ENVIRONMENT INDEPENDENT IMPLEMENTATION OF A SOFTWARE FRAMEWORK FOR FAST LOCAL AND GEOGRAPHICALLY DISTRIBUTED HYBRID SIMULATIONS

A. Schellenberg[1], H. Kim[2], S.A. Mahin[3] and G.L. Fenves[4]

[1] *Ph.D. Candidate, Dept. of Civil and Env. Eng., Univ. of California, Berkeley, CA, USA*
[2] *Graduate Student, Dept. of Civil and Env. Eng., Univ. of California, Berkeley, CA, USA*
[3] *Byron and Elivra Nishkian Prof., Dept. of Civil and Env. Eng., Univ. of California, Berkeley, CA, USA*
[4] *T.Y. and Margaret Lin Prof., Dept. of Civil and Env. Eng., Univ. of California, Berkeley, CA, USA*

ABSTRACT:

Hybrid simulation provides a versatile, realistic and cost-effective method for simulating dynamic response experimentally. Although hybrid simulation is undergoing widespread development worldwide, each site generally follows a different approach to implementation, which impedes researchers from effectively collaborating. It is especially difficult to perform important geographically distributed experiments that can take advantage of unique experimental, computational and intellectual resources worldwide. This paper describes the continuing evolution and several real-time applications of the Open-source Framework for Experimental Setup and Control (OpenFresco), an object oriented, environment independent software framework which is based on a systems analysis of the operations needed to perform local as well as geographically distributed hybrid simulations. OpenFresco's software architecture is explained in detail. Examples of rapid (including soft real-time) local and geographically distributed hybrid simulations, using OpenFresco in combination with different finite element analysis software packages, are presented along with the current development status and future possible improvements.

KEYWORDS: Rapid Hybrid Simulation, Geographically Distributed, OpenFresco, Finite Element Software

1. INTRODUCTION

Considerable research is currently being conducted worldwide to extend hybrid simulation to applications where advanced numerical techniques are utilized, boundary conditions are imposed in real-time, and dynamic loading conditions are caused by seismic events, wind, blast, impact, waves, fire and traffic. Similarly, efforts are underway to optimize capabilities for testing portions of a structure in geographically distributed laboratories, including techniques to improve network performance. To date few efforts have been made to develop a common software framework for implementing and deploying systems that can carry out hybrid tests. Typically, each implementation and execution of hybrid simulation has been problem specific and used to be strongly dependent on the computational procedure, the configuration of the experiment, and the control and data acquisition systems employed at the testing site. Such highly customized software implementations are difficult to adapt to different structural problems and even harder to port to different laboratories. Thus, the lack of a common framework has become an obstacle in the field of hybrid simulation by hindering the efficient use of resources and collaboration among experts in the field.

A software framework for experimental testing should ideally support a large variety of computational software, structural testing methods, specimen types, testing configurations, control and data acquisition systems and communication protocols. Furthermore, to accelerate the development and refinement of hybrid simulation, such software framework should be environment independent, robust, transparent, scalable and easily extensible. A software framework is a reusable design for a system or subsystem and defines the overall architecture of such system, meaning its fundamental components as well as the relationships among them. It should allow domain researchers to execute hybrid simulations without specialized knowledge about the underlying software. But at the same time, it should also enable hybrid simulation and IT specialists to extend the frontiers of the methodology, by permitting the effortless addition of new developments.

2. OPENFRESCO SOFTWARE ARCHITECTURE

The Open-source Framework for Experimental Setup and Control, OpenFresco, described herein is an improved and extended version of the software framework originally presented by Takahashi (Takahashi et al., 2006). Using object-oriented methodologies, the software framework and abstractions in Figure 1 are developed after identifying the requirements and separating these requirements into components. These abstract components are called software classes. Using encapsulation, a class conceals its functional details (data and operations on data) from other objects that send messages to it and thereby provides the basis for modular, flexible and extensible software.

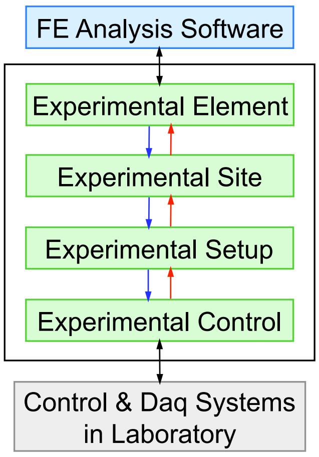


Figure 1: Abstract components of OpenFresco software architecture

The first abstraction is the *ExperimentalElement* class. It represents specimens such as trusses, beams, columns, braces, springs, walls and other parts of a structure that are physically tested in a laboratory. Analogous to an analytical element, the experimental element acts within the host finite element analysis software, but represents a physical portion of a model instead of a numerical one. The basic functionality of an experimental element is to provide the mass, damping and stiffness matrices as well as the residual force vector for a given deformation state. The experimental element is responsible for transforming prescribed boundary conditions from the global coordinate system of the FE-model into the local or basic element coordinate system. Additionally, it needs to provide inverse transformations for the measured response quantities, such as displacements, velocities, accelerations and forces, from the element coordinate system back to the global coordinate system of the FE-model.

The *ExperimentalSite* class represents laboratories and computational sites in the software framework. Since in the real world, laboratories and computational sites can physically be in different places, the experimental site objects in the software are responsible for providing communication methods to connect the different sites and store data that is received from or needs to be sent to other sites. Furthermore, the experimental sites should support different communication protocols such as TCP/IP, UDP or NHCP (NEES Hybrid-simulation Control Protocol) and be able to guarantee secure transactions over the Internet using cryptographic protocols such as TLS (Transport Layer Security) or SSL (Secure Sockets Layer).

The third abstraction down the chain is the *ExperimentalSetup* class. Such class is responsible for representing the possible configurations of the transfer systems constructed in the laboratories. Since a transfer system consists of actuators and measuring devices, the experimental setup needs to transform the prescribed boundary conditions from the local or basic element degrees-of-freedom of the experimental elements into the actuator degrees-of-freedom of the transfer system, utilizing the geometry and the kinematics of the loading and instrumentation system. Similarly the work conjugates measured by transducers and load cells need to be transformed back to the experimental element degrees-of-freedom. These transformations can be implemented either as simple linear transformation matrices (small displacements) or they can be implemented as complex algebraic transformations taking large displacement effects into account.

The last one of the main abstractions introduced here is the *ExperimentalControl* class. Since one of the required tasks for experimental testing is the communication with a wide variety of control and data acquisition systems, the *ExperimentalControl* class provides such interface and functionality in the software framework. The advantage of this abstraction and the encapsulation of these operations is that the *ExperimentalSetup* class separates the details of the loading system configuration from the *ExperimentalControl* class.

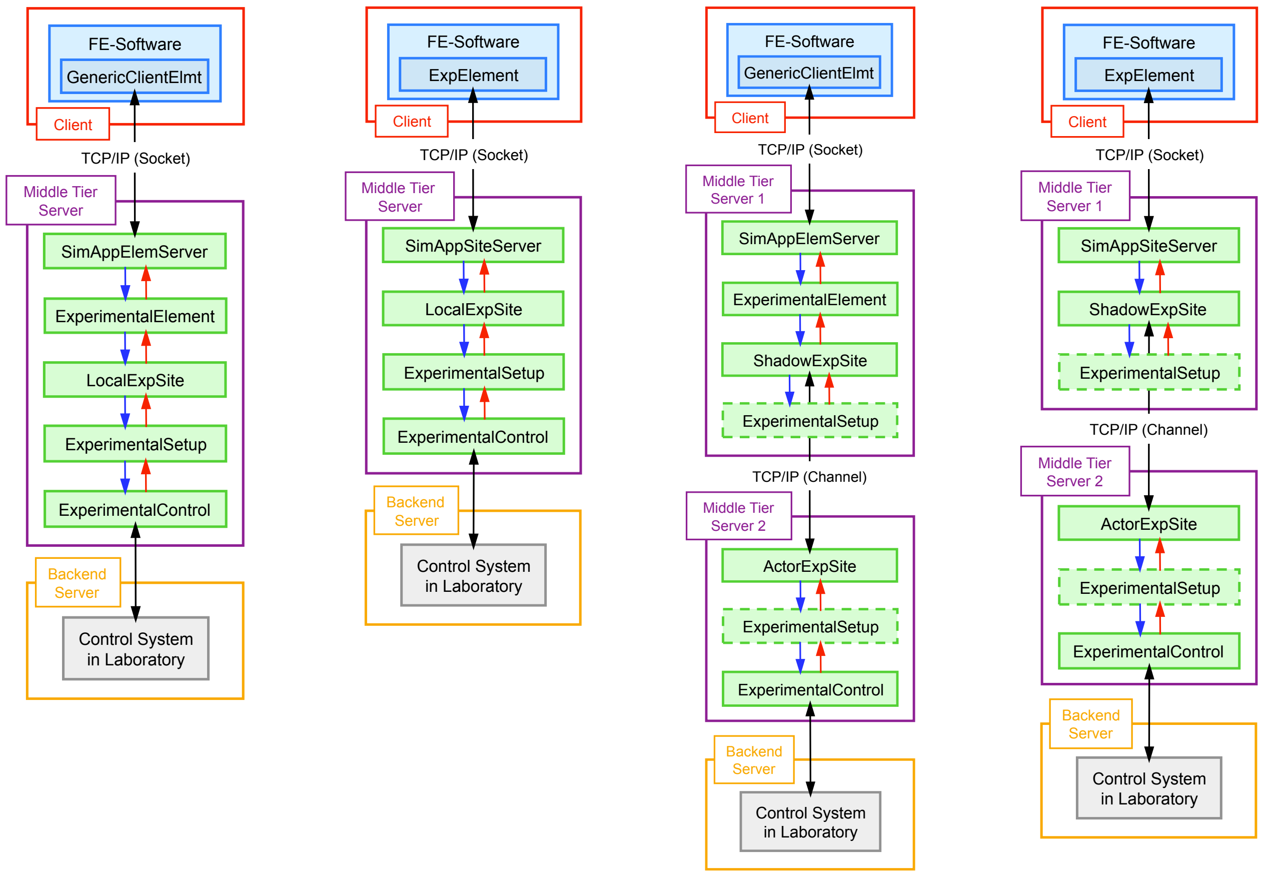
3. MULTI-TIER SOFTWARE ARCHITECTURE

To provide a modular and flexible way to utilize OpenFresco with any finite element analysis software of the user's choice, the three- or multi-tier software architecture is employed (Carnegie Mellon SEI, 2008). The three-tier software architecture wraps around the OpenFresco core modules described in Section 2 and provides the necessary functionality to interface with a wide variety of finite element software clients and control and data acquisition systems. The software is divided into three layers, a client, a middle tier (or application) server and a backend server. The middle tier server is located between the user interface (client) and the data management (server) components.

In the case of the experimental software framework, the finite element analysis software resembles the client or user-interface top tier, where the structural model is created and then analyzed. The backend server or the third tier is the laboratory server composed of the different control and data acquisition systems with their APIs. The middle tier server, which is also called simulation application server, provides process management services and data transformations so that different computational clients can query information from the different experimentally tested portions of a structure. In addition, due to this separation provided by the middle tier server, finite element analysis software written in different languages such as Matlab, Fortran, C, C++ etc. can be accommodated. If the middle tier server is further divided into two or more units, the design is referred to as multi- or n-tier architecture. For geographically distributed simulations, OpenFresco is employing a four-tier architecture with two middle tier server units.

In the case of a local deployment of OpenFresco, meaning that all processes are executed locally in one laboratory, there are two possible configurations available as shown in Figure 2(a). In the left configuration, which is the more general one, a generic client element has to be added to the finite element software that is being used for the analysis. The generic client element only requires the number of nodes, the degrees-of-freedom it is connected to and the port to communicate through as input parameters. Thus, data exchange with the OpenFresco middleware takes place through the generic client element, embedded by the user into each finite element analysis program using their published programming interfaces (e.g., user-defined elements). Since a generic element is used in the finite element analysis software, the simulation application element server then interfaces with the OpenFresco experimental element class. The advantage of this first configuration is that only one element, the generic client element, has to be added to the finite element analysis software and all the existing experimental elements in the OpenFresco software framework can be utilized to represent the experimental portions of a structure. In the right configuration in Figure 2(a), an experimental element instead of a generic client element is directly added to the finite element analysis software. This experimental element communicates with the simulation application site server in OpenFresco, which in turn interfaces with the local experimental site instead of the experimental element. The advantage of the second configuration is, that if a very specialized experimental element is needed, which is not available in and cannot be added to OpenFresco, such experimental element can directly be added to the computational software client.

As can be seen from Figure 2(b), in the case of a distributed deployment of OpenFresco, there are two very similar possible configurations available. The main difference being that the middle tier server is divided into two units that are distributed across a network. Since the middle tier server is divided into two units, the *ExperimentalSetup* class can either be located on the first middle tier server unit (usually running on the same machine as the finite element analysis client) or on the second middle tier server unit (usually running in the laboratory) but not on both sides simultaneously. In summary, OpenFresco's three- and four-tier architectural patterns provide the modularity and flexibility to interface any computational agent running on one or more machines with physical specimens being tested in one or more laboratories.

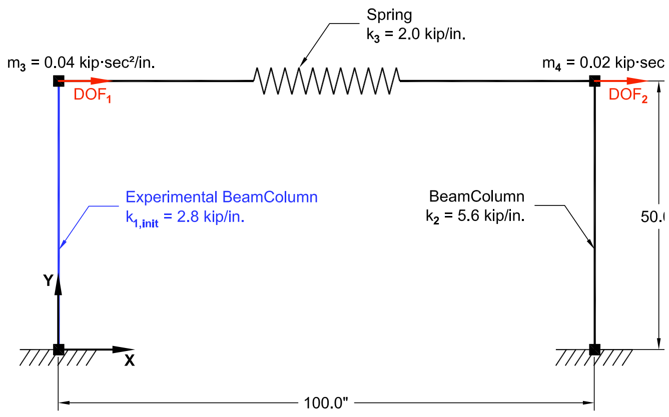


(a) Local deployment (b) Network deployment

Figure 2: Three-tier configurations for local deployment

4. RAPID HYBRID SIMULATION USING OPENFRESCO

This section illustrates how OpenFresco was utilized for rapid local and geographically distributed hybrid simulations. For its simplicity, a one-bay-frame model with one ductile and one elastic column connected by a fairly soft spring was selected for these hybrid simulations. The structural properties of the model are shown in Figure 3(a). The one-bay-frame had a height of 50", which corresponds to the height at which the actuators attached to the experimental steel specimen. Given that only the horizontal degrees-of-freedom were considered, this simple one-bay-frame model had a total of two free degrees-of-freedom, both with mass. The resulting vibration periods of the model were $T_1 = 0.622$ sec and $T_2 = 0.315$ sec for the first and second mode, respectively. To not only get contributions to the total response from the first mode, but also the second mode, the linear-elastic spring was given a fairly low stiffness. In addition, the damping matrix was chosen to be proportional to the mass matrix instead of the stiffness matrix, so that the second mode would be less damped than the first. The first mode was assigned a damping value of 5% of critical, which led to a 2.53% damping ratio in the second mode. The north-south component of the El Centro ground acceleration history recorded during the 1940 Imperial Valley earthquake was used to dynamically load the model. The 32 sec long time history consisted of 1600 data points recorded every 0.02 sec and was scaled to a pga of 0.319 g.



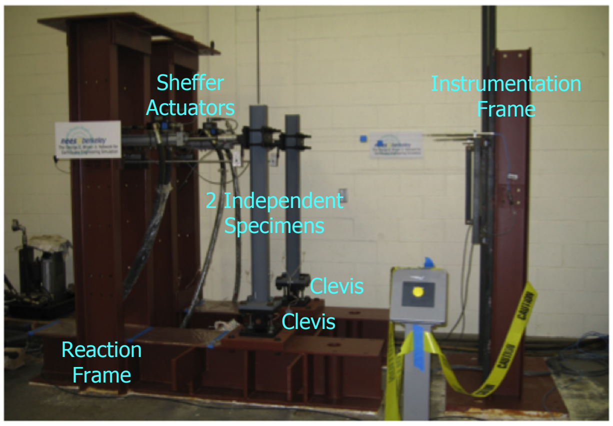


Figure 3: One bay frame model with structural properties and µNEES setup

4.1. Rapid Local Hybrid Simulation

The local hybrid simulation was conducted using the µNEES laboratory (Figure 3(b)) of the nees@berkeley equipment site in Richmond, California, which is a part of the University of California, Berkeley. The 50" high experimental S4x7.7 steel column specimen was mounted on top of a special clevis with two ductile replaceable steel coupons that was capable of simulating the plastic hinge zone at the end of a steel section. A 12 kip dynamic actuator was used to load the specimen during the tests.

OpenSees (Fenves et al., 2004), LS-Dyna, Matlab, and UI-SimCor (Kwon et al., 2007) were used as computational drivers to demonstrate that OpenFresco can be used with various finite element software packages. All of the computational software used a custom experimental element with the simulation application site server. The explicit Newmark method was chosen for the integration of the equations of motion in OpenSees, LS-Dyna, and Matlab while the Alpha-OS method was used in UI-SimCor. The OneActuator experimental setup object was employed along with the xPCtarget experimental control object to connect to the real-time xPC-Target machine, which was running the event-driven predictor-corrector model.

4.2 Rapid Geographically Distributed Hybrid Simulation

The latest advancements in networking technology paired with the experimental software framework OpenFresco make fast geographically distributed hybrid simulations lastly feasible. For the fast geographically distributed hybrid simulations which were performed between NEESinc, Headquarter in Davis, California and the µNEES laboratory in Richmond, California one of the 1 Gbit/sec connections of such network was utilized. The OpenSees finite element software was used to model and analyze the one-bay-frame structure on a portable computer at the NEESinc, Headquarter. The same µNEES setup from the Section 4.1 was utilized to test the experimental portion of the model and OpenFresco was employed to connect the two together across the sixty miles long network connection. The OpenSees finite element analysis software in combination with OpenFresco's tightly integrated experimental element and experimental site objects composed the client side of the distributed hybrid simulation. The explicit Newmark method was chosen for the integration of the equations of motion in OpenSees. The server side of the hybrid simulation consisted of OpenFresco's experimental site, experimental setup and experimental control objects. Experimental setup and control were identical to the ones used in Section 4.1. The predictor-corrector model was responsible for handling possible delays caused by the non-deterministic TCP/IP network transactions and the direct integration of the equations of motion.

The TCP_Socket object was utilized to setup a persistent TCP/IP connection from the client application to the server application. The TCP/IP connection guaranteed that the transmitted messages were never lost, damaged or received out of order. In addition, the persistent connection provided significant performance gains, since the TCP connection setup delay caused by the client server handshake only occurred once during initialization of the distributed test, rather than at the beginning of every single transaction. Besides the use of a persistent TCP/IP connection, the size of the data packages sent to the TCP stack also significantly affected the network performance. Since the TCP protocol has a data stream interface that permits applications to send data of any size to the TCP stack, it was imperative to determine a TCP segment size that would produce the best possible network performance. By increasing the size of the data packages that were transmitted between the Shadow Experimental Site and the Actor Experimental Site past the minimal size necessary for the response vectors, the so called "sender silly window syndrome" was avoided. Sending a storm of tiny data packages is very inefficient and can even disrupt other network traffic (Gourley et al., 2002).

Since the goal in this case study was to perform a hybrid simulation in real-time, the simulation time step, Δt_{sim}, had to be set equal to the integration time step, $\Delta t_{int} = 20$ msec. However, because the simulation time step had to be a multiple of the controller time step, $\Delta t_{con} = 1/1024$ sec, this was not exactly possible. It was decided to choose 20 substeps for each simulation time step. This yielded $\Delta t_{sim} = 19.53125$ msec, as long as the network and integration tasks were capable of executing in less than 60% of the predefined simulation time step. Thus, the theoretical testing rate was 0.9766, which turned out to be slightly faster than real-time. The predictor-corrector model automatically slowed down the actuator movement if the combined network and integration time exceeded 60% of the predefined simulation time step. To achieve optimal performance this condition needed to arise as little as possible.

4.3. Rapid Hybrid Simulation Results

For brevity, only the handpicked results from the geographically distributed test are provided. Since the local hybrid simulation used the same structural model, ground motion and experimental setup, the results were similar to the ones in this section.

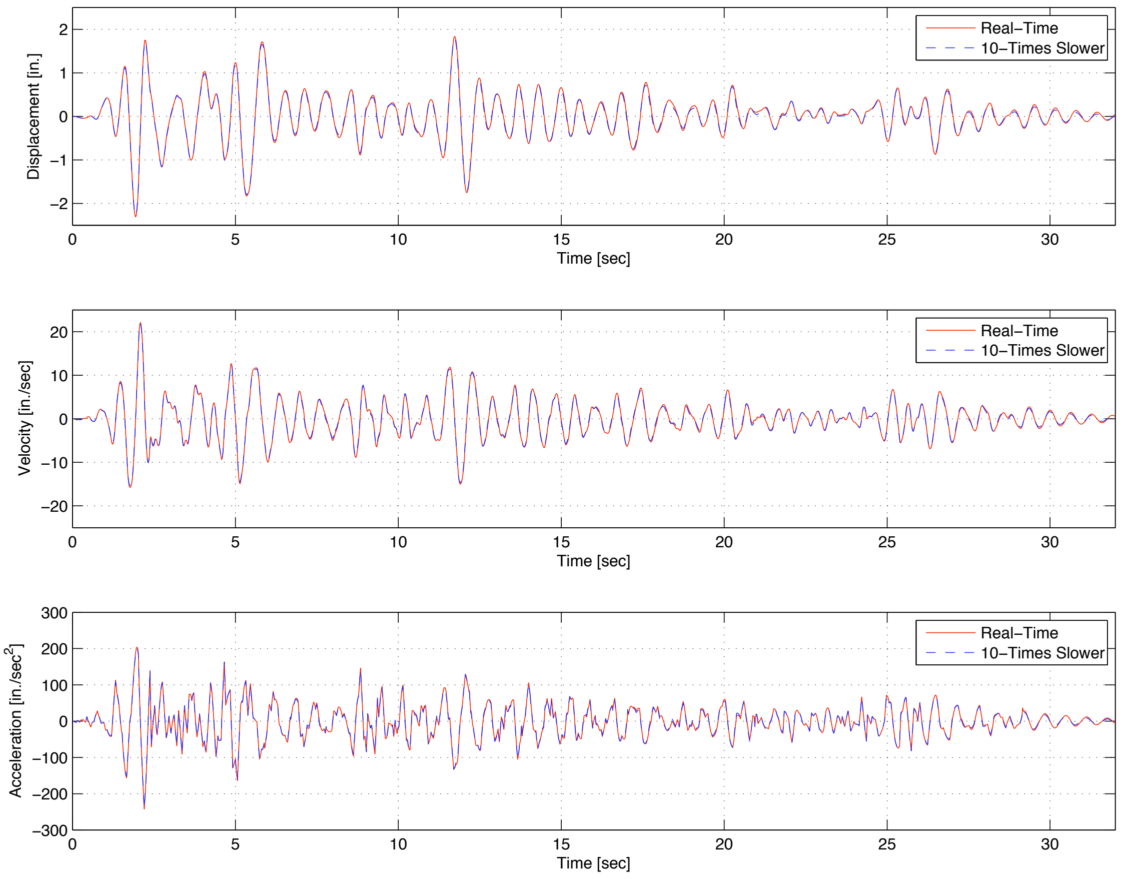


Figure 4: Comparison of response histories for real-time versus slow hybrid simulation

Figure 4 shows the comparison of the lateral displacement, velocity and acceleration response of the first free degree-of-freedom of the one-bay-frame structure between the real-time test and the ten-times slower test. The largest displacement, velocity and acceleration seen on top of the left experimental column were 2.31 in., 22.1 in./sec and 242 in./sec^2, respectively. Because the top of the left experimental column was attached to the actuator of the µNEES setup and additionally the test was executed in real time, the actuator experienced the same response maxima. Consequently, such extremes had to remain inside the actuator displacement limits of ±7.5 in. and the actuator velocity limits of ±23.3 in./sec, in order not to saturate the response or trigger the interlocks and shutdown the control system. The displacement, velocity and acceleration response at the top of the left experimental column were almost identical. This means that the effect of the testing rate for this specific test and structure was negligible and can be ignored.

As can be seen from Figure 5, the resisting forces, which were measured by the load cell and fed back to OpenSees, are reasonably smooth except at one short instance in time. So even though the hybrid simulation was executed in real-time with high velocities and accelerations, force oscillations stayed small. This can be explained by the very small weight (49 lb) of the physical components that the actuator had to move forth and back. If the weight had been larger, significant inertia forces would have been generated which in turn would have affected the force measurements to a much larger degree.

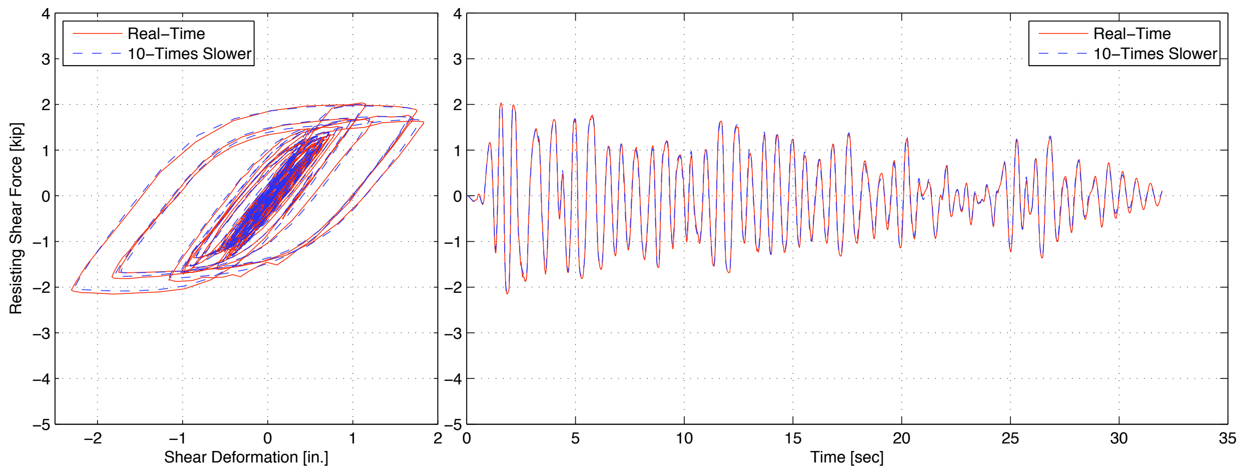


Figure 5: Comparison of hysteresis loops and force histories for real-time versus slow hybrid simulation

5. FUTURE CAPABILITIES

The recent release of OpenFresco, version 2.6, contains new innovative features and performance enhancements. It includes a robust feature allowing the user to couple two different finite element programs in a seamless way. This capability is encapsulated in the new adapter element (Schellenberg et al., 2008). OpenFresco is now fully compatibility with the latest releases of Mathworks xPC-Target (version 3.4) and MTS CSI software. A new four actuators experimental setup is added. The experimental control objects now have the ability to filter signals.

There are many possible improvements to future releases of OpenFresco to incorporate the ongoing research in hybrid simulation. One would be the ability to perform mixed displacement and force control and to switch between displacement and force control. During real time testing, there is a need to account for the contribution of the inertia of the specimen. A phantom or ghost element could mirror the physical specimen, which would be helpful in cases where the communication with the control system is either delayed or interrupted. The capability of having multiple experimental elements in one experimental site may enable labs that have one control system to have multiple experimental setups.

7. CONCLUSIONS

The software framework for experimental testing (OpenFresco), which has been presented in this paper, supports a wide range of computational software, structural testing methods, specimen types, testing configurations, control and data acquisition systems and communication protocols. The development of such software framework for hybrid simulation is based on the numerical analysis approach, in which a hybrid simulation can be viewed as a conventional finite element analysis where physical models of some portions of the structure are embedded in the numerical model. Object-oriented methodologies and a rigorous system analysis of the operations performed during hybrid simulations were used to determine the fundamental building blocks of the software framework. In order to support any finite element analysis software as the computational driver, a multi-tier client/server architecture was wrapped around those fundamental OpenFresco building blocks. This design approach, which guarantees that the software framework is environment independent, robust, transparent, scalable and easily extensible, has been explained in detail. A demonstration of how OpenFresco can be used for soft real-time local and geographically distributed hybrid simulations was provided. Finally, this paper explained the latest developments and the future direction of OpenFresco.

ACKNOWLEDGMENTS

Funding for this work was provided in part by the National Science Foundation through a subaward from the NEES Inc consortium under Cooperative Agreement CMS0402490. The authors appreciate the assistance of Prof. Yoshikazu Takahashi of Kyoto University, and Dr. Frank McKenna of UC Berkeley. The findings and conclusions are those of the authors alone, and may not reflect those of the sponsors.

REFERENCES

Carnegie Mellon SEI (2008). http://www.sei.cmu.edu/str/descriptions/threetier_body.html.

Fenves, G. L., McKenna, F., Scott, M. H., and Takahashi, Y. (2004). An object-oriented software environment for collaborative network simulation. Proceedings, 13th World Conference on Earthquake Engineering, Vancouver, Canada.

Gourley, D. and Totty, B. (2002). HTTP: The Definitive Guide. O'Reilly Media, Inc. Sebastopol, CA, United States.

Kwon, O.-S., Nakata, N., Park, K.-S., Elnashai, A., and Spencer, B. (2007). User Manual and Examples for UI-SIMCOR and NEES-SAM, University Of Illinois, Urbana-Champaign, IL, United States.

McKenna, F. T. (1997). Object-oriented finite element programming: frameworks for analysis, algorithms and parallel computing. Ph.D. Thesis, University of California, Berkeley, CA, United States.

Schellenberg, A., Huang, Y. and Mahin S. A. (2008). Structural FE-Software Coupling Through the Experimental Software Framework, OpenFresco. Proceedings, 14th World Conference on Earthquake Engineering, Beijing, China.

Takahashi, Y. and Fenves, G. (2006). Software Framework for Distributed Experimental-Computational Simulation of Structural Systems, Earthquake Engineering and Structural Dynamics, 35(3), 267-291.