Framework for Collaborative Structural Analysis Software Development

Jun Peng and Kincho H. Law
Stanford University

Abstract

This paper outlines a software framework that will serve as the core for collaborative structural analysis software development over the Internet. The collaborative framework would allow researchers and engineers to easily access the analysis platform and to incorporate new element technologies and solution strategies for structural analysis. The objective is to provide a "plug-and-play" environment where users can define and specify their own model, element types, and preferred solution techniques, and possibly the strategy most appropriate for solving their specific problem.

Introduction

It is well recognized that a significant gap exists in the state-of-art computing methodologies and the state-of-practice in structural engineering analysis programs. In order for analysis programs to continuously upgrade to keep up with new computing technology and incorporate new development, not only does the system need to be designed in a flexible and extendible way, but also a mechanism must be provided for the easy integration of new developments. However, most existing structural analysis programs are bundled in a software package and developed by individual organization. There is no easy way to incorporate new developments from users and researchers outside the organization.

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. A collaborative system is one where multiple users or agents engage in a shared activity, usually from remote locations. This paper describes a prototype implementation to illustrate a collaborative framework for structural analysis software development over the Internet.

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.

The Collaborative Framework

Figure 1 shows the system architecture and the research approach in developing the collaborative framework for structural analysis software development. 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 computer networks. The server then performs the analysis in a collaborative environment. The server can take advantage of distributed and/or parallel computing 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, elements can be accessed locally from the core element library; furthermore, additional elements can be obtained from on-line element service providers.

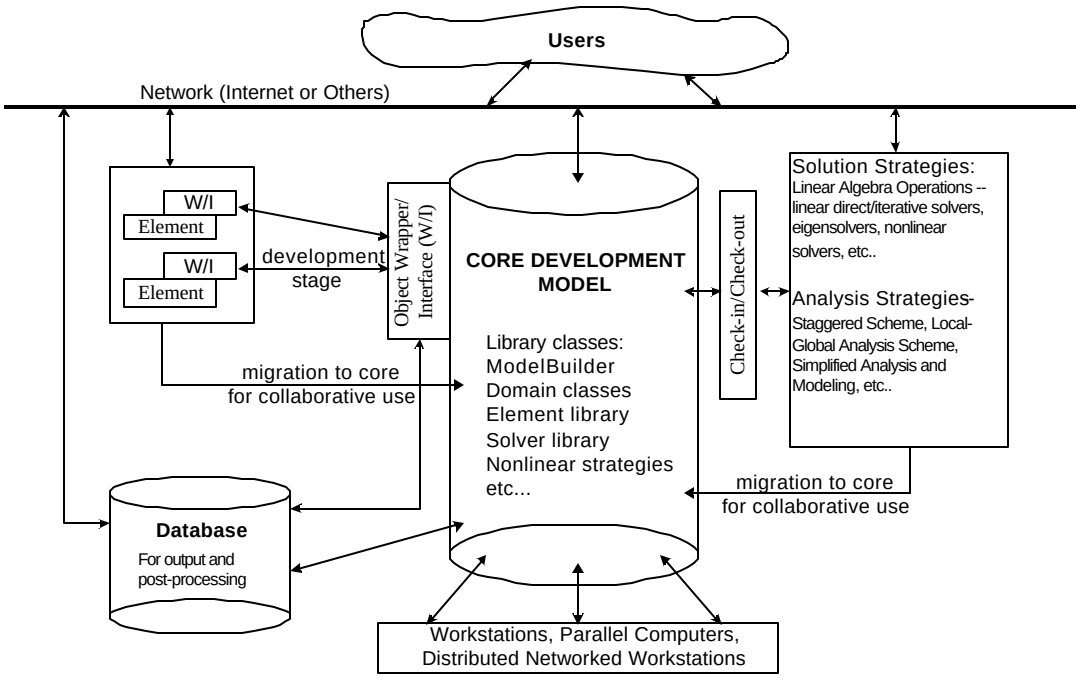

Figure 1: A Collaborative Framework for Finite Element Analysis

The collaborative framework is intended to support at least three types of usages: (1) routine users who are interested in performing linear, nonlinear and

dynamic analyses, (2) element technology developers, and (3) analysts who are interested in developing new analysis and solution strategies.

1. Users can have direct or remote access (possibly via 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 and incorporated into the platform.
2. For element technology developers, a standard interface/wrapper will be defined for communicating the element(s) with the core program. The element(s) can be written in languages such as C, Fortran, C++ and/or Java. When the developer and the system administrator agree to migrate the element(s) into the analysis core, the element(s) can be registered to the core and can be accessed remotely over the network.
3. For analysts who are interested in developing new analysis and solution strategies, a check-in/check-out paradigm that is commonly used in database and CAD environment will be extended to support the program development process (Krishnamurthy 1996). While element development may simply involve appending new elements into the element class library, analysis and solution strategies may require changes (e.g. adding new classes) in the class libraries. The check-in/check-out strategy may be appropriate since this will allow the analyst to fine-tune the class library for its best use. When the developer and the system administrator agree to migrate the procedures into the core platform, the core will be appropriately re-configured to adopt the new procedures, which will then be made available to other users.

Maintaining Service Identity. 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.

Distributed Element Service. For the new element developers, a standard interface/wrapper is defined for communicating the element with the analysis core. If the developer and the core administrator agree, the new element can be migrated to the core platform and become part of the element library. However, the developer can also choose to be an on-line element service provider. In this case, whenever the user wants to use this element, the request will forward to the service provider and the actual computation will be performed at the service provider's site.

In the prototype implementation, Java's *Remote Method Invocation (RMI)* is used to provide 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.

To standardize the implementation of a new element, a common interface of *ElementRemote* is provided, as shown in Figure 2. Element developers will implement an *ElementServer*, which is the subclass of *ElementRemote*. There are two important methods 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 housecleaning 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 should perform exception processing; they are ignored in Figure 2 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 = 1234;

       // This function is used to send the element data to server.
       public void formElement(int tag, Identity src, String input);
       // When the analysis finished, use this function to do housecleaning.
       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 2: Class Interface of ElementRemote

Figure 3 shows how the element service works by using Java RMI. The *Analysis Core* is connected with *ElementServer* through *ElementClient*. The actual code that generates the element stiffness matrix, the mass matrix etc. is wrapped by *ElementServer*. After the element developer finishes the new element implementation, the new element can be registered to the structural analysis core.

This step will include sending the *ElementClient (stub)* to the *Analysis Core*. After the registration, this element service becomes known by other users and is ready for use.

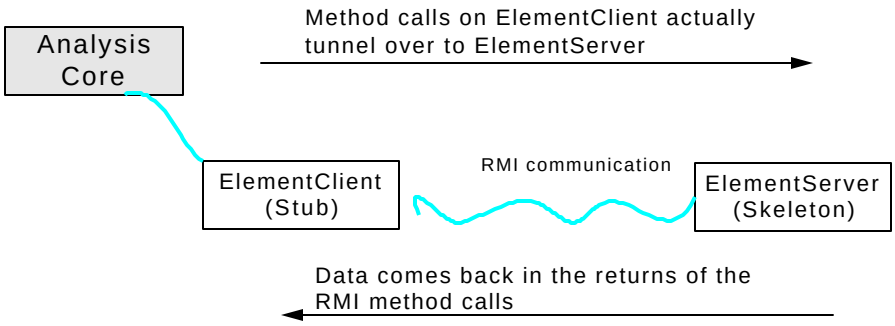

Figure 3: Element Service by using Java RMI

When the users want to use this new element, they can instantiate and make method calls to this element in the same way as they treat the local element class. The actual method calls will call on *ElementClient* and tunnel over to *ElementServer*. After *ElementServer* finishes the computation, the data returns to the caller, as shown in Figure 3.

Database Interface. The *database* refers to a collection of data that is managed by a database management system (DBMS). The DBMS can provide: data definition (define the data that will be in the database), data manipulation (retrieve, update and add data to the database), security and integrity (monitor access to database and check requests for correctness), data recovery (for recovery of data in case of a system failure), concurrency (managing multi-user access to the data), and a data dictionary (information about the data in the database). In this framework, an object-relational database (ORDB) is employed for the efficient accessing of large structured data, like matrix and vector.

For the *Analysis Core*, a standard interface for an ORDB is provided. This interface creates the object-oriented front-end that will allow the storage and retrieval of basic information on the *Node*, *Element*, *Load* and *Constraint* objects. In addition it will allow for the storage of the basic types: *ID*, *Vector*, *Matrix*, and *Message*. The advantage of this generic database interface is that new classes can be introduced without the creator of these classes being responsible for creating tables in the database and they can access database using inherited methods. The disadvantage of this approach is that no information regarding the schematic meaning of the data will exist within the database. To determine the meaning of the data, the data is retrieved from the database by the objects that placed the data there and the objects themselves must be queried for their data.

The *FE_Datastore* class is introduced into our design to provide the interface to a database. The introduction of this class into the framework removes from the program developers the need to program using a DBMS language and can allow multiple DBMSs to be used. The *FE_Datastore* class is an abstract class, it defines the methods that must be implemented for each new DBMS that is used with the system, and it is only the *FE_Datastore* object that interfaces with the DBMS.

A *FE_Datastore* object is associated with a *Domain* object. The interface of *FE_Datastore* is shown in Figure 4. During a *commitState()* the FE_Datastore object first ensures that the information in the base tables is correct. It then iterates over all the component objects (*Element*, *Node*, *Load*, *Constraint*) in the *Domain* and invokes *sendSelf()* on these objects. These objects use the methods for sending *ID*, *Vector*, *Matrix* and *Message* objects to send the data to the database. On a *retrieveState()* the *FE_Datastore* object first ensures that the objects exist, otherwise, it will create new data objects of the correct type and add them to the Domain. It then iterates over the components, which invoke the *recvSelf()* method to get the data from database.

```
class FE_Datastore : public ModelBuilder, public Channel
{
   public:
       FE_Datastore(Domain &theDomain, FEM_ObjectBroker &theBroker);
       virtual ~FE_Datastore();

       // pure virtual functions in addition to those defined
       // in the ModelBuilder and Channel classes
       virtual int getDbTag(void) =0;
       virtual int commitState(int commitTag);
       virtual int restoreState(int commitTag);

       virtual int validateBaseRelationsWrite(int commitTag)=0;
       virtual int validateBaseRelationsRead(int commitTag) =0;

};
```

Figure 4: Interface for **FE_Datastore** Class

As mentioned earlier, the data must be retrieved from the database by the objects that placed the data. For example, to retrieve the data regarding element object, it is up to that object to look for the data from the core memory and from the database, or based on the retrieved data to re-compute the required data. The pseudo code for querying element information is shown in Figure 5.

```
Info *getInfo(char *request, int commitTag)
{
       interpret the request;
       regarding different request, looking for Info;

       if (Info is in memory)            return Info;
       else
              contact FE_Datastore to call restoreState(commitTag),
              this brings element to the proper time step state.

              if (Info defined in table) return Info;
              else re-compute Info and return.
}
```

Figure 5: Pseudo Code for Querying Element Information

Security and Revision Control. For the collaborative computing system, accessing to the resources associated with the system needs to be restricted. Furthermore, since the system is continuously changing with new developments, we need an efficient way to develop a degree of control over what changes are made to the system, and keep track of previous changes to the system. Regarding the security and revision control, at least we will face the following challenges.

1. Only the registered users and developers can have access to the system; in other words, the system needs to be secured.
2. Developers should be able to check in and check out their developments easily.
3. After the developer adds a new feature, this feature needs to be easily integrated into the core and the users should be informed about the changes.
4. For minor modification, for example, fixing a bug in the system, there must be an easy way to inform other developers. In addition, their version should be easily and quickly updated.

For the security and reversion control of this collaborative framework, a web-based interface is provided. The web site consists of a broker and the repository of all the source code and documentation. Java Servlet is chosen as the web development tool because it provides web developers with a simple, consistent mechanism for extending the functionality of a web server. Written in Java, Servlets can access to the entire family of Java APIs, thus additional functionality and security rules of the server can be easily integrated. The security and revision control of the collaborative structural analysis system has the following features:

1. The access to the source code is protected by encryption and password. The system provides a registration form to gather the information about the users and developers.
2. The broker is introduced to manage the services. The developers can check in/check out the services over the Internet through this broker. In the prototype implementation, the broker interface defines only two methods, *register()* and *query()*. When a service registers its services with the broker, it provides its *Identity* and a string that describes the service. When the broker is queried for a service, the broker returns a registered service that matches the description of the query argument.
3. The server keeps a log of all the users. This makes it very easy to keep track of the version that the user has. Since the log is kept in the server, when the users log in, they will see what files have been updated and what new services have been registered after the last time they logged in. A mailing list is also kept in the server, so when there's some major changes to the source code, an email will be sent to all the users and developers to inform them about the changes.
4. Since the source code is maintained in a central server, the consistence of the code is very easy to ensure. After a developer fixes a bug, he can check in this bug fix. This minor change will send to all the users and developers by using email. Furthermore, the change will be posted in the web-server as a patch to the system.

Conclusions

This paper has introduced a software framework that will serve as the core for collaborative structural analysis program development. The main design principle of this core 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.

Unlike the 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 service if superior service becomes available. For the developers, they can concentrate on developing the components and easily provide their services to the core. Since the source code is controlled by the developers themselves, the code is very easy to maintain and upgrade.

Acknowledgement

This research is partially sponsored by Pacific Earthquake Engineering Research (PEER) Center (http://peer.berkeley.edu/). The authors would also like to acknowledge a "Technology for Education 2000" equipment grant from Intel Corporation in support of 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 Engineering, 13(1), 12-22*.

Krishnamurthy, K. (1996). "A Data Management Model for Change Control in Collaborative Design Environments." *Ph.D. Thesis, Department of Civil Engineering, Stanford University, Stanford, CA.*

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.

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*.

Authors

Jun Peng, Ph.D. Student, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305. E-mail: junpeng@leland.stanford.edu.

Kincho H. Law, Professor, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305. E-mail: law@cive.stanford.edu