EARTHQUAKE ENGINEERING AND STRUCTURAL DYNAMICS
Earthquake Engng Struct. Dyn. (in press)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/eqe.518

Software framework for distributed experimental–computational simulation of structural systems

Yoshikazu Takahashi[1,*,†] and Gregory L. Fenves[2]

[1]*Department of Urban Management, Kyoto University, Yoshida Honmachi, Sakyo-ku, Kyoto 606-8501, Japan*
[2]*Department of Civil and Environmental Engineering, University of California, Berkeley, Berkeley, CA 94720-1710, U.S.A.*

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 co-ordinates the control of local and distributed configurations of experimental equipment. The object-oriented design of the software promotes the sharing of modules for experimental equipment, test set-ups, 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. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: object-oriented software; pseudodynamic test method; software framework; distributed experimental test methods; hybrid experiments

*Correspondence to: Yoshikazu Takahashi, Department of Urban Management, Kyoto University, Yoshida Honmachi, Sakyo-ku, Kyoto 606-8501, Japan.
†E-mail: yos@catfish.kuciv.kyoto-u.ac.jp

Contract/grant sponsor: The Ministry of Education, Science, Sports and Culture, Japan; contract/grant number: Grant-in-Aid for Young Scientists (A) 16686029

Received 17 March 2005
Revised 3 June 2005
Accepted 14 June 2005

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 $20\,\mathrm{m} \times 15\,\mathrm{m}$ six degree-of-freedom (DOF) shaking table at the National Research Institute for Earth Science and Disaster Prevention (NIED) in Japan [1] 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. Since the 1970s, experimental methods have been developed to test structural systems using strong-reaction facilities and standard laboratory equipment such as servo-controlled hydraulic actuators, control systems, sensors, and data acquisition systems. The pseudodynamic test method [2–4] is an experimental method that is used to deform a structural specimen as if it were responding to earthquake ground motion using an on-line computer-controlled simulation of dynamic response. Recent research has generalized the pseudodynamic test method to include experimental substructures connected to a computational model through multiple DOF [5–9]. The pseudodynamic test method can be viewed as a hybrid simulation in which one portion of a structural system is simulated experimentally and other portions are simulated using a computational model. The software that implements the test method assembles the experimental and computational assemblies to determine the dynamic response of the complete system. Since hybrid simulation involves communication between experimental and computational assemblies, laboratories and computers can be geographically distributed and linked by a high-speed communication network. 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. The concept was discussed by Campbell and Stojadinovic [10] and also by Watanabe *et al.* [11]. Distributed hybrid tests have been conducted between Japan and Korea [12] and in Taiwan [13].

In the United States, the George E. Brown, Jr Network for Earthquake Engineering Simulation (NEES) [14] is a network of experimental sites and computing resources connected by the NEESgrid system using Grid-based middleware [15]. Under the NEES program, the Multi-site On-line Simulation Test (MOST) [16] demonstrated the use of NEESgrid for hybrid simulation, and Mosqueda *et al.* [17] developed a continuous control method for hybrid simulation using distributed experimental sites. MOST [15, 16] connected two experimental sites and one or more computational sites. It showed the potential for distributed testing, but the software architecture included a co-ordinator that resulted in an artificial separation between the simulation model and the time integration procedure for solving the governing equations of motion. Shing *et al.* [18] also developed a fast hybrid test (FHT) system with a Shared Common RAM Network (SCRAMNet). They used OpenSees [19–21] for computational simulation with a specialized integration scheme.

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. The deployment of NEES in the United States, and similar systems in other countries, provides a timely opportunity to develop flexible and extensible software for a

large variety of structural test methods, specimen configurations, control systems, communication protocols, computational models, solution algorithms, and computing resources that can be combined for a distributed hybrid test. To accomplish this goal, this paper presents a software framework for distributed experimental–computational simulation. The architecture provides a general approach for defining and conducting structural tests on local or distributed experimental sites and computational resources. In this paper, the requirements for experimental methods in structural engineering are examined in order to define the functions that the software must support. Using object-oriented software design methodologies, a framework of co-operating 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. The hybrid simulation software extends OpenSees and it takes advantage of OpenSees support for distributed computation. The new software framework is evaluated using a client–server application for a distributed pseudodynamic test of a bridge system with the experimental site in Japan acting as the server and the computational client located in the United States.

2. OPEN SYSTEM FOR EARTHQUAKE ENGINEERING SIMULATION

The Open System for Earthquake Engineering Simulation, OpenSees [19–21], has been developed by the Pacific Earthquake Engineering Research Center (PEER) for modelling and simulating the seismic response of structural and geotechnical systems in support of performance-based earthquake engineering methodologies. OpenSees is an object-oriented software framework implemented in C++ [22] through an open-source development process. Most users specify and conduct an OpenSees simulation using the Tcl scripting language [23]. As will be demonstrated in this paper, OpenSees can be used to build other applications, such as for distributed hybrid simulation.

The fundamental characteristic of object-oriented software is abstraction: identifying the important software behaviour needed to solve a problem and breaking it down into components, which are referred to as software classes. Software objects are instances of a class, which contain the specification for constructing and operating on objects constructed from the class. An object encapsulates data and operators on the data, thus hiding the implementation of the operators from the specification. Data hiding encourages modular, flexible, and extensible software. An operator is invoked by sending an object a message; the object is then responsible for invoking the implementation of the operator based on its class. A framework is a set of inter-related classes that can be used to develop an application to solve a problem. For structural simulation, object-oriented methodologies were introduced in the beginning in 1990s. Fenves [24], Baugh and Rehak [25] 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*. [26] 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 [20, 27–35].

Figure 1 shows the high-level classes in OpenSees and the relationships between them. The figure uses simplified graphical notation of the Unified Modeling Language [36, 37], which is defined in the appendix. All classes have definitions of operators, but for clarity

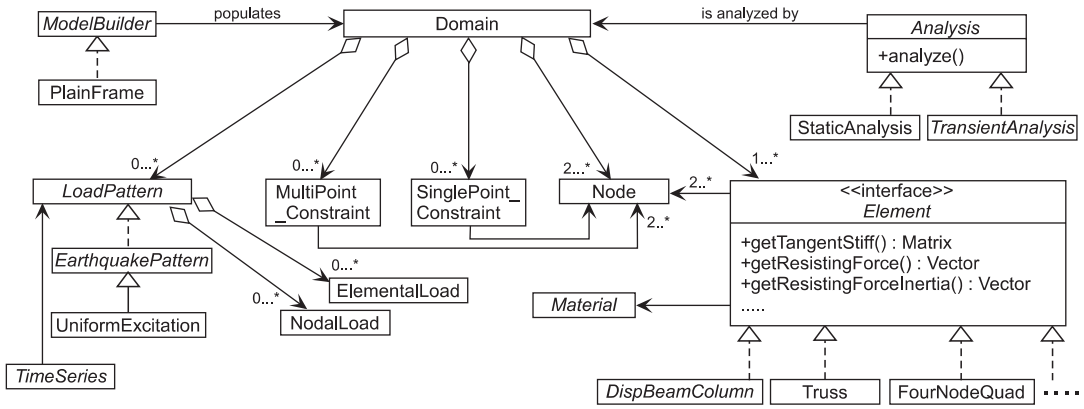

Figure 1. High-level classes in the OpenSees software framework for computational simulation.

only selected ones are shown in the class diagrams presented herein. Relationships with a diamond link represent aggregation of objects, which is a common software design pattern [38] known as Composition. The high-level classes in Figure 1 are *Domain*, *ModelBuilder*, and *Analysis* to represent, respectively, the data structures for a model, the methods for creating models, and the operators for advancing the state of a model through an analysis. An instance of *Domain* is an aggregation of other objects making up a simulation model, including classes representing nodes, elements, constraints, and load patterns. As shown in Figure 1, *Element* is an abstract class, and subclasses implement element formulations for beam-column elements and continuum elements (the figure only shows a small sample of the elements in OpenSees). The most important function for *Element* is to compute the resisting forces given specified displacement and velocities at the DOF. Each subclass of *Element* defines the operator *getResistingForce( )*, which returns a reference (memory address) for the vector of resisting forces. Since computational elements involve material (or constitutive) relationships, the state information is often encapsulated in *Material* objects, which have similar state determination operators.

In Figure 1, *Analysis* is an abstract class that is responsible for advancing the *Domain* object from its current state to a new state based on one step of the *LoadPattern*. Subclasses of *Analysis* provide the implementations for static analysis, transient analysis, or other analysis procedures. Although not shown in the figure, *Analysis* is a composition of the other objects required for solving the governing equations, such as *Integrator* and *SolutionAlgorithm*. These components are abstract classes, which make it possible to choose the type of analysis by selecting appropriate subclasses. Once the *Analysis* object is created as a composition, the operator *analyze( )* advances the state of the model.

An important aspect of the OpenSees architecture is the support for parallel computing [39]. The main classes for parallel computing are *Actor*, *Shadow*, *Channel* and *MovableObject*, as shown in Figure 2. 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

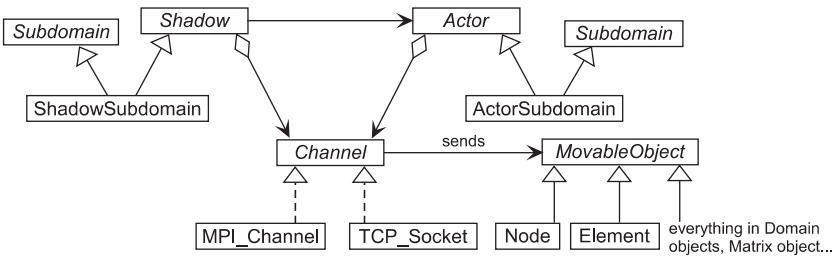

Figure 2. OpenSees classes for parallel computation and communication.

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.

In summary, the OpenSees object-oriented framework provides classes that can be combined for specific models and solution procedures in a simulation. The coupling between classes is minimized, which provides a great deal of flexibility. New classes can be developed more easily because the interaction between classes is defined through interfaces (often called application program interfaces) and there are few global data structures. The software can be extended for other applications, such as described in the remainder of this paper for hybrid experimental–computational simulation.

3. 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 set-ups for applying boundary conditions, as shown for example in Figure 3, which illustrates three configurations of actuators for two-DOF loading on a cantilever specimen. 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.

Cyclic loading of a specimen at a slow rate with an *a priori* deformation history is a basic experimental method [40]. In the quasi-static test method, the loading is selected to represent cycles of deformation expected during an earthquake. The experimental set-up 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 set-up 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

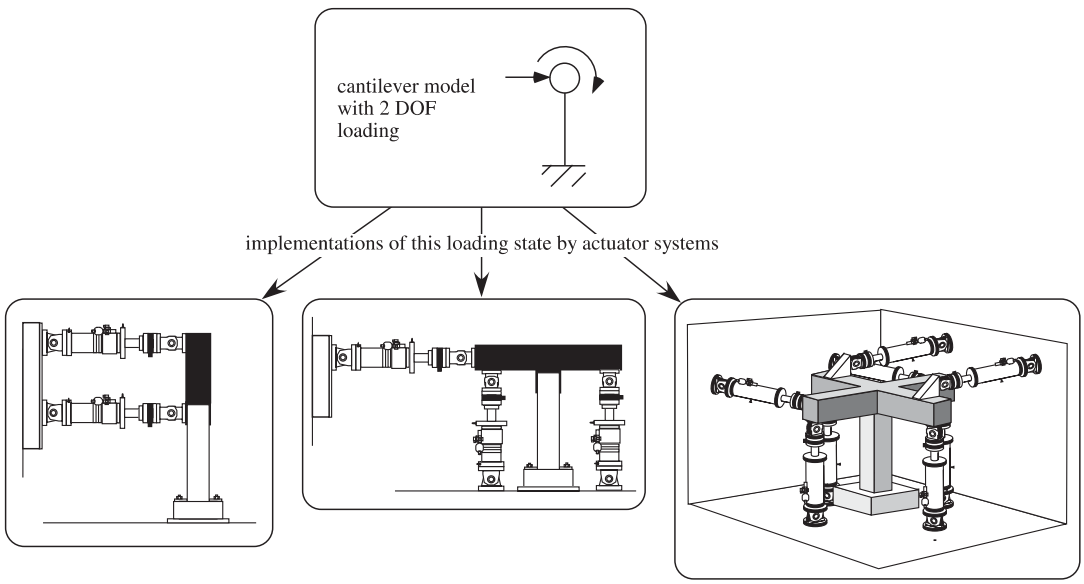

Figure 3. Possible configurations for two-DOF loading on a cantilever specimen.

of the components in the experimental set-up and to specify the boundary conditions through displacements or forces in one or more actuators.

As discussed in Section 1, the pseudodynamic test method is a computer-controlled experimental technique for simulating the response of structures [2]. It is an effective alternative to shaking table testing because it can load specimens that exceed the size, load, or weight limits of shaking tables using experimental set-ups similar to that for quasi-static tests. 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 non-linear equations of motion but an iterative procedure is not desirable for testing, therefore several time integration schemes have been proposed [41–45], and a variety of techniques have been developed to improve efficiency and accuracy [4, 7, 8]. By utilizing partitioning concepts, the pseudodynamic test method allows testing a component or assembly experimentally, whereas the other portions of the structural system are modelled 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. A hold time allows relaxation of a specimen's resisting forces and other experimental errors that are fed back into the solution. To mitigate these errors, continuous pseudodynamic test methods have been developed [9, 17]. To generate continuous displacement signals, researchers adopted an interpolation–extrapolation procedure [46]. Since this procedure runs fast and independently from the equation solution, it is usually implemented locally in a digital signal processor (DSP).

The software for the pseudodynamic test method needs to provide a variety of computational models for structural systems, including non-linear material and geometric behaviour.

In addition, the test method requires specialized time integration schemes and definitions of boundary conditions for the experimental specimen. To support interpolation–extrapolation procedures, 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.

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 [46–49]. 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.

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) [50] 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 [51] and it is also fault-tolerant in that it does not rely on the underlying transport layer (e.g. TCP) to deliver messages reliably. The MOST project [15, 16] used NTCP for communication between experimental and computational sites.

4. 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 set-up, and (3) obtain the resisting forces and displacements from the experimental set-up. 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 set-ups 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.

4.1. Modelling of experimental systems

As an example of software modelling of experimental systems, Figure 4 shows the important objects for representing experimental set-ups of the two-DOF displacement-controlled loading on the specimen illustrated in Figure 3. 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 $\mathbf{d}_e$. The software can represent a variety of set-ups for the experimental element, such as the three alternatives, set1, set2, and set3 shown in Figure 4. In a set-up, 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 set-ups.

Two abstract classes are defined to provide a flexible representation of test methods. The first one, *ExperimentalSetup*, transforms the DOF, $\mathbf{d}_e$, for an experimental element to a vector, $\mathbf{s}_c$, for the control system depending on the geometry and kinematics of the loading system.

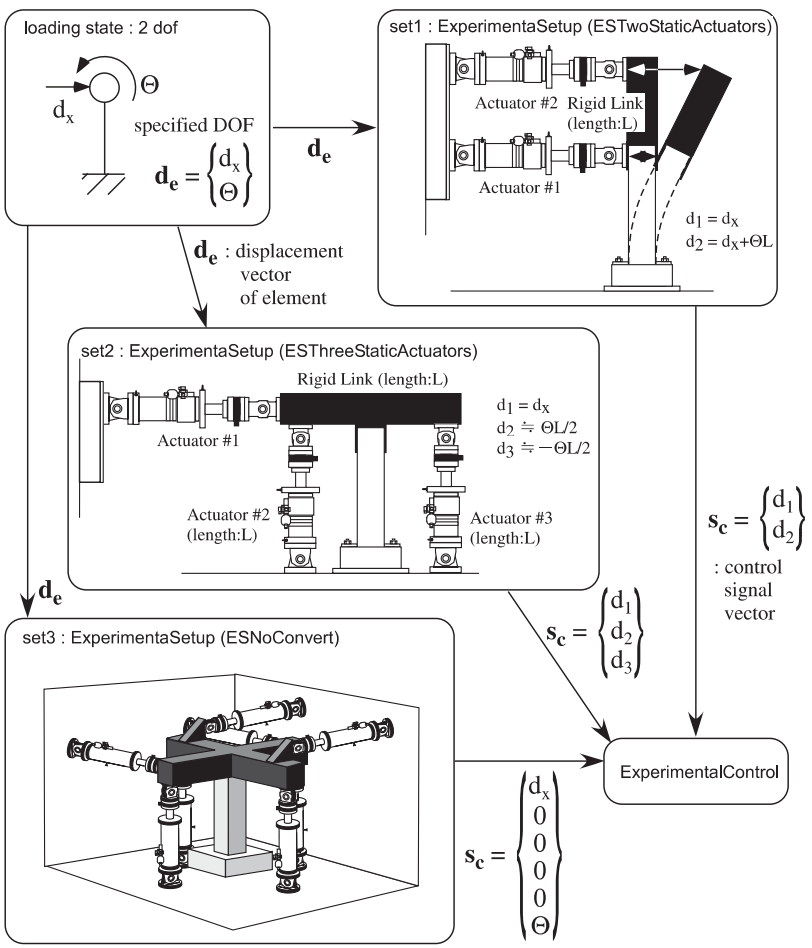

Figure 4. Examples of ExperimentalSetup object for loading a cantilever specimen with two-DOF.

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 5. *ExperimentalSetup* is an abstract class that represents the set-up 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 [38]. 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 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 5 provide information for the control of an experimental set-up. The specifications of the experimental equipment, such as range, gain and units, are modelled 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* classes for their control systems and experimental equipment. Laboratories with similar equipment could share the software implementations. For each experimental set-up, a subclass of *ExperimentalSetup* needs to be implemented for the configuration of the

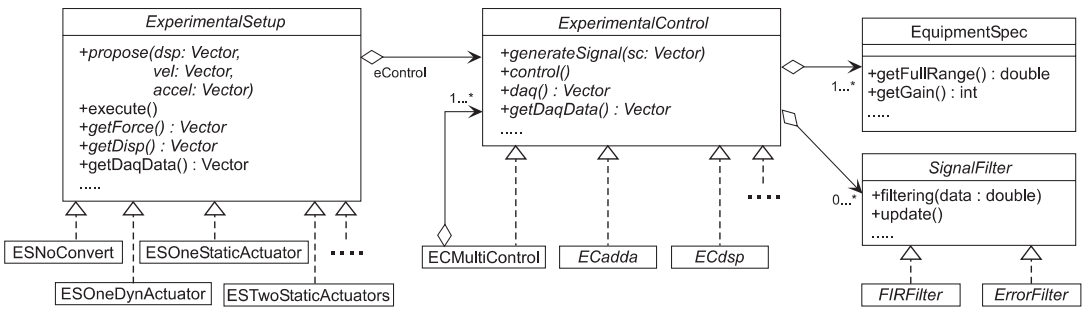

Figure 5. Principal classes for representing experimental methods.

actuators and other details of the test set-up. Libraries of experimental set-up 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 *Experimental-Control* objects.

To demonstrate the flexibility of the software architecture, Figure 6 shows two configurations for an experimental set-up object, in this case a one-DOF loading on a cantilever specimen. The transformation in the *ExperimentalSetup* is simple because of the single-DOF loading, and it is implemented as a subclass, *ESOneStaticActuator*. In Figure 6(a), the actuator is driven by an analogue controller with a signal generated by an AD/DA board. The *ECadda* object, an instance of a subclass of *ExperimentalControl*, generates the target displacement signal based on the command vector $\mathbf{s}_c$ and acquires analogue experimental data. These data are packed in the vector $\mathbf{s}_d$ and sent to the *ESOneStaticActuator* object. Finally, the *ESOneStaticActuator* object converts the acquired data to the resisting force vector. As an alternative, Figure 6(b) shows a set-up with a controller that uses analogue signals generated by a program executing in a DSP. The *ECdsp* object communicates with the DSP, sends the command vector $\mathbf{s}_c$, and gets the DAQ vector $\mathbf{s}_d$. As a final example, Figure 7 shows a test set-up for a two-DOF loading on a cantilever specimen. For the configuration of actuators, a subclass *ESTwoStaticActuators* is introduced. The *ECadda* object can be reused from the

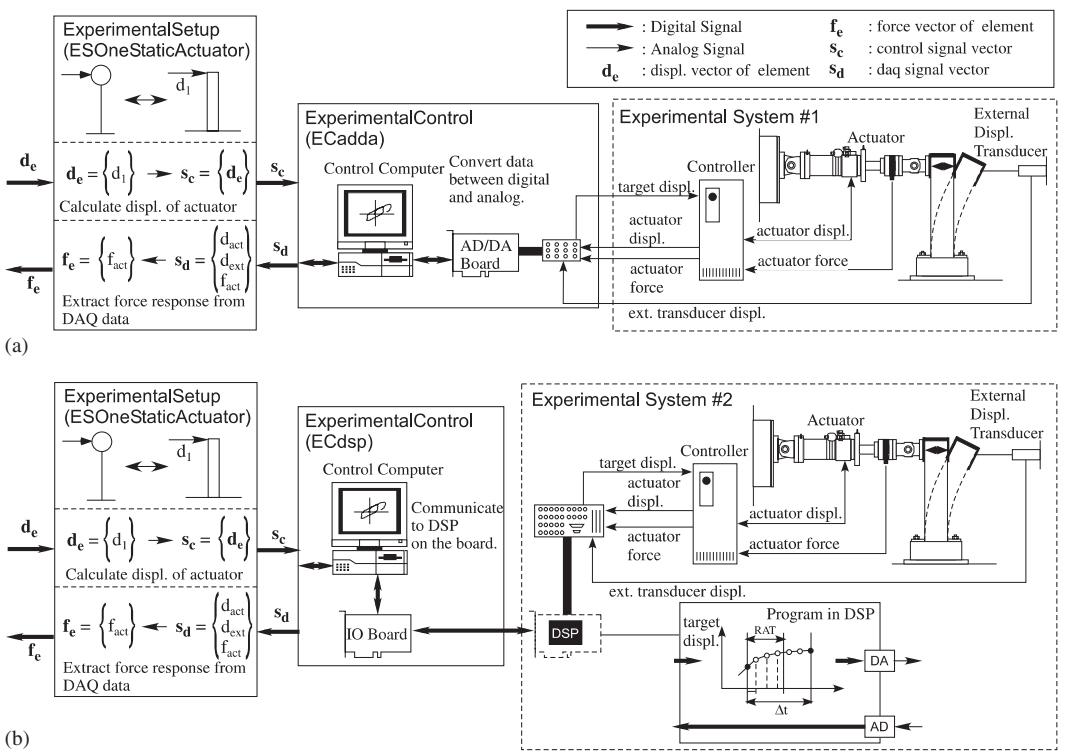

Figure 6. Quasi-static load test set-ups for single-DOF loading with two different experimental control systems: (a) analog control; and (b) digital control.

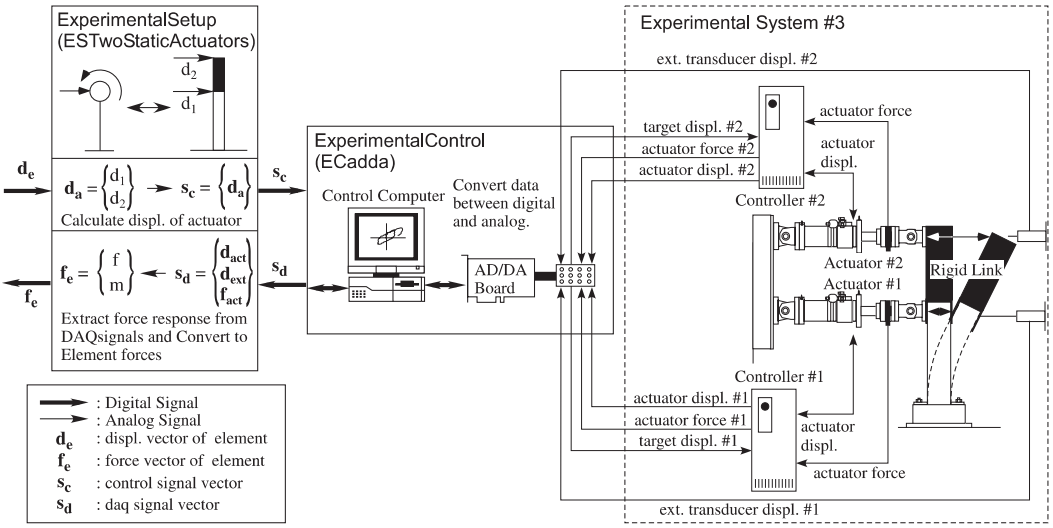

Figure 7. Quasi-static load test set-up for two-DOF loading on a cantilever specimen.

previous example because the controllers use analogue signals generated by the AD/DA board. These examples illustrate that the design of the software has factored the essential aspects into independent classes. Different *ExperimentalControl* objects can interact with the same *ExperimentalSetup* object. For the control system at an equipment site, an *ExperimentalControl* subclass can be implemented to interoperate with any experimental set-up.

4.2. Modelling of distributed experimental sites

The software framework described in the previous subsection defined objects that represent the experimental set-up, irrespective of the location of the experiment. To accommodate geographically distributed experimental set-ups, the class *ExperimentalSite* is introduced to provide a representation of local or remote experimental sites in a uniform manner. Figure 8 shows the class design to provide communication with an *ExperimentalSite*, either locally or remotely. A key operator for *ExperimentalSite* objects is *getforce( )* and Figure 8 shows the pseudocode 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 set-up. This situation is analogues 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 experimental set-up. At another location, a client program has a representation of the experimental site using an instance of *RemoteExprSite*, which is a subclass of *Shadow*. In a client–server

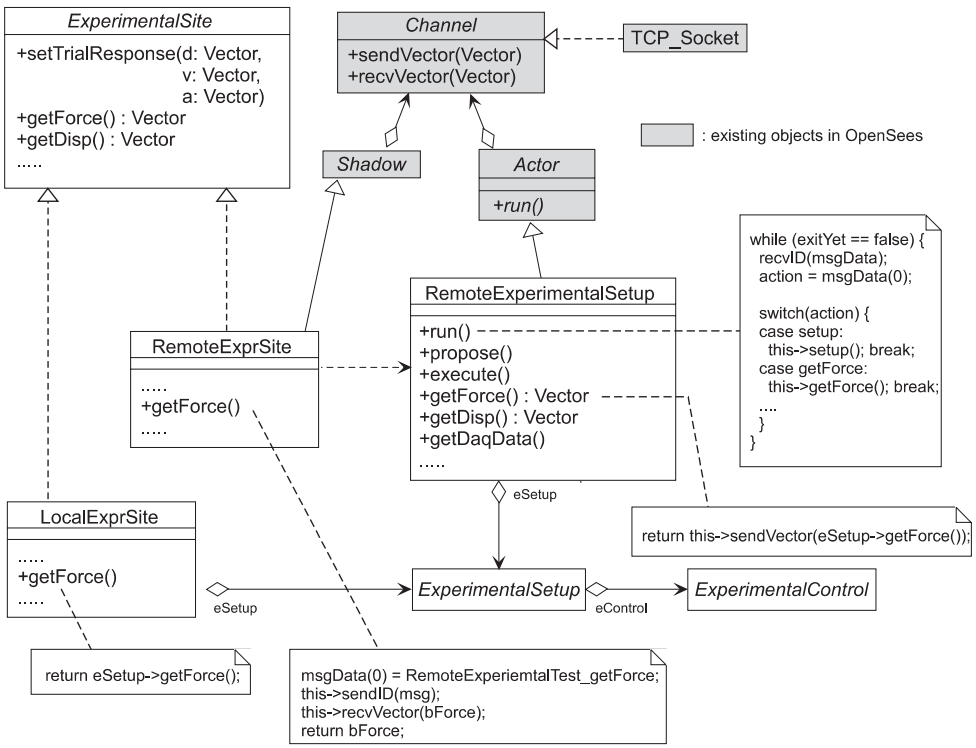

Figure 8. *ExperimentalSite* classes for communicating between local and remote *ExperimentalSetup* objects.

approach, the server program waits for requests from clients using a loop operation. The server loop is implemented in *RemoteExperimentalSetup* but almost all other methods are the same as *ExperimentalSetup*. Therefore, *RemoteExperimentalSetup* is designed with an Adapter pattern [38] of *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. A variety of network communication protocols have been developed with TCP sockets [52] one of the most widely used for the internet. TCP sockets provide end-to-end connections between two applications using TCP/IP communication. The OpenSees *Channel* may be implemented using other protocols such as MPI or NTCP [50].

The software framework for experimental test sites is defined by the class diagrams in Figure 5 and Figure 8. 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 Figures 9 and 10, respectively. These diagrams show three classes and three major methods: *setTrialResponse( )*, *getForce( )*, and

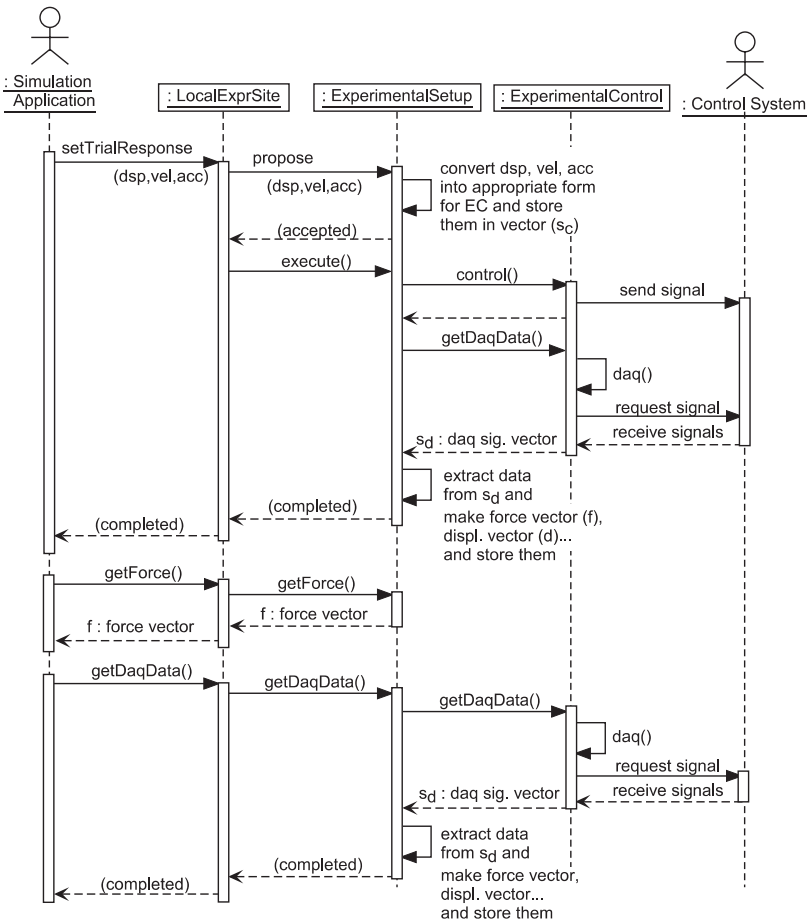

Figure 9. Sequence diagram for local experimental test.

getDaqData( ). The method *setTrialResponse( )* 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 5. 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 set-up and can combine computational and experimental elements as described in the next subsection on collaboration with OpenSees.

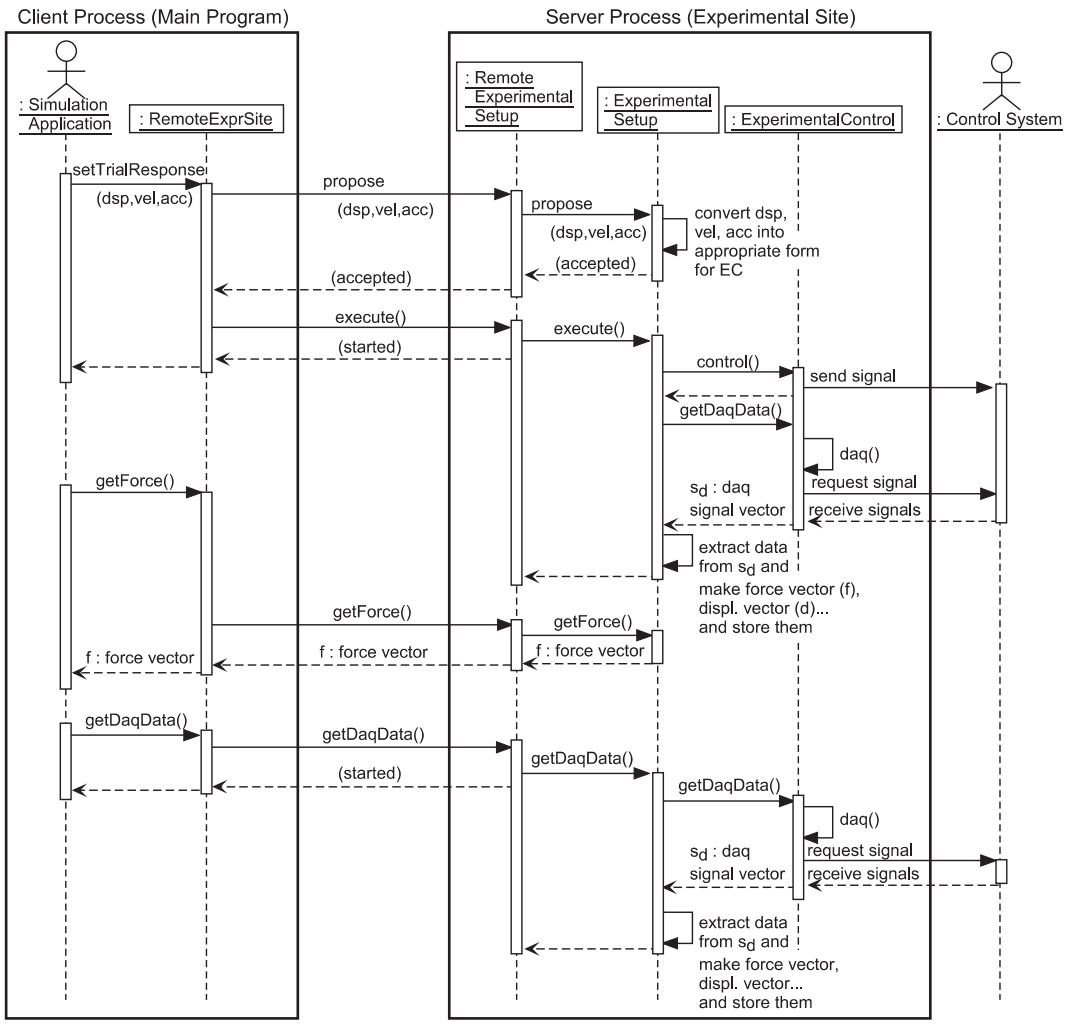

Figure 10. Sequence diagram for distributed experimental test.

4.3. Collaboration between experimental systems and OpenSees

As described in Section 3, 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 11 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 (Figure 1). In a similar manner, an experimental object has methods to set values of the DOF and to obtain the resisting force vector.

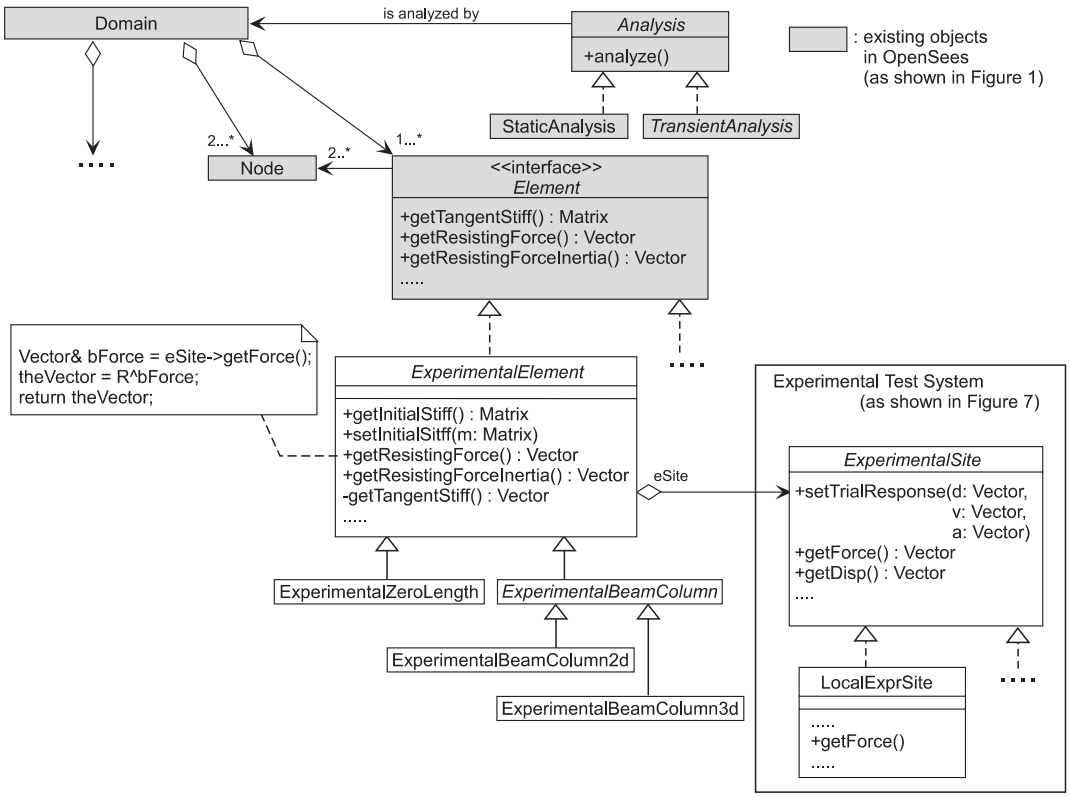

Figure 11. Class diagram of collaboration relationships between OpenSees and experimental test system.

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.

For time integration of the equations of motion, OpenSees has several implicit step-by-step methods, e.g. Newmark β, Wilson θ and HHT-α. 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 α-operator splitting integration method [43, 44], which has been implemented as an *Integrator* object in OpenSees.

The collaboration between OpenSees and an experimental system is illustrated in Figure 12, 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 co-ordinates the process, and also the computational model. As a result, a computational simulation and

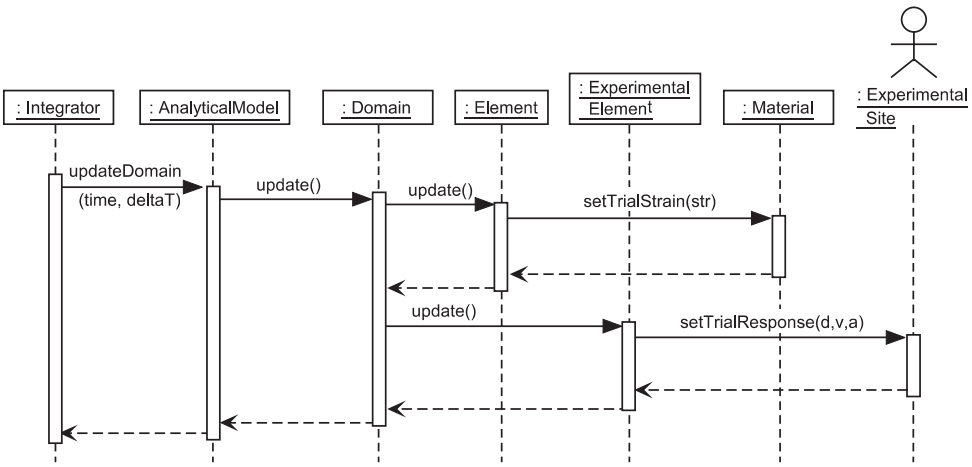

Figure 12. Sequence diagram for updating the state of an experimental element in one step of time integration.

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.

5. EXAMPLE OF DISTRIBUTED PSEUDODYNAMIC TEST

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), U.S.A. Figure 13 schematically illustrates the distributed test.

5.1. Modelling

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 (Figure 13). In the hybrid simulation, the pier was modelled 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 14), 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 DOF equations of motion using the α-operator splitting time integration method.

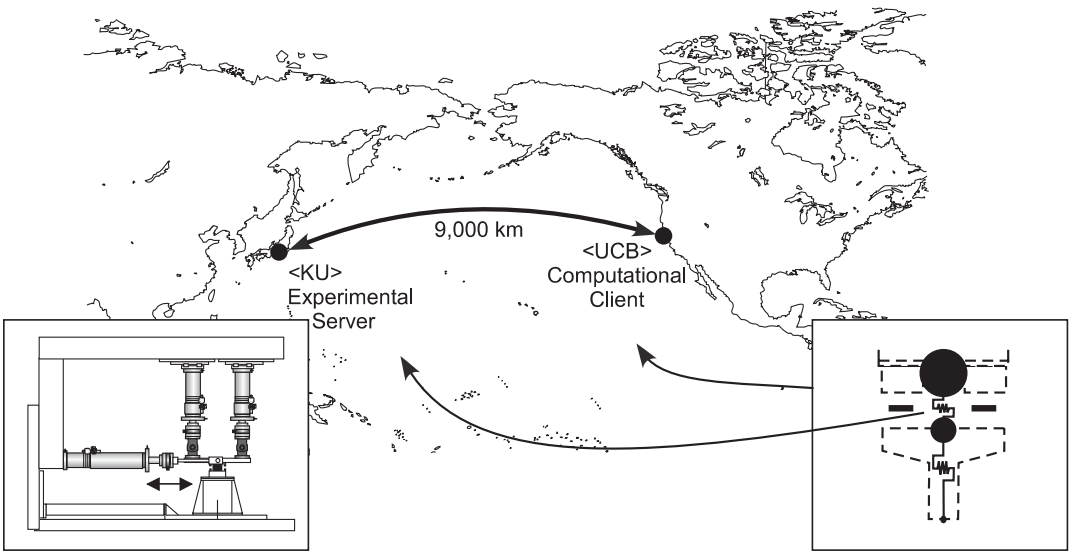

Figure 13. Schematic of distributed experimental–computational simulation of a bridge pier with the pseudodynamic test method.

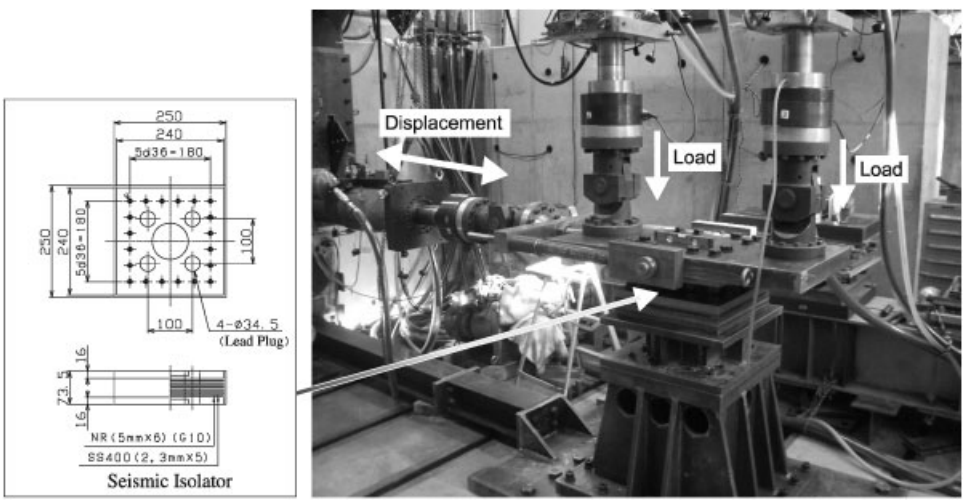

Figure 14. Experimental test set-up for seismic isolation bearing at Kyoto University.

5.2. *Test system*

The experimental test set-up at KU is illustrated in Figure 14. 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 test was monitored by web browsers with a network camera.

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 GB/s international network was used for the test. The round trip communication time for packets between KU and UCB was about 200 ms.

5.3. *Client–server application with the framework*

A high-level diagram of the client–server application is shown in Figure 15. The C++ main programs for the experimental server and the computational client programs are given in Figures 16 and 17, 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 16, 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

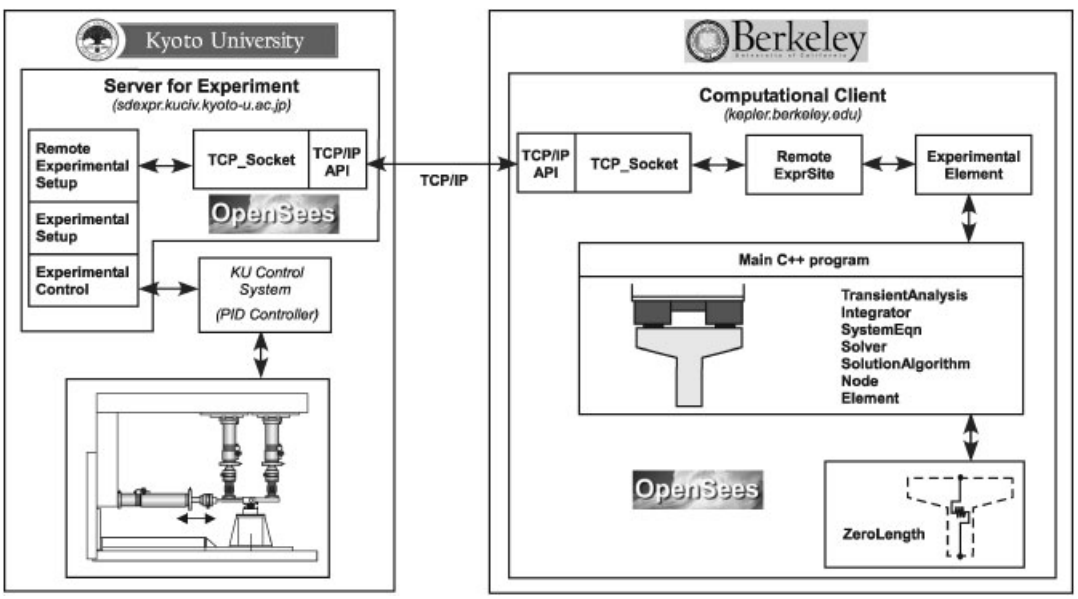

Figure 15. Client–server application for distributed pseudodynamic test of bridge pier.


```
int main(int argc, char **argv)                                                  1
{                                                                                2
  // Setting of the Channel-----------------------------------------------      3
  startup_socket();                                                              4
  TCP_Socket *theChannel = new TCP_Socket(atoi(argv[1]));                        5
                                                                                 6
  // Setting of the ExperimentalControl-----------------------------------      7
  // define EquipmentSpecs for controller                                        8
  EquipmentSpec **equip = new EquipmentSpec* [3];                                9
  equip[0] = new EquipmentSpec(1, "ActDispl", "mm", 250.0, 2);                   10
  equip[1] = new EquipmentSpec(2, "ActForce", "tonf", 40.0, 1 );                 11
  equip[2] = new EquipmentSpec(3, "ExtDispl", "mm", 250.0, 2 );                  12
                                                                                 13
  EquipmentSpec **ctrl_e = new EquipmentSpec* [1];                               14
  ctrl_e[0] = equip[0];                                                          15
  EquipmentSpec **daq_e = new EquipmentSpec* [3];                                16
  daq_e[0] = equip[0]; daq_e[1] = equip[1]; daq_e[2] = equip[2];                 17
                                                                                 18
  // create the Experimental Control                                             19
  ExperimentalControl *theController                                             20
                    = new ECNIEseries(1, 1, 1, ctrl_e, 3, daq_e);                21
                                                                                 22
  // Setting of the ExperimentalSetup-------------------------------------      23
  ExperimentalSetup *TestSetup                                                   24
                  = new ESOneStaticActuator(1,*theController,1);                 25
                                                                                 26
  // conversion between prototype units and experimental setup units             27
  double S = 1./1.78; // similarity ratio of length                              28
  Vector factor(3);                                                              29
  factor(1) = 1000.0; // m -> mm                                                 30
  factor(1) *= S;  // prototype -> model (length)                                31
  TestSetup->setFactorDisp(factor);                                              32
  factor(1) = 1./9.8e3; // N -> tonf                                             33
  factor(1) *= S*S; // prototype -> model (force)                                34
  factor(1) /= 2.0;  // number of isolator                                       35
  TestSetup->setFactorForce(factor);                                             36
                                                                                 37
  RemoteExperimentalSetup *rTestSetup                                            38
         = new RemoteExperimentalSetup(1, *TestSetup, *theChannel);              39
                                                                                 40
  // Event loop-----------------------------------------------------------      41
  //  setTrialResponse, getDaqdata, getForce, getDisp....                        42
  rTestSetup->run();                                                             43
                                                                                 44
  // Cleanup socket-------------------------------------------------------      45
  cleanup_socket();                                                              46
                                                                                 47
  return 0;                                                                      48
}                                                                                49
                                                                                 50
```

Figure 16. C++ main program for experimental server.

the NI-DAQ library [53] 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, *ESOneStaticActuator* 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 set-up 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 set-up.

To implement the pseudodynamic test method, Figure 17 gives the client program. The client determines the loading for the bridge pier and requests the server to obtain restor-


```
int main(int argc, char **argv)                                                    1
{                                                                                  2
  // Setting of the Channel-------------------------------------------             3
  startup_socket();                                                                4
  TCP_Socket *theChannel = new TCP_Socket(atoi(argv[1]), argv[2]);                 5
                                                                                   6
  // Setting of the domain--------------------------------------------             7
  Domain *theDomain = new Domain();                                                8
  int ndm = 2; int ndf = 3;                                                        9
                                                                                   10
  // Define nodes-----------------------------------------------------             11
  . . . .                                                                          12
  // Define single point constraints----------------------------------             13
  . . . .                                                                          14
  // Define force-deformation relationship for pier-------------------             15
  UniaxialMaterial *theMat = new Steel01(3, 233.76e4, 3.5e7, 0.1);                 16
                                                                                   17
  // Define element---------------------------------------------------             18
  . . . .                                                                          19
  ZeroLength *ele = new ZeroLength(1, ndm, 1, 2, x, yp, *theMat, 0);               20
  theDomain->addElement(ele);                                                      21
                                                                                   22
  // Setting of the ExperimentalSite----------------------------------             23
  ExperimentalSite *theSite = new RemoteExprSite(1, *theChannel);                  24
  int dsize = 3; theSite->setDataSize(dsize);                                      25
                                                                                   26
  ExperimentalElement *ele2 = new ExperimentalBeamColumn2d(2, 2, 3, *theSite);     27
  Matrix initK(3,3); initK(1,1) = 4.9e7; // (N/m);                                 28
  theDomain->addElement(ele2);                                                     29
  ele2->setInitialStiff(initK);                                                    30
                                                                                   31
  // Setting of Uniform Excitation------------------------------------             32
  . . . .                                                                          33
  // Setting of Recorder----------------------------------------------             34
  . . . .                                                                          35
  // Setting of Analysis----------------------------------------------             36
  AnalysisModel  *theModel = new AnalysisModel();                                  37
  EquiSolnAlgo   *theSolnAlgo = new Linear();                                      38
  TransientIntegrator *theIntegrator = new OperatorSplitting(0.7);                 39
  ConstraintHandler *theHandler = new PlainHandler();                              40
  DOF_Numberer   *theNumberer = new PlainNumberer();                               41
  BandSPDLinSolver *theSolver = new BandSPDLinLapackSolver();                      42
  LinearSOE      *theSOE = new BandSPDLinSOE(*theSolver);                          43
                                                                                   44
  DirectIntegrationAnalysis theAnalysis(*theDomain, *theHandler,                   45
                         *theNumberer, *theModel, *theSolnAlgo,                    46
                         *theSOE, *theIntegrator);                                 47
                                                                                   48
  // perform the analysis---------------------------------------------             49
  int numSteps = (int)(10.0/deltaT);                                               50
  theAnalysis.analyze(numSteps, deltaT);                                           51
                                                                                   52
  // kill the process of the remote site------------------------------             53
  the Domain->clearAll();                                                          54
  cleanup_socket();                                                                55
                                                                                   56
  return 0;                                                                        57
}                                                                                  58
                                                                                   59
```

Figure 17. C++ main program for computational client.

ing 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 17 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

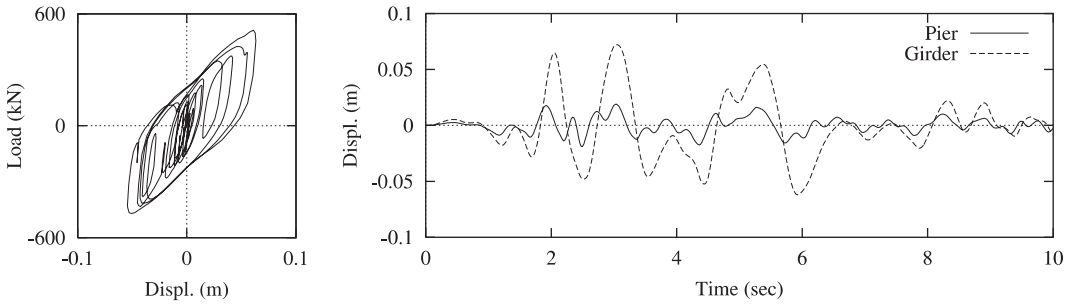

Figure 18. Results of distributed pseudodynamic test.

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.

5.4. Test results

The results of the distributed pseudodynamic test for the hybrid model of the bridge pier are shown in Figure 18. 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 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 min to complete. When the physical control system of Figure 15 was replaced by a numerical bilinear material object, the distributed simulation took 8 min. 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.

6. 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 co-ordinator 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 co-ordination 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 [54] and communication methods, including NTCP, will be added into the framework. Extensions for real-time testing will allow a broader range of experimental methods.

APPENDIX A: UML NOTATION

The figures on object-oriented software design are described in a UML (Unified Modeling Language [36, 37]) 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 behaviour defined by the abstract class. Relationships with a diamond link represent aggregation of objects.

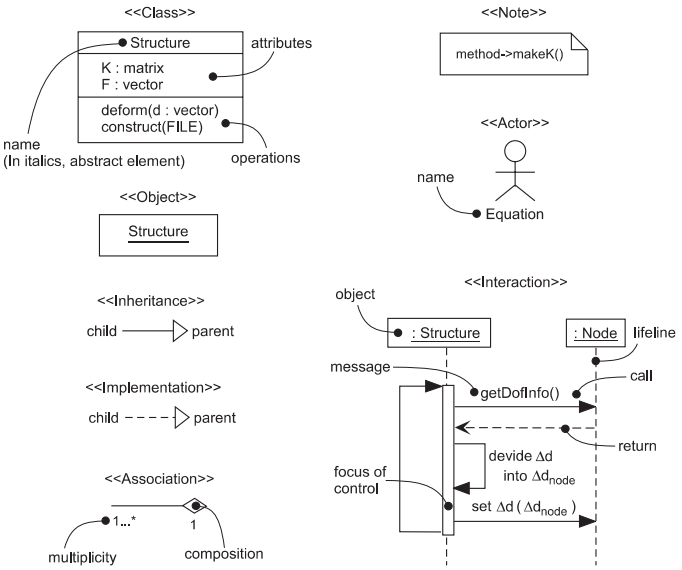

Figure A1. UML notation.

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

1. Ohtani K, Ogawa N, Katayama T, Shibata H. 3-D full-scale earthquake testing facility and earthquake engineering network. *Proceedings of the 3rd World Conference on Structural Control*, Como, Italy, 2002.
2. Hakuno M, Shidawara M, Hara T. Dynamic destructive test of a cantilever beam controlled by an analog-computer. *Journal of JSCE* 1969; **171**:1–9 (in Japanese).
3. Takanashi K, Udagawa K, Seki M, Okada T, Tanaka H. Non-linear earthquake response analysis of structures by a computer-actuator on-lin system. *Transaction of the Architectural Institute of Japan* 1975; **229**:77–83.
4. Iemura H. Development and future prospect of hybrid experiment. *Journal of JSCE* 1985; **356/I-3**:1–10 (in Japanese).
5. Dermitzakis S, Mahin S. Development of substructuring techniques for on-line computer controlled seismic performance testing. *Technical Report UCB/EERC-85/04*, Earthquake Engineering Research Center, University of California, Berkeley, 1985.
6. Takanashi K, Nakashima M. Japanese activies on on-line testing. *Journal of Engineering Mechanics* 1987; **113**(7):1014–1032.
7. Mahin SA, Shing PSB, Thewalt CR, Hanson RD. Pseudodynamic test method: current status and future directions. *Journal of Structural Engineering* (ASCE) 1989; **115**(8):2113–2128.
8. Shing PSB, Nakashima M, Bursi OS. Application of pseudodynamic test method to structural research. *Earthquake Spectra* 1996; **12**(1):29–56.
9. Magonette G. Development and application of large-scale continuous pseudo-dynamic testing techniques. *Philosophical Transactions of The Royal Society of London* 2001; **A**(359):1771–1799.

10. Campbell S, Stojadinovic B. A system for simultaneous pseudodynamic testing of multiple substructures. *Proceedings of the Sixth U.S. National Conference on Earthquake Engineering*, Seattle, U.S.A., 1998.
11. Watanabe E, Sugiura K, Nagata K, Suzuka Y. Development of parallel pseudo-dynamic test system. *Proceedings of 10th Japan Earthquake Engineering Symposium*, vol. 2. 1998; 2205–2210 (in Japanese).
12. Watanabe E, Yun CB, Sugiura K, Park DU, Nagata K. On-line interactive testing between KAIST and Kyoto University. *Proceedings of the 14th KKNN Symposium on Civil Engineering*, Kyoto, Japan, 2001; 369–374.
13. Tsai KC, Yeh CC, Yang YC, Wang KJ, Chen PC. Seismic hazard mitigation: internet-based hybrid testing framework and examples. *International Colloquium on Natural Hazard Mitigation: Methods and Applications*, France, 2003.
14. NEES Consortium, Inc page. http://www.nees.org (13 November 2003).
15. Specner B, Finholt T, Foster I, Kesselman C. Neesgrid: a distributed collaboratory for advanced earthquake engineering experimentation and simulation. *Proceedings of 13th World Conference on Earthquake Engineering*, Vancouver, Canada, 2004; 1674. DVD-ROM.
16. Multi-site, on-line simulation test, NEESgrid. http://www.neesgrid.org/most (13 November 2003).
17. Mosqueda G, Stojadinovic B, Mahin S. Geographically distributed continuous hybrid simulation. *Proceedings of 13th World Conference on Earthquake Engineering*, Vancouver, Canada, 2004; 0959. DVD-ROM.
18. Shing P, Wei Z, Jung RY, Stauffer E. NEES fast hybrid test system at the University of Colorado. *Proceedings of 13th World Conference on Earthquake Engineering*, Vancouver, Canada, 2004; 3497. DVD-ROM.
19. OpenSees page. http://opensees.berkeley.edu (13 November 2003).
20. McKenna F. Object oriented finite element analysis: frameworks for analysis algorithms and parallel computing. *Ph.D. Thesis*, University of California, Berkeley, 1996.
21. Fenves GL, McKenna F, Scott MH, Takahashi Y. An object-oriented software environment for collaborative network simulation. *Proceedings of 13th World Conference on Earthquake Engineering*, Vancouver, Canada, 2004; 1492. DVD-ROM.
22. Stroustrup B. *The C++ Programming Language* (3rd edn). Addison-Wesley: Reading, MA, 1997.
23. Ousterhout J. *Tcl and the Tk Toolkit*. Addison-Wesley: Reading, MA, 1994.
24. Fenves GL. Object-oriented programming for engineering software development. *Engineering with Computers* 1990; **6**:1–15.
25. Baugh Jr JW, Rehak DR. Data abstraction in engineering software development. *Journal of Computing in Civil Engineering* (ASCE) 1992; **6**(3):282–301.
26. Forde BW, Foschi RO, Stiemer SF. Object-oriented finite element analysis. *Computers and Structures* 1990; **34**(1):355–374.
27. Miller GR. An object-oriented approach to structural analysis and design. *Computers and Structures* 1991; **40**(1):75–82.
28. Zimmermann T, Dubois-Pèlerin Y, Bomme P. Object-oriented finite element programming: I. governing principles. *Computer Methods in Applied Mechanics and Engineering* 1992; **98**:291–303.
29. Mackie RI. Object oriented programming of the finite element method. *International Journal for Numerical Methods in Engineering* 1992; **35**:425–436.
30. Ishida E, Niimi K, Fukuwa N, Nakai S, Taga N. Object-oriented programming of dynamic finite-element analysis using substructure method. *Proceedings of Symposium on Numerical Analytical Methods for Structural Engineering*, Tokyo, Japan, vol. 17. 1993; 471–476 (in Japanese).
31. Rucki MD, Miller GR. An algorithmic framework for flexible finite element-based structural modeling. *Computer Methods in Applied Mechanics and Engineering* 1996; **136**:363–384.
32. Takahashi Y, Igarashi A, Iemura H. Application of object-oriented approach to earthquake engineering. *Journal of Civil Engineering Information Processing System* (JSCE) 1996; **5**:123–130 (in Japanese).
33. Archer G, Fenves G, Thewalt C. A new object-oriented finite element analysis program architecture. *Computers and Structures* 1999; **70**:63–75.
34. Takahashi Y, Igarashi A, Iemura H. Object-oriented analysis and design of structural analysis system. *Journal of JSCE* 2001; **I-57**(689):301–320 (in Japanese).
35. Chen HM, Archer GC. A distributed object-oriented finite-element analysis program architecture. *Computer-Aided Civil and Infrastructure Engineering* 2001; **16**:326–336.
36. Booch G, Rumbaugh J, Jacobson I. *The Unified Modeling Language User Guide Version 1.3*. Addison-Wesley: Reading, MA, 1999.
37. Rumbaugh J, Jacobson I, Booch G. *The Unified Modeling Language Reference Manual*. Addison-Wesley: Reading, MA, 1999.
38. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Object-Oriented Architecture*. Addison-Wesley: Reading, MA, 1995.
39. McKenna F, Fenves GL. An object-oriented software design for parallel structural analysis. *Advanced Technology in Structural Engineering*. Structural Engineering Institute, ASCE: Philadelphia, U.S.A., 2000.
40. Leon R, Deierlein G. Considerations for the use of quasi-static testing. *Earthquake Spectra* 1996; **12**(1): 87–109.

41. Thewalt CR, Mahin SA. Hybrid solution techniques for generalized pseudodynamic testing. *UCB/EERC- 87/09*, University of California, Berkeley, 1987.
42. Nakashima M, Kaminosono T, Ishida M, Ando K. Integration techniques for substructure pseudo dynamic test. *Proceedings of the 4th U.S. National Conference on Earthquake Engineering*, Palm Springs, U.S.A., vol. 2. 1990; 515–524.
43. Nakashima M, Akazawa T, Sakaguchi O. Integration method capable of controlling experimental error growth in substructure pseudo dynamic test. *Journal of Structural and Construction Engineering* (AIJ) 1993; **454**: 61–70 (in Japanese).
44. Combescure D, Pegon P. α-operator splitting time integration technique for pseudodynamic testing error propagation analysis. *Soil Dynamics and Earthquake Engineering* 1997; **16**:427–443.
45. Wang Y, Lee C, Yo T. Modified state-space procedures for pseudodynamic testing. *Earthquake Engineering and Structural Dynamics* 2001; **30**:59–80.
46. Nakashima M, Masaoka N. Real-time on-line test for MDOF system. *Earthquake Engineering and Structural Dynamics* 1999; **28**:393–420.
47. Nakashima M, Kato H, Takaoka E. Development of real-time pseudo dynamic testing. *Earthquake Engineering and Structural Dynamics* 1992; **21**:79–92.
48. Horiuchi T, Inoue M, Konno T, Namita Y. Real-time hybrid experimental system with actuator delay compensation and its application to a piping system with energy absorber. *Earthquake Engineering and Structural Dynamics* 1999; **28**:1121–1141.
49. Igarashi A, Iemura H, Suwa T. Development of substructured shaking table test method. *Proceedings of 12th World Conference on Earthquake Engineering* 2000; 1775.
50. Pearlman L, D'Arcy M, Johnson E, Kesselman C, Plaszcak P. *Neesgrid Teleoperation Control Protocol (NTCP)*. NEESgrid 2004-23, NEESgrid, www.neesgrid.org, 2004.
51. Welch V, Siebenlist F, Forster I, Bresnahan J, Czajkowski K, Gawor J, Kesselman C, Meder S, Perlman L, Tuecke S. Security for grid services. *12th IEEE International Symposium on High Performance Distributed Computing*, 2003.
52. Stevens R. *UNIX Network Programming*, vol. 1. (2nd edn) Prentice-Hall: Englewood Cliffs, NJ, 1998.
53. National Instruments Inc. *The NI-DAQ User Manual for PC Compatibles*, 2001.
54. Tomofuji H, Pan P, Nakashima M, Liu D. Substructure online test using displacement-force combined and switching control. *Journal of Structure and Construction Engineering* (AIJ) 2004; (585):85–92 (in Japanese).

