Software framework for distributed experimental-computational simulation of structural systems*

Yoshikazu Takahashi and Gregory L. Fenves

1 Department of Urban Management, Kyoto University, Yoshida Honnachi, Sakyo-ku, Kyoto, 606-8501, Japan
E-mail: yos@catfish.kuciv.kyoto-u.ac.jp
2 Department of Civil and Environmental Engineering, University of California, Berkeley, Berkeley, CA 94720-1710, USA

SUMMARY

Supported by the recent advancement of experimental test methods, numerical simulation, and high-speed communication networks, it is possible to distribute geographically the testing of structural systems using hybrid experimental-computational simulation. One of the barriers for this advanced testing is the lack of flexible software for hybrid simulation using heterogeneous experimental equipment. To address this need, an object-oriented software framework is designed, developed, implemented, and demonstrated for distributed experimental-computational simulation of structural systems. The software computes the imposed displacements for a range of test methods and coordinates the control of local and distributed configurations of experimental equipment with OpenSees APIs. The object-oriented design of the software promotes the sharing of modules for experimental equipment, test setups, simulation models, and test methods. The communication model for distributed hybrid testing is similar to that used for parallel computing to solve structural simulation problems. As a demonstration, a distributed pseudodynamic test was conducted using a client-server approach, in which the server program controlled the test equipment in Japan and the client program performed the computational simulation in the United States. The distributed hybrid simulation showed that the software framework is flexible and reliable.

KEY WORDS: Object-oriented software, Pseudodynamic test method, Software framework, Distributed experimental test methods, Hybrid experiments

1. INTRODUCTION

Experimental testing of structural systems is essential for improving knowledge about component and system performance in earthquakes. Shaking table testing can provide important experimental data about critical issues such as the effect of component damage on system response, collapse mechanisms, residual deformation and post-earthquake capacity. For example, the new E-Defense facility is a 20 m × 15 m six degree-of-freedom shaking table at the National Research Institute for Earth Science

*This paper is described on the basis on the EESD paper [1] (now printing)

Contract/grant sponsor: the Ministry of Education, Science, Sports and Culture, Japan; contract/grant number: Grant-in-Aid for Young Scientists (A) 16686029
and Disaster Prevention (NIED) in Japan will provide significantly increased capability for structural experiments. Even with this facility, most structural systems are too large to test at or near full-scale. With recent improvement of the Internet all over the world, a geographically distributed testing allows researchers to combine the capabilities of two or more sites to conduct tests on structural systems that could not be performed at any one site because of capacity limits. Distributed hybrid tests have been conducted between Japan and Korea [2]. In the United States, the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) is a network of experimental sites and computing resources connected by the NEESgrid system using Grid-based middleware [3]. Under the NEES program, the Multi-site Online Simulation Test (MOST) demonstrated the use of NEESgrid for hybrid simulation, connecting two experimental sites and one or more computational sites. It showed the potential for distributed testing, but the software architecture included a coordinator that resulted in an artificial separation between the simulation model and the time integration procedure for solving the governing equations of motion.

Even with these recent advances, structural testing has typically been conducted using customized software that is dependent on the configuration of an experiment and the computational procedure for the test method. Customized software, however, is difficult to adapt to other experiments, particularly when multiple sites need to communicate in a distributed test. For structural simulation, object-oriented methodologies were introduced in the beginning in 1990s. Fenves [4], Baugh and Rehak [5] were among those who emphasized the importance of abstraction in engineering software development and advocated object-oriented approaches. The first object-oriented analysis application was the linear and static finite element method. In 1990, Forde et al. [6] defined classes such as Node, Material, and Element, for linear analysis. In the past decade, many researchers have developed object-oriented software for finite element analysis [7, 8]. In the paper, Using object-oriented software design methodologies, a framework of cooperating software classes is developed for a variety of experimental and computational approaches, allowing mixing of computational and experimental elements with communication between the two over a network.

2. SOFTWARE REQUIREMENTS FOR STRUCTURAL TEST SYSTEMS

Structural testing involves imposing displacement or force boundary conditions on a specimen according to a test method and a loading protocol. There are many types experimental setups for applying boundary conditions. The actuators are controlled by a control system, but most software for computing the control signals is customized for an experimental site and test method. As a consequence of the specialized software, it is difficult to develop and implement new test methods, such as hybrid testing or geographically distributed testing, or to exchange software from one laboratory to another. Prior to developing a software architecture for structural test systems to address this problem, the requirements for test methods are summarized.

2.1. Quasi-static cyclic load test method

Cyclic loading of a specimen at a slow rate with an a priori deformation history is a basic experimental method. In the quasi-static test method, the loading is selected to represent cycles of deformation expected during an earthquake. The experimental setup consists of a specimen, servo-control hydraulic actuators, PID controllers, and control computers. With several actuators it is possible to apply a multi-DOF loading on a specimen with an appropriate control system. The components in the experimental
setup communicate using proprietary or public protocols, such as GP-IB (General Purpose Interface Bus, IEEE 488). The primary requirement is that the software must have the flexibility to interact with a variety of the components in the experimental setup and to specify the boundary conditions through displacements or forces in one or more actuators.

2.2. Pseudodynamic test method

The pseudodynamic test method is a computer–controlled experimental technique for simulating the response of structures. In the pseudodynamic test method, the software is required to solve the equations of motion for the computational model of a structural system using step-by-step time integration. An implicit integration scheme is usually used for the nonlinear equations of motion but an iterative procedure is not desirable for testing, therefore several time integration schemes have been proposed [9, 10]. By utilizing partitioning concepts, the pseudodynamic test method allows testing a component or assembly experimentally, whereas the other portions of the structural system are modeled computationally. In early implementations of the pseudodynamic test method, the actuators were put on hold while the equations of motion were solved to determine the displacement target for the next time step. To mitigate errors caused by hold time, continuous pseudodynamic test methods have been developed with a DSP [11].

The software for the pseudodynamic test method needs to provide a variety of computational models for structural systems, including nonlinear material and geometric behavior. In addition, the test method requires specialized time integration schemes and definitions of boundary conditions for the experimental specimen. To support continuous pseudodynamic test methods, the software must be able to communicate with the DSP in the control system. The software should provide fault tolerance to terminate the test safely in the event of an equipment, communication, or specimen failure.

2.3. Real-time on-line test method

A variety of new types of earthquake mitigation technologies have been developed for controlling structural response. Since many of the devices, such as viscous dampers, have velocity-dependent characteristics, they cannot be tested by the conventional pseudodynamic test method. To test these devices, the real-time on-line test method and the substructured shaking table test method have been developed [12, 13]. The testing must occur at a faster rate than for the pseudodynamic test method with the time scale preserved, which means the software must solve the equations of motion at each time step in real-time. In addition, the high rate of actuation and specimen motion introduces inertia forces that must be compensated for. Finally, velocities must be computed accurately and imposed through the actuators. The software needs to provide these capabilities with high-speed computation and communication.

2.4. Distributed test method

The motivation for the distributed test method is to provide the functionality of the aforementioned test methods but with specimens located at one or more experimental sites. Distributed testing allows multiple sites to collaborate on a test of a structural system that would exceed the capacity available at a single site. In addition to the requirements for the pseudodynamic and real-time test methods, the software must be able to control distributed experimental sites using a communications network. The de facto communications protocol at the transport and network layers is TCP/IP, but the standard protocols have inadequate security for distributed control of experimental equipment. To support distributed
control applications, the NEESgrid Teleoperation Control Protocol (NTCP) provides negotiation, execution, and verification of distributed control actions through a transaction-based protocol. The protocol is reasonably secure through the use of Grid authentication and access control and it is also fault-tolerant in that it does not rely on the underlying transport layer (e.g. TCP) to deliver messages reliably.

3. OBJECT-ORIENTED SOFTWARE FRAMEWORK FOR EXPERIMENTAL METHODS

As described in the previous section, the software requirements for experimental systems are to (1) compute the imposed displacements, and possibly velocity and accelerations, for a specimen based on the test method, (2) communicate the imposed boundary conditions to the control system for the experimental setup, and (3) obtain the resisting forces and displacements from the experimental setup. To address these requirements, an object-oriented software framework for hybrid simulation with a variety of local and distributed test methods is developed. The approach is to define the abstractions of experimental setups for conducting the tests, including the interfaces with the experimental equipment. The communication between single or multiple experimental sites is handled in a uniform manner to support distributed testing. Finally, the experimental software is designed to collaborate with computational simulation software, such as OpenSees, for solving the governing equations of motion for a structure.

3.1. Modeling of experimental systems

As an example of software modeling of experimental systems, Figure 1 shows the important objects for representing experimental setups of the two-DOF displacement-controlled loading on the specimen. The round boxes are objects and the arrows show the flow of the data. The experimental element is a cantilever with the two DOF represented by $d_e$. The software can represent a variety of setups for the experimental element, such as the three alternatives, set1, set2, and set3 shown in Figure 1. In a setup, each actuator is controlled by displacement control signals. The software must transform the cantilever element DOF data to the actuator displacements, which are indicated in the figure for the three setups.

Two abstract classes are defined to provide a flexible representation of test methods. The first one, ExperimentalSetup, transforms the degrees-of-freedom, $d_e$, for an experimental element to a vector, $s_c$, for the control system depending on the geometry and kinematics of the loading system. The second class, ExperimentalControl, represents the control system and is responsible for converting the actuator displacements needed for the DSP and PID controllers. The advantage of this approach is that ExperimentalSetup hides the specifics of the experimental configuration from ExperimentalControl. Since an ExperimentalControl object is hidden from testing applications, it can communicate with the hardware without consideration of the details of the test method.

The class diagram for experimental methods is shown in Figure 2. ExperimentalSetup is an abstract class that represents the setup and configuration of an experiment. ExperimentalControl is responsible for accessing the control system and data acquisition system. The separation of ExperimentalSetup and ExperimentalControl allows specialization of both classes by subclassing them independently, and the communication between the two is done by an association named eControl. This approach is known as a Bridge software pattern [14]. The implementations in subclasses of ExperimentalSetup, such as experimental systems for different numbers and arrangements of actuators, has operators that propose a set of displacements, velocities, and acceleration for the DOF, execute the proposed DOF, get the
Figure 1. Examples of ExperimentalSetup object for loading a cantilever specimen with two DOF

resisting forces that result from the execution, and get the values of the DOF from the data acquisition (DAQ) system. The important operators for ExperimentalControl subclasses are to generate a signal for the control system, perform a control action, signal the DAQ to acquire data, and get the data from the DAQ.

Other classes in Figure 2 provide information for the control of an experimental setup. The specifications of the experimental equipment, such as range, gain and units, are modeled by the EquipmentSpec class. Specific ExperimentalControl subclasses control the equipment according to the assigned EquipmentSpec objects. The software can represent control systems that involve feedback loops of measured data from a data acquisition system. To do this, the class ECMultiControl is a composition of multiple ExperimentalControl objects in which one object controls actuators and the other ExperimentalControl object acquires experimental data. The SignalFilter class is available for filtering signals to deal with errors or noise in the experimental system; a variety of filters are implemented using subclassing.

The advantage of this software architecture is that laboratories can develop ExperimentalControl
3.2. Modeling of distributed experimental sites

The software framework described in the previous subsection defined objects that represent the experimental setup, irrespective of the location of the experiment. To accommodate geographically distributed experimental setups, the class ExperimentalSite is introduced to provide a representation of local or remote experimental sites in a uniform manner.

An important aspect of the OpenSees architecture is the support for parallel computing [15, 16]. The main classes for parallel computing are Actor, Shadow, Channel and MovableObject, as shown in Figure 3. An Actor is an object that executes asynchronously on a remote processor, and a Shadow object represents the remote object in local memory space. A message intended for a remote Actor is sent to the local Shadow object. The Shadow object is responsible for sending and receiving messages to and from the remote Actor(s). The communication between the Actor and the Shadow objects is conducted through a Channel object. The Channel object provides general communication mechanisms between the two objects, and it can be implemented using standard protocols, such as TCP/IP and MPI (message-passing-interface). Domain objects inherited from an Actor and a Shadow, respectively, can communicate with a Channel object. Figure 4 shows the class design to provide communication with an ExperimentalSite, either locally or remotely. A key operator for ExperimentalSite objects is getForce() and Figure 4 shows the pseudo-code that implements it.

For local tests, an instance of the subclass LocalExprSite relates directly to an ExperimentalSetup object. Applications communicate with the ExperimentalSetup object through the LocalExprSite object. A request to get the restoring force for a LocalExprSite object is passed directly as a getForce() operator to the ExperimentalSetup object. For distributed testing, however, an ExperimentalSite object needs to handle the communication between an application that defines the loading, such as through a computational simulation, and a remote experimental setup. This situation is analogous to the distributed computing model in OpenSees with the Actor and Shadow classes. Consequently, the class RemoteExperimentalSetup is introduced as a subclass of Actor to execute on a server for the classes for their control systems and experimental equipment. Laboratories with similar equipment could share the software implementations. For each experimental setup, a subclass of ExperimentalSetup needs to be implemented for the configuration of the actuators and other details of the test setup. Libraries of experimental setup classes provide a starting point for sharing implementations, and because of the defined interfaces, all ExperimentalSetup objects are designed to collaborate with any laboratory’s ExperimentalControl objects.
experimental setup. At another location, a client program has a representation of the experimental site using an instance of \textit{RemoteExprSite}, which is a subclass of \textit{Shadow}. In a client-server approach, the server program waits for requests from clients using a loop operation. The server loop is implemented in \textit{RemoteExperimentalSetup} but almost all other methods are the same as \textit{ExperimentalSetup}. Therefore \textit{RemoteExperimentalSetup} is designed with an Adapter pattern [14] of \textit{ExperimentalSetup} to which is added the communication method for the server to receive commands from the client program. The advantage of this approach is that it uses the distributed computing model in OpenSees, and as a result the software can be used for experimental testing, computational simulation, or a hybrid of the two.
This client-server architecture uses the communication mechanisms in OpenSees with the client, RemoteExprSite, and the server, RemoteExperimentalSetup, communicating through a Channel object. The OpenSees Channel has been implemented using TCP sockets or other protocols such as MPI.

The software framework for experimental test sites is defined by the class diagrams in Figure 2 and Figure 4. To illustrate how the objects interact with each other, sequence diagrams show the messages and flow of information between objects. The sequence diagrams for a local and distributed experimental test are shown in Figure 5 and Figure 6, respectively. These diagrams show three classes and three major methods: set TrialResponse(), getForce(), and getDaqData(). The method set TrialResponse() is used by the application to specify the displacement of a specimen. The LocalExperimentalSite and ExperimentalSetup are responsible for proposing and executing the action through the experimental control. The acquired data are stored in ExperimentalSetup, as shown in
Figure 2. To obtain force data, the method `getForce()` is invoked by the application to get the data from the `ExperimentalSetup` through the `LocalExperimentalSite`. The role of `getDaqData()` is similar to `getForce()` but it is used for acquiring the current data from the data acquisition system. Comparing the sequence diagrams for a local and distributed test, the right-hand side of the `ExperimentalSetup` and `ExperimentalControl` objects are the same. Therefore, experimental sites can have software interfaces for its equipment and test methods without regard to the applications or whether the applications will be run on a local or remote computer. On the other hand, simulation methods can be developed without regard to the experimental setup and can combine computational and experimental elements as described in the next subsection on collaboration with OpenSees.
3.3. Collaboration between experimental systems and OpenSees

As described in Section 2, structural test methods require the software to solve the equation of motion for determining the next step of the loading path. To accomplish this, OpenSees is extended to provide the computational simulation needed for structural testing either locally or in a distributed manner. Figure 7 shows the classes for hybrid simulation, which consists of three parts: OpenSees core, the experimental test system, and the class ExperimentalElement to interface between the two. The OpenSees class Element has methods to compute the resisting force vector for values of the DOF. In a similar manner, an experimental object has methods to set values of the DOF and to obtain the resisting force vector.

An ExperimentalElement object is associated with an ExperimentalSite. Once an ExperimentalElement object defines values of DOF for an ExperimentalSite object, the resisting force vector is obtained through the test system. From the point of the ExperimentalSite’s view, the ExperimentalElement object is the application. On the other hand, a simulation using OpenSees only needs to obtain a element resisting force vector from the ExperimentalSite object regardless of the location of the experimental site. Since the ExperimentalElement acts as an Element from OpenSees’ view, it can be used along with computational elements without changing OpenSees.

Figure 7. Class diagram of collaboration relationships between OpenSees and experimental test system
Figure 8. Sequence diagram for updating the state of an experimental element in one step of time integration

For time integration of the equations of motion, OpenSees has several implicit step-by-step methods, e.g. Newmark $\beta$, Wilson $\theta$ and HHT-$\alpha$. One of the critical requirements for on-line tests is to avoid load reversals during the time required to solve the implicit equations iteratively. This can be achieved using the $\alpha$-Operator Splitting integration method, which has been implemented as an Integrator object in OpenSees.

The collaboration between OpenSees and an experimental system is illustrated in Figure 8, which is the sequence diagram for updating the state of elements in the integration of the equations of motion. OpenSees is responsible for the integration, which coordinates the process, and also the computational model. As a result, a computational simulation and an experimental-computational simulation are conducted in an identical manner. It should be noted that the experimental element itself could be simulated by hiding it as an implementation of an ExperimentalElement; this simulation could be done locally or remotely. This is useful for simulating a hybrid test before the actual experimental equipment and specimen are used, or for further distributing the computational model to take advantage of different simulation software, models, or computational resources.

4. EXAMPLE OF DISTRIBUTED PSEUDODYNAMIC TEST

4.1. Overview

To demonstrate the software framework, a distributed experimental-computational simulation was carried out using the pseudodynamic test method at Kyoto University (KU), Japan, and a computational site at the University of California, Berkeley (UCB), USA.

The prototype structure is a single-column bridge pier with two lead rubber seismic isolation bearings supporting a girder. The weight of the girder and the pier are 2400 and 1300 kN, respectively. In the hybrid simulation, the pier was modeled computationally as a bilinear hysteresis relationship (elastic stiffness, 35 MN/m$^2$; yield force, 2340 kN). A lead-rubber seismic isolation bearing was tested experimentally at KU as a 1:1.78 scaled specimen (Figure 9), and the two bearings were assumed to
be identical in the model. For the hybrid simulation, the NS component of the 1940 El Centro ground motion was used as the horizontal support acceleration of the pier in the transverse direction. The client program performed the simulation by solving the two degree-of-freedom equations of motion using the \( \alpha \)–Operator Splitting time integration method.

The experimental test setup at KU is illustrated in Figure 9. Three 400 kN Schenck actuators were used; two applied the vertical load (294 kN) on the bearing under load-control and the horizontal actuator imposed the specified displacements as controlled by the server program. The AD/DA board was a National Instruments PCI-6036e for control and data acquisition.

The server program for the experimental test system ran at KU using a PC with Microsoft Windows 2000. The client program was OpenSees running on a Linux workstation at UCB. The PC and the workstation were connected to their university networks. Since Kyoto University is a member of Super SINET (Science Information Network, Japan), the 5 gigabit/sec international network was used for the test. The round trip communication time for packets between KU and UCB was about 200 msec.

4.2. Client-server application with the framework

A high-level diagram of the client-server application is shown in Figure 10. The C++ main programs for the experimental server and the computational client programs are given in Figure 11 and Figure 12, respectively. For the network communication between the two, the TCP_Socket implementation of Channel is adopted in lines 4–5 for both the server and the client programs.

Considering the experimental server program in Figure 11, the equipment specifications are defined in lines 9–12 for the actuator displacement and force, and also for the displacement transducers. These specifications are used for defining the control and data acquisition equipment (lines 14–17), and constructing the experimental control object ECNIEseries (lines 20–21). ECNIEseries is a subclass of ExperimentalControl and is implemented with the NI-DAQ library as the low-level functions for testing the specimen off-line. This controller handles one control signal for the horizontal actuator under displacement control and three DAQ signals. Therefore, EOneStaticActuator is the
ExperimentalSetup object in lines 24–25. In lines 28–36, the unit conversions between the prototype and experimental model are defined, and in lines 38–39 the remote experimental setup is constructed. After the experimental system is defined, line 43 executes the event loop for the server. Since the server does not depend on the client, a variety of test methods could be accommodated with this program for the experimental setup.

To implement the pseudodynamic test method, Figure 12 gives the client program. The client determines the loading for the bridge pier and requests the server to obtain restoring force of the ExperimentalElement. In lines 12–35 of the client program, the OpenSees Domain objects are defined (the usual constructors for the Domain objects in a model, i.e. Node, are omitted in Figure 12 for clarity). For the pier, a ZeroLength element and Steel material are used (lines 16–20). The isolation bearing is represented experimentally in this model, so in line 27 an ExperimentalBeamColumn2d element is defined for the bearing through association with the experimental site in line 24. The analysis procedure for the hybrid simulation is defined in lines 37–47 using OpenSees Analysis objects with the α-OperatorSplitting integrator (line 39). This source code is almost the same as for a computational model with the exception of the definition of the element for the seismic isolation bearing. For distributed hybrid simulation, the communication between the computational client and the experimental server is encapsulated in Channel and ExperimentalSite, so the client is not dependent on the specifics of the experimental site, the test method, or the communication protocol.

4.3. Test results

The results of the distributed pseudodynamic test for the hybrid model of the bridge pier are shown in Figure 13. The left-hand plot is the hysteresis loop of the seismic isolation bearing, expressed in the units for the prototype. The history plot on the right-hand side of the figure shows the displacement.
int main(int argc, char **argv)
{
    // Setting of the Channel---------------------------------------------
    startup_socket();
    TCP_Socket *theChannel = new TCP_Socket(atoi(argv[1]));

    // Setting of the ExperimentalControl--------------------------------
    // define EquipmentSpecs for controller
    EquipmentSpec **equip = new EquipmentSpec*[3];
    equip[0] = new EquipmentSpec(1, "ActDispl", "mm", 250.0, 2);
    equip[1] = new EquipmentSpec(2, "ActForce", "tonf", 40.0, 1);
    equip[2] = new EquipmentSpec(3, "ExtDispl", "mm", 250.0, 2);

    EquipmentSpec **ctrl_e = new EquipmentSpec*[1];
    ctrl_e[0] = equip[0];

    EquipmentSpec **daq_e = new EquipmentSpec*[3];
    daq_e[0] = equip[0];
    daq_e[1] = equip[1];
    daq_e[2] = equip[2];

    // create the Experimental Control
    ExperimentalControl *theController
        = new ECNIEseries(1, 1, 1, ctrl_e, 3, daq_e);

    // Setting of the ExperimentalSetup----------------------------------
    ExperimentalSetup *TestSetup
        = new ESOOneStaticActuator(1,*theController,1);

    // conversion between prototype units and experimental setup units
    double S = 1./1.78; // similarity ratio of length
    Vector factor(3);
    factor(1) *= S; // prototype -> model (length)
    TestSetup->setFactorDisp(factor);
    factor(1) = 1./9.8e3; // N -> tonf
    factor(1) *= S*S; // prototype -> model (force)
    factor(1) /= 2.0; // number of isolator
    TestSetup->setFactorForce(factor);

    RemoteExperimentalSetup *rTestSetup
        = new RemoteExperimentalSetup(1, *TestSetup, *theChannel);

    // Event loop--------------------------------------------------------
    // setTrialResponse, getDaqdata, getForce, getDisp...
    rTestSetup->run();

    // Cleanup socket----------------------------------------------------
    cleanup_socket();
    return 0;
}

Figure 11. C++ main program for experimental server

of the girder and the top of the pier with respect to the ground. The hysteresis shows the bilinear loop and dissipation of a large amount of the energy. As a result, the pier responded in an elastic region for the specified ground motion. Since the load on the horizontal actuator was applied at a low rate, the hybrid simulation with 500 time steps took 30 minutes to complete. When the physical control system of Figure 10 was replaced by a numerical bilinear material object, the distributed simulation took 8 minutes. This result shows that most of the time for the hybrid simulation is to conduct the test locally because the network communication and computational speed were very fast compared with the loading rate.
int main(int argc, char **argv)
{
    // Setting of the Channel--------------------------------------------
    startup_socket();
    TCP_Socket *theChannel = new TCP_Socket(atoi(argv[1]), argv[2]);

    // Setting of the domain---------------------------------------------
    Domain *theDomain = new Domain();
    int ndm = 2; int ndf = 3;
    // Define nodes------------------------------------------------------
    // Define single point constraints-----------------------------------
    // Define force-deformation relationship for pier---------------------
    UniaxialMaterial *theMat = new Steel01(3, 233.76e4, 3.5e7, 0.1);
    // Define element----------------------------------------------------
    ZeroLength *ele = new ZeroLength(1, ndm, 1, 2, x, yp, *theMat, 0);
    theDomain->addElement(ele);
    // Setting of the ExperimentalSite-----------------------------------
    ExperimentalSite *theSite = new RemoteExprSite(1, *theChannel);
    int dsize = 3; theSite->setDataSize(dsize);
    ExperimentalElement *ele2 = new ExperimentalBeamColumn2d(2, 2, 3, *theSite);
    Matrix initK(3, 3); initK(1, 1) = 4.9e7; // (N/m);
    theDomain->addElement(ele2);
    ele2->setInitialStiff(initK);
    // Setting of Uniform Excitation-------------------------------------
    // Setting of Recorder-----------------------------------------------
    // Setting of Analysis-----------------------------------------------
    AnalysisModel *theModel = new AnalysisModel();
    EquiSolnAlgo *theSolnAlgo = new Linear();
    TransientIntegrator *theIntegrator = new OperatorSplitting(0.7);
    ConstraintHandler *theHandler = new PlainHandler();
    DOF_Numberer *theNumberer = new PlainNumberer();
    BandSPDLinSolver *theSolver = new BandSPDLinLapackSolver();
    LinearSOE *theSOE = new BandSPDLinSOE(*theSolver);
    DirectIntegrationAnalysis theAnalysis(*theDomain, *theHandler, *theNumberer, *theModel, *theSolnAlgo, *theSOE, *theIntegrator);
    // perform the analysis----------------------------------------------
    int numSteps = (int)((10.0/deltaT);
    theAnalysis.analyze(numSteps, deltaT);
    // kill the process of the remote site-----------------------------
    theDomain->clearAll();
    cleanup_socket();
    return 0;
}

Figure 12. C++ main program for computational client

5. CONCLUSION

This paper has presented an object-oriented software framework for distributed experimental – computational simulation of structural systems. Computational simulation is provided in the framework to support a wide range of experimental test methods. The framework is composed of three parts, an experimental test subsystem to control local and distributed test configurations, OpenSees core for
computing the seismic response of a structural system, and a communication interface between the computational component and one or more experimental test subsystems. OpenSees is used for the time integration of the equations of motion and as the coordinator for the experimental and computational components in the hybrid model.

Based on a requirements analysis, the experimental system is designed using software classes for ExperimentalSetup, ExperimentalControl, and ExperimentalSite. ExperimentalSetup defines the interface for transforming the degrees-of-freedom for an experimental element to a vector appropriate for ExperimentalControl. ExperimentalControl represents the control system itself and is responsible for converting the DOF data for the control system. The ExperimentalSetup classes hide the specifics of the experimental configuration from ExperimentalControl, and since an ExperimentalControl object is in turn hidden from applications, they can communicate with the experimental equipment without knowledge of the specific test method or specimen. The ExperimentalSite classes are responsible for the communication between the ExperimentalSetup and hybrid simulation applications. The communication approach in OpenSees is adopted in a manner identical to the communication requirements for parallel processing.

The flexible architecture allows collaboration between the simulation software, OpenSees, and experimental sites. With the introduction of an element representing an experiment, ExperimentalElement, an experimental test subsystem can collaborate with OpenSees without any modification. The experimental system objects provide the methods to communicate with and control the experimental system. OpenSees provides the coordination of the hybrid simulation through solution of equations of motion using step-by-step integration. As a demonstration of the software framework, a distributed pseudodynamic test was presented. Connecting sites in the United States and Japan, the seismic response of the girder–pier system was simulated with client and server programs for the computational and experimental components, respectively.

In future work, other test methods involving mixed displacement and force control and communication methods, including NTCP, will be added into the framework. Extensions for real-time testing will allow a broader range of experimental methods.

ACKNOWLEDGEMENTS

The authors thank Dr. Frank McKenna of UC Berkeley for his assistance with extending OpenSees for
hybrid simulation. The development of OpenSees has been supported by the Pacific Earthquake Engineering Research Center under grant no. EEC-9701568 from the National Science Foundation. The authors also gratefully acknowledge Professors Stephen A. Mahin and Bozidar Stojadinovic at UC Berkeley for fruitful discussion on the seismic experimental test methods and comments on drafts of this paper. Finally, Professor Hirokazu Iemura provided enthusiastic support for this research and conducting the experiments at Kyoto University.

REFERENCES

The figures on object-oriented software design are described in a UML (Unified Modeling Language [17,18]) type of notation. The meaning of each symbol is shown as follows.

A class is represented by a box with the name of the class at the top. A class name in italics is an abstract class, which means that it provides a specification of operators, but the implementations are given in subclasses that are shown hierarchically with the triangular symbol and link to the subclass. Subclasses are sometimes referred to as concrete classes because they provide a specific implementation of the software behavior defined by the abstract class. Relationships with a diamond link represent aggregation of objects.