being used in this research.

References

Han, C. S., Kunz, J. C. and Law, K. H. (1999). "Building Design Services in a Distributed Architecture." *Journal of Computing in Civil Engrg.,* 13(1), 12-22.

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.

O'Reilly, T. (1999). "Lessons From Open Source Software Development." *Communications of the ACM,* April 1999, 42(4), 33-37.

Santiago, E. De. (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.*