CU-NEES-08-1

NEES at CU Boulder
01000110 01001000 01010100
The George E Brown, Jr. Network for Earthquake Engineering Simulation

Report to NEESinc
Ref.: OMSA-2006-SSL-UCoB

Site Implementation and Evaluation of OpenFresco and SIMCOR at CU-NEES FHT Laboratory

by

Dr. Gary Haussmann
Technical Director

January 21, 2008

January 2008

Department of Civil Environmental and Architectural Engineering
University of Colorado
UCB 428
Boulder, Colorado 80309-0428

Contents

1 Objective **3**

2 Preparation of *CU-NEES* site **3**

3 Initial Installation and Testing of *OpenFresco* and *SIMCOR* **3**

4 Integration of *OpenFresco* with Production System **4**

5 Testing and Validation of *OpenFresco* **6**

6 Testing and Validation of *SIMCOR* **6**

7 Distributed Multi-Site Test **10**

8 Construction of a Hard Real-Time System for Hybrid Testing **11**

9 Summary and Conclusions **14**

A OpenFresco and SIMCOR Initial Validation: OneBayFrame Example Client **17**

B Multi-site Test: Configuration File for Two-Story Structure **21**

C Realtime Validation: Configuration file for old realtime software **25**

D Realtime Validation: Configuration file for new realtime software **27**

Abstract

As part of ongoing activities to maintain and update the hybrid capabilities of the *CU-NEES* site, two hybrid software tools were installed and configured to work with the *CU-NEES* site-specific hardware. The two software tools, *OpenFresco* and *SIMCOR* , enable distributed hybrid simulation and compatibility with other *NEES* sites. In order to provide hard real-time hybrid capabilities several parts of the *OpenFresco* software framework were modified or augmented. Hybrid tests performed for a variety of test set-ups and situations were performed in conjuction with other *NEES* to check the functionality and correctness of the hybrid test software.

1 Objective

This report documents the processes and activities involved with executing the plans present in "Proposal for Site Implementation and Evaluation of *OpenFresco* and *SIMCOR*". The original proposal proposed installing, localizing, and testing the hybrid simulation toolkits *OpenFresco* [2] and *SIMCOR* [3] at the *CU-NEES* site, making any changes or configuration needed for the specific needs and equipment at *CU-NEES* . The intended end result of this effort is to make both *OpenFresco* and *SIMCOR* available for performing hybrid simulation in addition to the hybrid simulation software already present at the *CU-NEES* site.

By making *OpenFresco* and *SIMCOR* available, the *CU-NEES* site expands its hybrid simulation capability to allow for interoperability with other *NEES* sites which also implement *OpenFresco* and/or *SIMCOR* . Previously, the *CU-NEES* site has used heavily customized software to enabled hard realtime hybrid simulation [7], and evaluated the possible used of *OpenFresco* for fast and/or realtime hybrid testing [6, 1]. Compatibility and interoperability with other *NEES* sites is a key motivation for replacing the custom software used at *CU-NEES* with versions of *OpenFresco* and *SIMCOR* . However, in order to maintain the current realtime capabilities of the *CU-NEES* site a new version of the custom hybrid simulation software used at *CU-NEES* will be constructed based on *OpenFresco* software components. This customized version of *OpenFresco* is specifically intended to satisfy the hard realtime [1] requirements currently imposed for realtime tests at *CU-NEES* while still maintaining compatibility with other *NEES* sites. While the current version of *CU-NEES* custom software supports realtime hybrid testing, the custom software was not designed for use in distributed simulation and therefore does not interoperate with the hybrid software used at other *NEES* sites.

2 Preparation of *CU-NEES* site

Several activities were performed in preparation for the localization effort. The software for the MTS controller was upgraded to the most recent version. Several legacy computers were replaced or upgraded. Up-to-date versions of Microsoft Visual Studio and Mathworks MATLAB were installed. To replace the ETS Operating System used on the Real-time target, several Real-time Operating Systems (RTOS) were examined including VxWorks, Blue Cat Lynux, and Debian GNU/Linux with preemption patches.

3 Initial Installation and Testing of *OpenFresco* and *SIMCOR*

Initial installation of the software packages involved a fake hybrid test; instead of using an actual physical specimen to generate responses a simple program is used as a stand-in for the physical specimen, providing data values similar to those of an actual physical specimen. Once this was accomplished, the specific details and possible modifications involved to perform a full hybrid test at the *CU-NEES* site were examined.

Initial testing of *OpenFresco* first required obtaining copies of the relevant software. In order to use the most up-to-date versions, source code for both *OpenSEES* and *OpenFresco* were checked out of the development repositories. The versions used were Version 1.7.4 of *OpenSEES* and Version 2.5 of *OpenFresco* . After the software was installed, the next step involved getting a set of configuration files for example tests and using these configuration

[1] Hard real-time simulation imposes strict timing requirements of the software [4, 5] that are not provided by *OpenFresco* running on standard operating systems

files to run a local test (on one computer) or a distributed test (on two computers connected by the internet). Using the provided instructions, running these example tests was fast and straightforward, although it should be noted that the *CU-NEES* site has had extensive previous experience with OpenSEES and *OpenFresco* . A new user without much experience with the TCL interpreter and command line interfaces may have problems understanding and modifying the required input files. Specifically, users have to learn the basics of the TCL interpreter as well as the multilayered architecture of *OpenFresco* . In addition, all the examples use numeric identifiers to name various object instances, resulting in lines such as

```
expSetup OneActuator 1 -control 1 1
```

where the first "1" names the `expSetup` object, the second "1" is the name of the controller to use, and the third "1" does not name an object but instead specifies which in direction the controller operates. While this reflects the long legacy of earlier simulation tools which could only use numbers to distinguish different nodes and elements, the use of numbers in this case (and used in a similar manner by *SIMCOR* , where the modules can only be referred to by number) is confusing and nonideal.

The initial testing of *SIMCOR* was performed by simply downloading the set of MATLAB script files and some examples, and then running specific examples from MATLAB. Users without much MATLAB experience may find the configuration and executing of *SIMCOR* confusing, since it requires editing a MATLAB file, making the MATLAB current directory the same as the configuration file, and then starting *SIMCOR* from within MATLAB. However, once *SIMCOR* is initiated from MATLAB, the user interface is very intuitive.

4 Integration of *OpenFresco* with Production System

For both *OpenFresco* and *SIMCOR* validation tests, the local control method was chosen to be *OpenFresco* , since the xPC-based *OpenFresco* module—which uses a MATLAB xPC real-time operating system to run the module that drives the physical specimen (see figure 1)—provides a good fit with the equipment setup at *CU-NEES* . The hardware and network configuration is shown in figure 1. The integration process was as follows:

1. In addition to the physical specimen and actuator controller/sensors, this setup required a host computer to contain the main *OpenFresco* software and a target computer to contain the xPC real-time environment.

2. The *OpenFresco* software application and associated files were downloaded from NEESforge using the *SubVersion* version control system. By basing the local *CU-NEES* copy of the software off the master copy at NEESforge, bug fixes and feature updates can be downloaded and integrated into the local CU copy fairly quickly.

3. The installation also required certain supplementary software needed to compile and run *OpenFresco* , specifically: MATLAB/Simulink, MATLAB Real-time Workshop, MATLAB/xPC Real-time environment, the MATLAB xPC development files, Microsoft Visual Studio 2005, the TCL/TK interpreter, and several header files from *OpenSEES* .

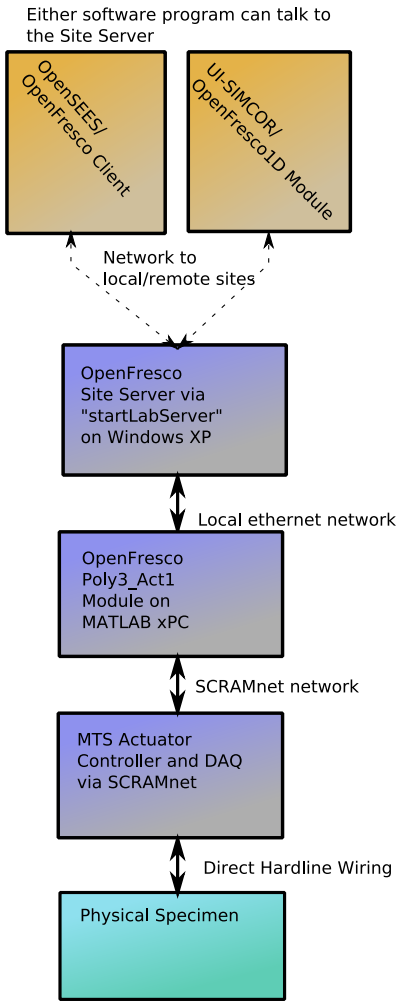

Figure 1: Using an OpenFresco site server to provide access to the physical specimen

4. The main *OpenFresco* software–the portion used to configure the lab environment and connect to other *NEES* sites–was compiled using Microsoft Visual Studio. This produced a program Openfresco.exe which was used to run the various input files such as OneBayFrame_Server1a.tcl.

5. The *OpenSEES* xPC controller module must be built in MATLAB/Simulink to produce a controller that will run in the xPC Real-time environment. This module requires some modification based on the configuration of the MTS controller and the desired actuator(s) that are installed in the laboratory.

6. The host system containing *OpenFresco* required some change in order to make it accessible to computers at other *NEES* sites. Specifically, the host system requires a static IP address as well as a firewall exception so that remote sites can connect to and communicate with the host system.

Upon finishing the setup a simple hybrid test was performed with actuators off to verify that data flow was proceeding through the entire framework, from software analysis to MTS controller. The simple test was performed using *OpenSEES* /*OpenFresco* running a modified version of the *OneBayFrame* (figure 2) example that is included in the *OpenFresco* framework. In this test *OpenSEES* simulates a simple structure and interfaces with the physical specimen via the *OpenFresco* Site Server.

A second test was run using *SIMCOR* as the coordinator and communicating with the *OpenFresco* Site Server. The *SIMCOR* software includes a OpenFresco1D module which allows it to interface with physical specimens via *OpenFresco* . Since the *OpenFresco* Site Server can be accessed by both *OpenFresco* and *SIMCOR* it seems prudent to use the *OpenFresco* Site Server as a favored setup at *CU-NEES* for distributed and non-realtime hybrid tests.

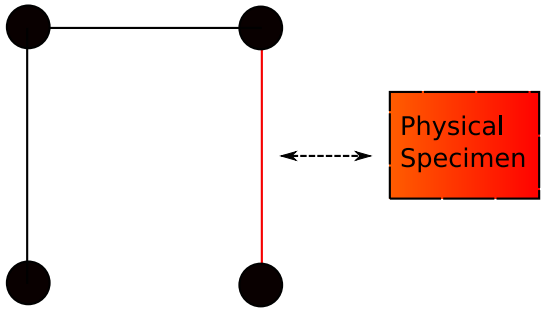

Figure 2: The OneBayFrame example used for the bulk of the *OpenFresco* tests. The red element is the substructure element which is linked to a physical specimen.

5 Testing and Validation of *OpenFresco*

In order to fully test out the functionalty of the *OpenFresco* components, a series of tests was performed to exercise both the local and distributed hybrid testing capabilities of *OpenFresco* . The structure and control configuration used for all tests in this section was the *OneBayFrame* example provided in the *OpenFresco* software package, described in the appendix on page 27.

The tests started out as a simple simulation, becoming progressively more complicated (figure 3):

1. First was a local test involving *OpenSEES* and *OpenFresco* which was pure computational analysis; that is, no actuator or physical lab was involved. Instead running lab equipment, an *OpenFresco* SimUniaxialMaterial was used to model an elastic element.

2. Next was a local-only test involving the live actuator at *CU-NEES* . *OpenSEES* analyzes a computational model and using *OpenFresco* as a client to connect to an *OpenFresco* server running the xPCControl, which uses the xPC Real-time environment to interface to a Physical Specimen.

3. Last was a distributed test involving Berkeley-*NEES* and *CU-NEES* At Berkeley-*NEES* , *OpenSEES* performs the numerical analysis and uses *OpenFresco* as a client to connect to an *OpenFresco* server at *CU-NEES* . At *CU-NEES* , an *OpenFresco* server runs the *OpenFresco* xPCControl which uses the xPC Real-time environment to interface to a Physical Specimen.

From a numerical analysis standpoint all three tests are equivalent and should produce the same results. The three tests differ in the location where certain software runs (local or distributed) and how the response force is generated (from computer code or actuator measurements). Results show very good agreement between the three tests. A plot comparing the Nodal displacement of the various tests is shown in figure 4.

The tests were performed with the help of Andreas Schellenberg and Catherine Whyte at UC-Berkeley.

6 Testing and Validation of *SIMCOR*

In order to verify that the *SIMCOR* software installation could correctly run a simulation and communicate with other sites as part of a distributed tests, we performed a set of three tests.

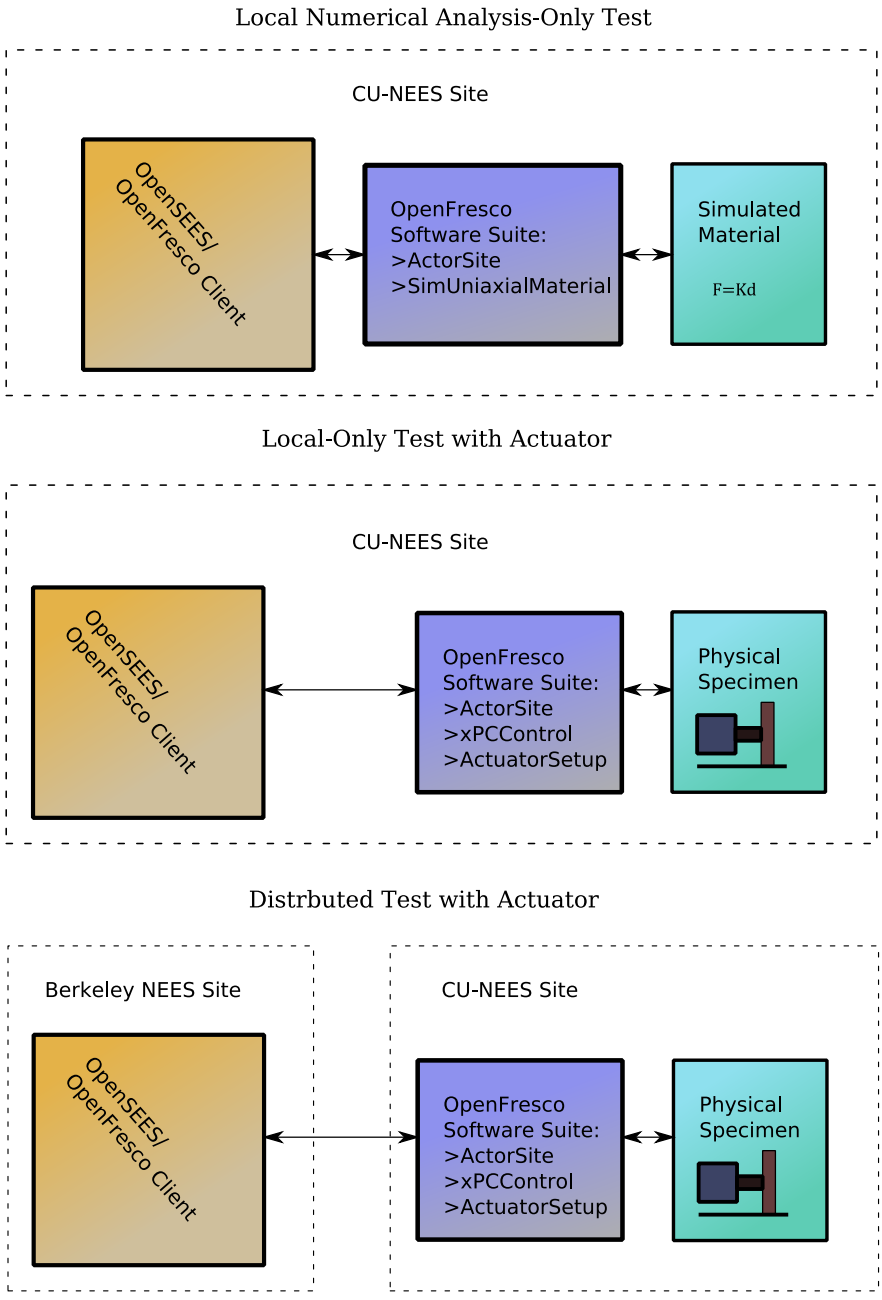

Figure 3: The three successive tests performed during *OpenFresco* Testing

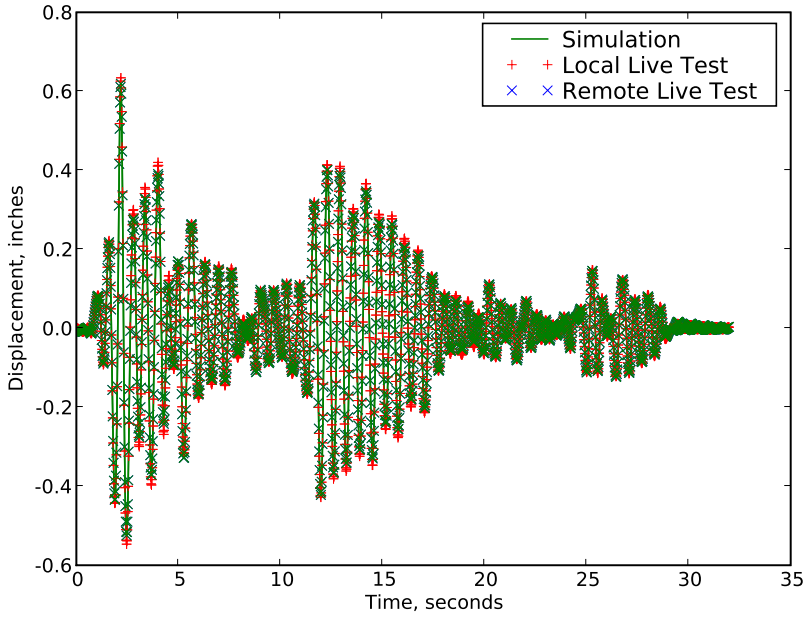

Figure 4: Comparison of simulation data from the three successive OpenFresco-based hybrid tests

The tests started out as a simple simulation, becoming progressively more complicated:

1. A local test involving *SIMCOR* as pure simulation and analysis; that is, no actuator or physical lab was involved. Instead running lab equipment, an *OpenFresco* SimUniaxial-Material was used to model an elastic element.

2. A local test involving SIMCOR for integration and the *OpenFresco* xPCControl, which uses the xPC Real-time environment to interface to a Physical Specimen.

3. A distributed test involving UIUC–*NEES* and *CU-NEES* At UIUC–*NEES* , *SIMCOR* performs the integration and coordination. At *CU-NEES* , an *OpenFresco* server runs the *OpenFresco* xPCControl which uses the xPC Real-time environment to interface to a Physical Specimen (figure 6).

From a numerical analysis standpoint all three tests are equivalent and should produce the same results. The three tests differ in the location where certain software runs (local or distributed) and how the response force is generated (from computer code or actuator measurements).

In this case, the physical specimen was a large-scale hydraulic actuator in the *CU-NEES* lab. Due to the impact on other projects taking place in the lab, a separate test rig was not built up specifically for this test. Instead, the actuator in use for a concurrent experiment was reconfigured and re-used. In lieu of an actual specimen to provide the measured force response, a pseudo force was generated from the measured displacement and a predefined stiffness K_p. This setup allowed us to maintain the equipment in-place for other tests taking place in the lab, allowing for fairly quick setup and breakdown times involving the tests performed here.

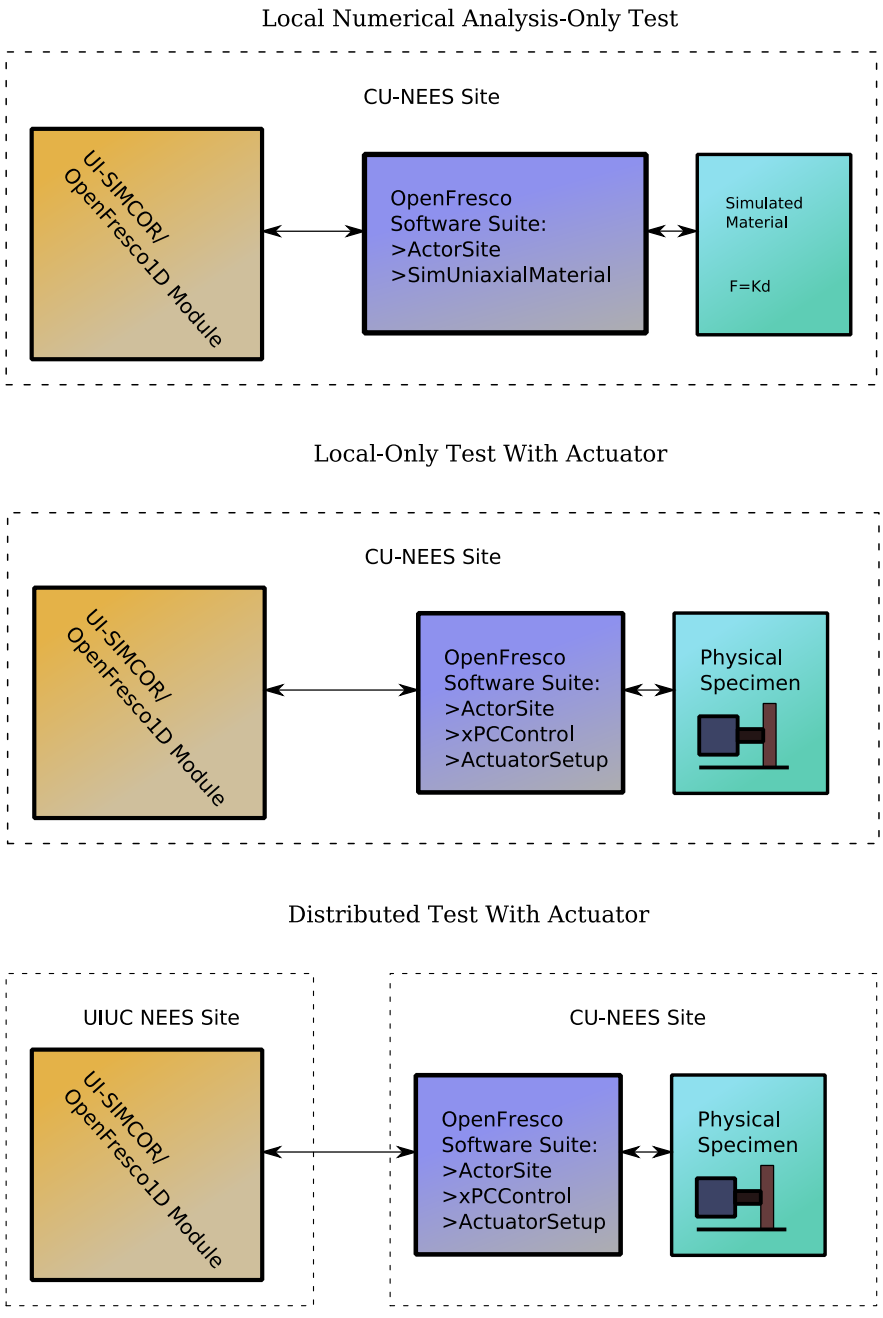

Figure 5: The three successive tests performed during *SIMCOR* testing.

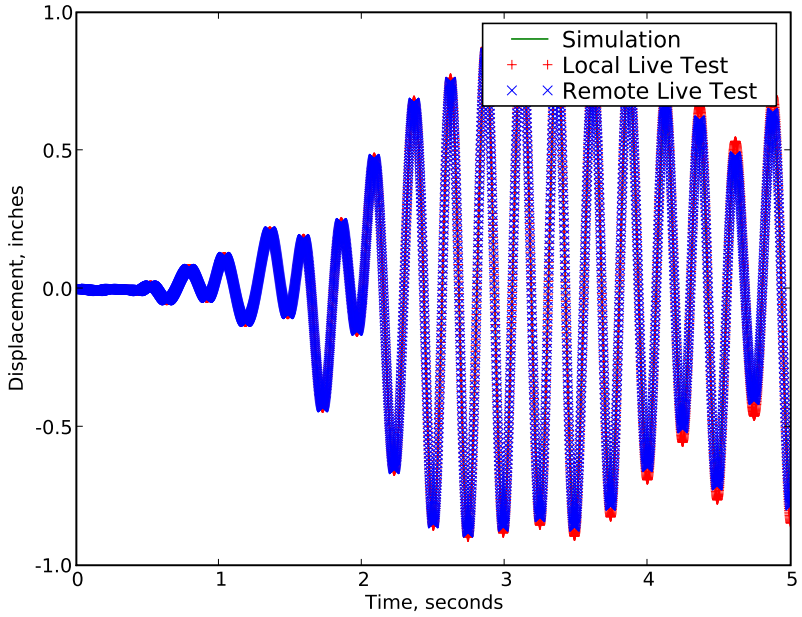

Figure 6: Comparison of local and remote SIMCOR-based hybrid tests

The distributed test was performed as a joint test between UIUC-*NEES* and *CU-NEES* . There were some initial problems establishing a connection between *SIMCOR* at UIUC and the *OpenFresco* server at CU. These problems were exacerbated by the lack of or incorrect diagnostic information. In one case, the *SIMCOR* module said it had connected to the CU server when no network traffic from UIUC was arriving at the CU server. A better way to test, start, and possibly restart the network connection for a distributed test is clearly desired.

The primary source of error is due to the displacement measurement; both noise and actuator lag produce a measured displacementand thus a measured forcethat does not exactly match the values produced by the ideal response provided by the SimUniaxialMaterial in *OpenFresco* .

The distributed test was performed with the help of Kyu-Sik Park at UIUC. We appreciate his taking the time and effort to help CU with this test.

7 Distributed Multi-Site Test

As part of the localization project, a multi-site distributed test was conducted involving *CU-NEES* , Lehigh-*NEES* , and the University of Connecticut. This test consisted of one site–University of Connecticut (U-Conn)–that performed analysis using *OpenSEES* and communicated with the *CU-NEES* and Lehigh-*NEES* sites via *OpenFresco* . Both the *CU-NEES* and Lehigh-*NEES* applied displacement commands provided by U-Conn and supplied force feedback values back to U-Conn.

The first attempt involving the three sites encountered problems due to apparent incompatibilites between *OpenFresco* versions. In short, *CU-NEES* was using *OpenFresco* version 2.5 while the other two sites were using *OpenFresco* version 2.0. The test was rescheduled for the following week while version upgrades were performed.

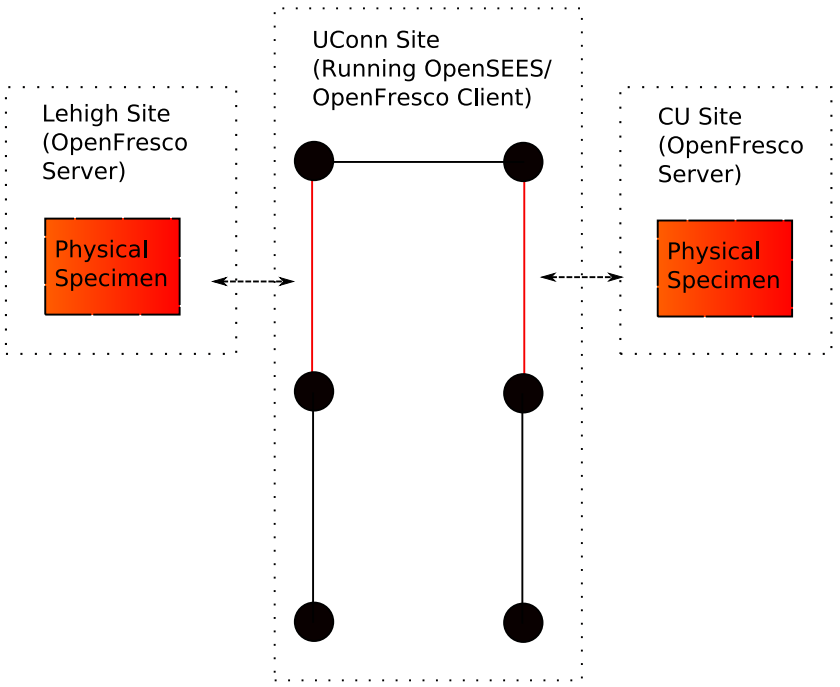

Figure 7: Schematic of the structure used in the multi-site test, illustrating what portion of the test each site was responsible for.

On the second attempt, attempts to run a hybrid test would result in *OpenSEES* locking up after running two or three timesteps. Use of network tools showed the network connection between U-Conn and *CU-NEES* was being severed prematurely. Several network tests were done and established that the Internet Protocol port 80 would not properly connect between U-Conn and *CU-NEES* . After modifying the ports used, a successful distributed multi-site simulation was performed. The test structure used was a two-story building based on the *OneBayFrame* structure (figure 7). The results are shown in figure 8. The differences in nodal displacement results between the local-only and distributed test can be attributed to errors produced from the two physical specimens being tested.

The multi-site test was performed with the help of Tommy Marullo at Lehigh University and Richard Christenson at University of Connecticut.

8 Construction of a Hard Real-Time System for Hybrid Testing

Although both *OpenFresco* and *SIMCOR* can perform distributed hybrid tests and in some cases run fast enough to keep the physical specimen in near-constant motion, neither software framework is currently capable of satisfying hard real-time constraints. Previously the *CU-NEES* site has used a heavily customized version of *OpenSEES* running on the ETS/Phar lap real-time operating system (RTOS) to satisfy real-time constraints. As part of the effort to install and test *OpenFresco* , we also proceeded to construct a hardware/software system to replace the old ETS-based hard real-time system.

The construction of the new hard real-time system consists of several steps:

1. Replacing the old real-time computer with newer faster computer

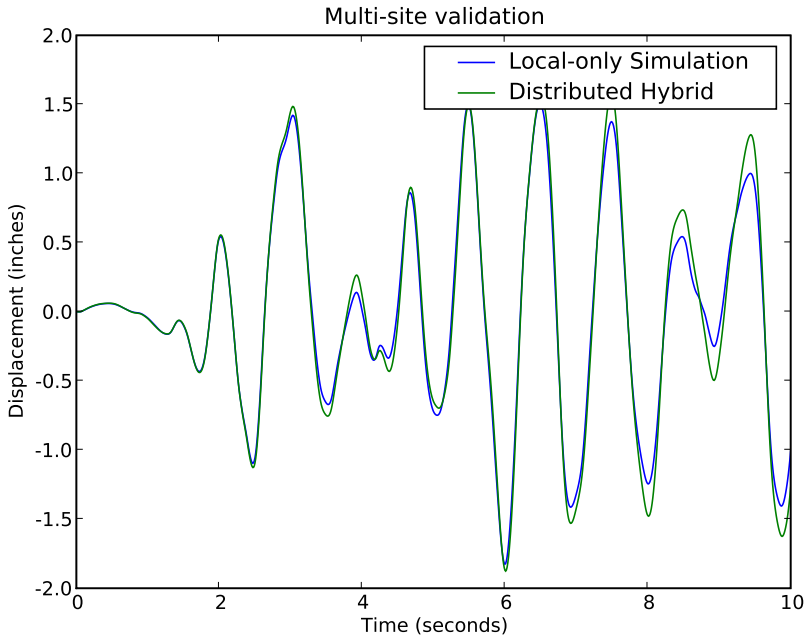

Figure 8: Comparing results from a local-only numerical test and a distributed multi-site test.

2. Replacing the ETS/Phar lap operating system with a modern RTOS
3. Modifying *OpenSEES* / *OpenFresco* to support hard real-time as a special case of the local site configuration available in *OpenFresco* .

The replacement operating system consists of the Linux/GNU operating system, modified to provide real-time response. The default Linux kernel does not provide hard real-time capability, but a set of modifications referred to as the Real-time preemption Patch, or **CONFIG_PREEMPT_RT**, provide a platform for running real-time processes in Linux:

> *The real-time preemption patch set seeks to provide deterministic response times with a stock Linux kernel. It works by making everything preemptable, including code (spinlock-protected critical sections, interrupt handlers) which cannot be preempted in current kernels.*
>
> *The Linux Foundation Weather Forecast*

By replacing the ETS/Phar lap RTOS with a preemptable Linux kernel, we also gain several advantages, including:

- The original two-computer setup required by ETS/Phar lap is replaced by a single computer. The developement system provided for the ETS RTOS required that simulation software be written and built on one computer (the host) then downloaded to a second computer (the target) where the actual software execution takes place. This new solution requires only a single computer which acts as both host and target, takes up less lab space, and removed many of the logistic problems associated with a host/target configuration.

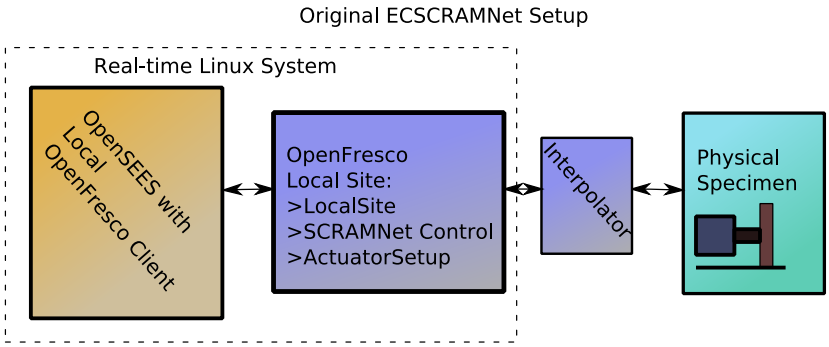

Figure 9: Standard OpenFresco setup on the real-time system, using the interpolator module as an intermediary.

- The ETS operating system maintained some compatibility with Microsoft Windows but not enough to properly support the TCL interpreter that *OpenSEES* uses. To remedy this problem a custom interpreter was written for the old system by *CU-NEES* which imitates TCL closely but not exactly, which made compability with other sites difficult. The new replacement system supports a full TCL interpreter under the Linux/GNU operating system.

- By placing the simulation software on the same platform as the host, certain activities such as debugging and visualization became much easier whereas with the host/target configuration required by ETS these activities are often difficult and impractical.

Other than the preemptable kernel patch, in all the respects the computing system operates as a standard desktop Linux computer, with typical file I/O and networking support and graphics access via X windows. The SCRAMNet network is accessed using a SCRAMNet SC150E card, using the Linux drivers provided by SYSTRAN. Due to the special nature of the real-time Linux kernel, the SYSTRAN drivers had to be modified slightly to correctly receive interrupts from the SCRAMNet network. Specifically, the interrupt processing has to properly receive and process the software interrupts generated by the RT kernel, whereas in a standard kernel the interrupt can be assumed to come directly from the SCRAMnet card.

A further modification is required to reflect differences between the implicit hybrid algorithm used at *CU-NEES* and the standard integrators and hybrid method employed in *OpenFresco* . In particular, *OpenFresco* uses Alpha-Operator Splitting or Explicit Newmark integration methods for the analysis, while using a interpolator and/or predictor/corrector module which mediates data between *OpenFresco* and the actual Physical Specimen (see figure 9).

In contrast, the integration method used at *CU-NEES* (Wei, 2005) incorporates the interpolation and correction into the main analysis cycle. Thus the interpolation module is not required, and a modified ECSCRAMNet control is required to bypass the interpolator and gain direct access to the Physical Specimen (figure 10).

The modifications described above required changes in several C++ source code files, specifically:

- *ECSCRAMNet.h* and *ECSCRAMnet.cpp* were copied and modified to directly access the MTS controller instead of using an intermediary; the modified object was renamed *ECSCRAMNetRaw* to differentiate it from the standard SCRAMNet control.

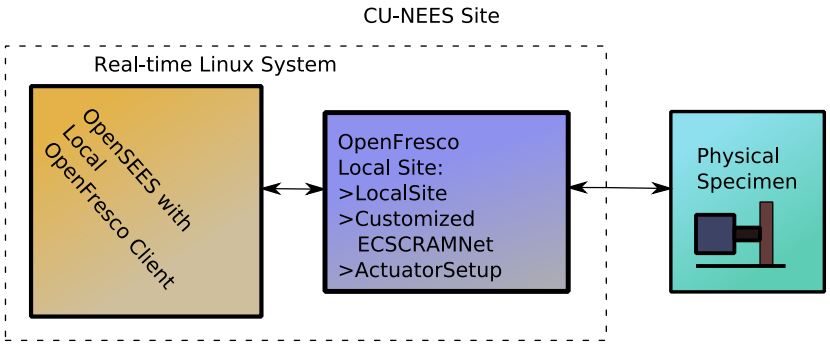

Figure 10: Customized OpenFresco setup on the real-time system, using a custom SCRAMNet control and removing the interpolator module.

- A special solver object *ImplicitShingSolver*, based on the *ModifiedNewton* solver was added which uses a fixed number of iterations and allows other objects to query it for the current iteration.
- Experimental elements require additional code to use the iteration information provided by *ImplicitShingSolver* in order to properly interpolate the target displacements. The elements also need to perform force correction, factoring the displacement error into the restoring force. For this project, two interpolating elements were created: *EEInterpolatingTruss* and *EEInterpolatingTwoNodeLink*.
- The main analysis loop of *OpenSEES* along with the bulk of domain setup (i.e., definition of nodes and elements) remain unchanged.
- Timing and synchronization are performed by a signal handler in the *ECSCRAMNetRaw* control object. In practice, this functionality should be shunted off into a new class designed for general purpose synchronization and timing.

A simple hybrid test was performed using this new realtime *OpenFresco* program, and compared with test results using the old realtime hybrid software that has been used in previous hybrid tests at *CU-NEES* . Unfortunately, the older hybrid simulation software used at *CU-NEES* uses different semantics and configuration parameters in the TCL file to setup and run a hybrid test. Thus, one cannot take a TCL file used in a hybrid test using the old software and simply input it into the new hybrid software; some modificatoins must be made to the input file. In light of this, the test structure used was a simple beam, so that the test configuration could be transferred between old and new hybrid software programs with a minimum of effort. The structure is shown in figure 11. The test results of both old and new software programs are shown in figure 12, along with *OpenSEES* non-hybrid results for context. The results between old and new software programs are nearly identical, with some difference that can be accounted for by measurement noise and rounding differences that occur in the two programs.

9 Summary and Conclusions

The tools *OpenFresco* and *SIMCOR* , both software components used in conducting hybrid tests for earthquake engineering, have been installed and configured at the *CU-NEES* site. Dis-

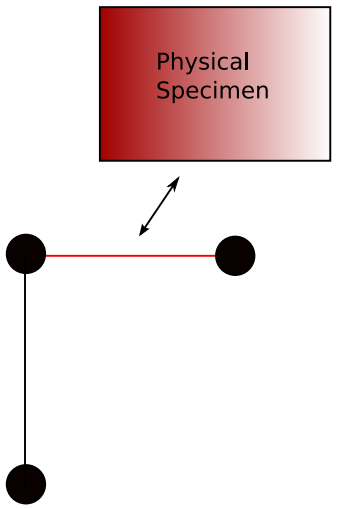

Figure 11: The test structure used for comparing old and new real-time hybrid software.

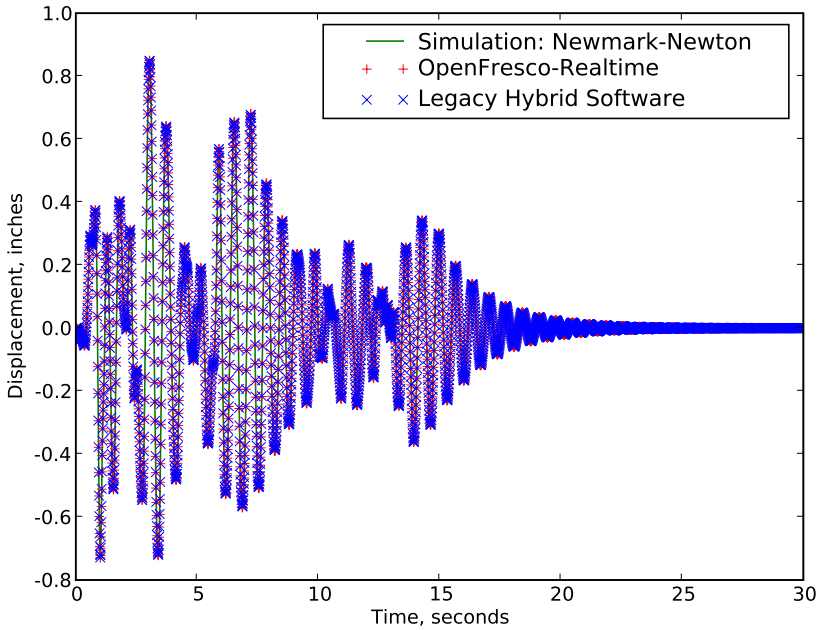

Figure 12: Comparison of test results using both old (Legacy) and new (OpenFresco-Realtime) hybrid test software programs, using numerical/non-hybrid results as a reference.

tributed tests performed jointly with other *NEES* sites verify that the *CU-NEES* can participate in a distributed hybrid test involving either or both software tools.

As part of the validation, a simple hybrid test was performed as both a local test (*CU-NEES* site only) and a distributed test (CU and another, remote, *NEES* site). The results show favorable comparisons between the local and distributed tests, for both *OpenFresco* and *SIMCOR* . In addition, several parts of *OpenSEES* and *OpenFresco* were modified to enable the use of *OpenFresco* in hybrid tests involving hard real-time constraints and using the implicit Shing method for hybrid simulation.

Overall, both *SIMCOR* and *OpenFresco* were able to perform a distributed hybrid test correctly and capably, coupling analysis and measurements into a full hybrid simulation.

Based on the distributed tests using both *SIMCOR* and *OpenFresco* , an urgent need for better diagnostic feedback is needed. There were frequent occurrences where, during an attempt to run a distributed simulation, the simulation simply stopped with no warning or information, due to a bad network connection, protocol mismatch, or sundry other networking-related errors. Both *SIMCOR* and *OpenFresco* would be much more useful if they would recognize network errors and provide information to the software users.

In order to perform hard realtime hybrid testing, *OpenSEES* and *OpenFresco* were extensively modified to run on a realtime operating system and take advantage of the realtime aspects of that operating system. Additional modifications were performed to synchronize with the timing provided MTS controller and to directly access the data parameters of the MTS controller.

In order to provide more general realtime support certain aspects of *OpenFresco* would require extension and enhancements. These include:

- Support for timing of the test, which would allow different experimental controls (local and remote) to synchronize their operations. The current implementation of *OpenFresco* synchronizes the hybrid simulation implicitly by waiting for response data from each control serially. Support for asynchronously waiting on all participating controls would reduce the overall waiting time. Also, the timing implicit in various controls could be made more configurable in order to provide more control over timing and delays within the simulation data flow.

- The interface for experimental elements, sites, and controls currently provide data flow for physical quantities such as displacement, velocity, and force. Information about the simulation time is also important, especially for realtime tests, and the interface should make quantities such as simulation time, timestep, and iteration (if any) available. In the case of *CU-NEES* , the iteration and timestep are needed by the experimental element to properly interpolate the command values for realtime simulation. Even in a non realtime context, certain controls such as the xPC experimental control require a timestep value, which is stored in a configuration file separate from the main simulation configuration.

Finally, both projects could benefit from more documentation elaborating on how they interact with other applications, which would benefit users trying to integrate these projects with their own software. For example, a sequence diagram detailing how *OpenFresco* fits into the standard integrator and analysis process of *OpenSEES* would serve as a useful example when trying to integrate it into a custom finite-element tool.

This Report was made possible with funding support from the National Science Foundation under Cooperative Agreement No. CMMI-0402490 and George E. Brown, Jr. Network for

Earthquake Engineering Simulation Incorporated (*NEESinc*) under the Operations and Maintenance Subaward Agreement OMSA-2006-SSL-UCoB.

References

[1] Gary Haussmann. Evaluation of openfresco and simcor for fast hybrid single site simulation. 2007. CU NEES Number CU-NEES-07-2.

[2] Openfresco: Framework for experimental setup and control. http://neesforge.nees.org/projects/openfresco/.

[3] Simulation coordinator for distributed hybrid simulation and testing. http://neesforge.nees.org/projects/simcor/.

[4] John Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 10 October 1988.

[5] John Stankovic. *Deadline scheduling for real-time systems: EDF and related algoritms.* Klewer Academic Publishers, 1998.

[6] Eric Stauffer. Assessment of opensees for use in realtime and fast hybrid testing. CU NEES Number CU-NEES-07-1, 1 January 2007.

[7] Zhong Wei. *Fast Hybrid Test System for Substructure Evaluation.* PhD thesis, University of Colorado, Boulder, 2005.

A OpenFresco and SIMCOR Initial Validation: OneBayFrame Example Client

```
# File: OneBayFrame_Client1.tcl
# (use with OneBayFrame_Server1a.tcl & OneBayFrame_Server1b.tcl)
#
# $Revision: $
# $Date: $
# $URL: $
#
# Written: Andreas Schellenberg (andreas.schellenberg@gmx.net)
# Created: 11/06
# Revision: A
#
# Purpose: this file contains the tcl input to perform
# a local hybrid simulation of a one bay frame with
# two experimental zero length elements.
# The specimens are simulated using the SimUniaxialMaterials
# controller.


# ------------------------------
```

```
# Start of model generation
# -----------------------------------
# create ModelBuilder (with two-dimensions and 2 DOF/node)
model BasicBuilder -ndm 2 -ndf 2

# Load OpenFresco package
# -----------------------
# (make sure all dlls are in the same folder as openSees.exe)
loadPackage OpenFresco

# Define geometry for model
# -------------------------
set mass3 0.04
set mass4 0.02
# node $tag $xCrd $yCrd $mass
node  1     0.0    0.00
node  2   100.0    0.00
node  3     0.0   54.00  -mass $mass3 $mass3
node  4   100.0   54.00  -mass $mass4 $mass4

# set the boundary conditions
# fix $tag $DX $DY
fix 1   1  1
fix 2   1  1
fix 3   0  1
fix 4   0  1

# Define materials
# ----------------
uniaxialMaterial Elastic 3 [expr 2.0*100.0/1.0]

# Define experimental site
# ------------------------
# expSite RemoteSite $tag <-setup $setupTag> $ipAddr $ipPort <
    $dataSize>
expSite RemoteSite 1 "127.0.0.1" 8090
expSite RemoteSite 2 "128.138.228.117" 80

# Define experimental elements
# ----------------------------
# left and right columns
# expElement twoNodeLink $eleTag $iNode $jNode -dir $dirs -site
    $siteTag -initStif $Kij <-orient $x1 $x2 $x3 $y1 $y2 $y3> <-iMod>
     <-mass $m>
expElement twoNodeLink 1 1 3 -dir 2 -site 1 -initStif 2.8 -orient 0
    1 0 -1 0 0
expElement twoNodeLink 2 2 4 -dir 2 -site 2 -initStif 5.6 -orient 0
    1 0 -1 0 0
```

```
# Define numerical elements
# ------------------------------
# spring
# element truss $eleTag $iNode $jNode $A $matTag
element truss 3 3 4 1.0 3

# Define dynamic loads
# ------------------------------
# set time series to be passed to uniform excitation
set dt 0.02
set scale 0.2
set accelSeries "Path -filePath elcentro.txt -dt $dt -factor [expr
   386.1*$scale]"

# create UniformExcitation load pattern
# pattern UniformExcitation $tag $dir
pattern UniformExcitation  1 1 -accel $accelSeries

# calculate the rayleigh damping factors for nodes & elements
set alphaM      1.010017396536;  # D = alphaM*M
set betaK       0.0;             # D = betaK*Kcurrent
set betaKinit   0.0;             # D = beatKinit*Kinit
set betaKcomm   0.0;             # D = betaKcomm*KlastCommit

# set the rayleigh damping
rayleigh $alphaM $betaK $betaKinit $betaKcomm;
# ------------------------------
# End of model generation
# ------------------------------


# ------------------------------
# Start of analysis generation
# ------------------------------
# create the system of equations
system BandGeneral

# create the DOF numberer
numberer Plain

# create the constraint handler
constraints Plain

# create the convergence test
test EnergyIncr 1.0e-6 10
```

```
# create the integration scheme
integrator NewmarkExplicit 0.5
#integrator AlphaOS 1.0

# create the solution algorithm
algorithm Linear

# create the analysis object
analysis Transient
#-----------------------------
# End of analysis generation
#-----------------------------


#-----------------------------
# Start of recorder generation
#-----------------------------
# create the recorder objects
recorder Node -file Node_Dsp.out -time -node 3 4 -dof 1 disp
recorder Node -file Node_Vel.out -time -node 3 4 -dof 1 vel
recorder Node -file Node_Acc.out -time -node 3 4 -dof 1 accel

recorder Element -file Elmt_Frc.out  -time -ele 1 2 3 forces
recorder Element -file Elmt_tDef.out -time -ele 1 2
   targetDisplacements
recorder Element -file Elmt_mDef.out -time -ele 1 2
   measuredDisplacements
#-----------------------------
# End of recorder generation
#-----------------------------


#-----------------------------
# Finally perform the analysis
#-----------------------------
# perform an eigenvalue analysis
set pi 3.14159265358979
set lambda [eigen 1]
puts "\nEigenvalues at start of transient:"
puts "lambda          omega          period"
foreach lambda $lambda {
   set omega [expr pow($lambda,0.5)]
   set period [expr 2*$pi/pow($lambda,0.5)]
   puts "$lambda   $omega   $period \n"}

# open output file for writing
set outFileID [open elapsedTime.txt w]
# perform the transient analysis
```

```
set tTot [time {
    for {set i 1} {$i < 1600} {incr i} {
        set t [time {analyze  1  $dt}]
        puts $outFileID $t
        }
    }]
puts "Elapsed Time = $tTot \n"
# close the output file
close $outFileID

wipe
# ------------------------------------
# End of analysis
# ------------------------------------
```

B Multi-site Test: Configuration File for Two-Story Structure

```
# File: Test1a_Distr_client.tcl
#
# Written: Cheng Chen
# Created: Sept 17 2007
# Revision: A

# ---------------------------
# Start of model generation
# ---------------------------
# create ModelBuilder (with weo-dimensions and 3 DOF/node)
model BasicBuilder -ndm 2 -ndf 3

# Load OpenFresco package
# -----------------------
# (make sure all dlls are in the same folder as openSees.exe)
loadPackage OpenFresco

# Define geometry for model
# -------------------------
set mass0 1.866
set mass1 5.982
set mass2 4.116

# node $tag $xCrd $yCrd $mass
node  1    0.0    0.0  -mass $mass0 $mass0 0.0
node  2    0.0  144.0  -mass $mass1 $mass1 0.0
node  3    0.0  288.0  -mass $mass2 $mass2 0.0
node  4  360.0    0.0  -mass $mass0 $mass0 0.0
node  5  360.0  144.0  -mass $mass1 $mass1 0.0
```

```
node  6  360.0  288.0  -mass $mass2 $mass2 0.0

# set the boundary conditions
# fix $tag $DX $DY $RZ
fix  1  1  1  1
fix  2  0  1  1
fix  3  0  1  1
fix  4  1  1  1
fix  5  0  1  1
fix  6  0  1  1

# Define materials
# -------------------
# uniaxialMaterial Steel02 $matTag $Fy $E $b $R0 $cR1 $cR2 $a1 $a2
   $a3 $a4
uniaxialMaterial Elastic 1 505

# Define experimental site
# ------------------------
# expSite RemoteSite $tag <-setup $setupTag> $ipAddr $ipPort <
   $dataSize>
expSite RemoteSite 1 "128.138.228.117" 990
expSite RemoteSite 2 "128.180.53.3" 8092

# geometric transformation
# geoTransf type $tag
geomTransf Linear 10

# Define experimental elements
# ----------------------------
# left and right columns

# Define numerical elements
# -------------------------

# Define element

# expElement $eleTag $iNode $jNode $transTag -site $siteTag
   -initStif $Kij <-iMod> <-rho $rho>
element elasticBeamColumn 1 1 2 91.4 29000 4330 10

# element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iz
   $transTag
expElement beamColumn 2 2 3 10 -site 1 -initStif 18407 0 0 0 505
   -36334 0 -36334 3488056

# expElement $eleTag $iNode $jNode $transTag -site $siteTag
   -initStif $Kij <-iMod> <-rho $rho>
```

```
expElement beamColumn 3 4 5 10 -site 2 -initStif 18407 0 0 0 505
    -36334 0 -36334 3488056

element elasticBeamColumn 4 5 6 91.4 29000 4330 10
element elasticBeamColumn 5 2 5 44.2 29000 9040 10
element elasticBeamColumn 6 3 6 44.2 29000 9040 10


# Define dynamic loads
# ------------------------
# set time series to be passed to uniform excitation
set dt 0.01
set scale 1.0
set accelSeries "Path -filePath ELC270.txt -dt $dt -factor [expr 386
    .1*$scale]"

# Get Initial Stiffness
# ------------------------
#initialize

# create UniformExcitation load pattern
# pattern UniformExcitation $tag $dir
pattern UniformExcitation 11 1 -accel $accelSeries

# Define damping
# rayleigh $alphaM $betaK $betaKinit $betaKcomm
rayleigh  0.18289   0   0.0017984   0

# ------------------------
# End of model generation
# ------------------------

# ------------------------------
# Start of recorder generation
# ------------------------------
# create the recorder objects
recorder Node -file Node_Dsp.out -time -node 1 2 3 4 5 6 -dof 1 disp
recorder Node -file Node_Vel.out -time -node 1 2 3 4 5 6 -dof 1 vel
recorder Node -file Node_Acc.out -time -node 1 2 3 4 5 6 -dof 1
    accel

recorder Element -file Elmt_Frc.out -time -ele 1 2 3 4 5 6 force
recorder Element -file Elmt_Def.out -time -ele 1 2 3 4 5 6
    deformation
# ------------------------
# End of recorder generation
# ------------------------
```

```
#--------------------------------
# Start of analysis generation
#--------------------------------
# create the system of equations
system BandGeneral

# create the DOF numberer
numberer Plain
# create the constraint handler
constraints Plain
# create the convergence test
test EnergyIncr 1.0e-6 10

# create the integration scheme
integrator NewmarkExplicit 0.5

# create the integration algorithm
algorithm Linear

# create the analysis object
analysis Transient
#--------------------------------
# End of analysis generation
#--------------------------------

#--------------------------------
# Finally perform the analysis
#--------------------------------
set pi 3.14159265358979
set lambda [eigen 2]
puts "\nEigenvalues at start of transient:"
puts "   lambda\t   omega\t   period"
foreach lambda $lambda {
        set omega [expr pow($lambda,0.5)]
        set period [expr 2*$pi/pow($lambda,0.5)]
        puts "$lambda  $omega  $period"
}

# open output file for writing
set outFileID [open elapseTime.txt w]
# perform the transient analysis
set tTot [time {
        for {set i 1} {$i<4000} {incr i} {
                set t [time {analyze 1 $dt}]
                puts $outFileID $t
                puts "step number $i"
        }
}]
```

```
puts "Elapsed Time = $tTot \n"

# close the output file
close $outFileID

wipe
# ----------------
# End of analysis
# ----------------
```

C Realtime Validation: Configuration file for old realtime software

```
#
# Test example using a single beam, for testing/validation purposes.
#

model BasicBuilder -ndf 3 -ndm 2

#loadPackage OpenFresco

# define a few nodes.  We need only three, two for the cantilever
   and a third for the
# "reaction wall"

#         node #      x         y
node      1           0.0       0.0
node      2           0.0       156.0
node      3           50.0      156.0


# nodal mass
#         node #      xMass     yMass    RotMass
mass      2           1.0       1.0      0.0001

# boundary conditions
# type    node #      constrain:  x?   y?   rot?
fix       1                       1    1    1
fix       3                       1    1    1

# specify steel material properties
uniaxialMaterial Steel01 1 50 29000 1e-6

uniaxialMaterial Elastic 2 5.6
```

```
geomTransf Linear 1



# cantilever
#                                ele# NodeI NodeJ   A       E       I
    Transform
element elasticBeamColumn    1     1      2        75.6    29000   3400
    1

#
# hybrid elements
#

element sNodeElement 11 2 1 1 1 1 1 1  5.6 0 0 0 5.6 0 0 0 5.6   1 0
    0 0 1 0 0 0 1     1 0 0 0 1 0 0 0 1

pattern UniformExcitation 1 1 -accel "Path -filePath elcentro.txt
    -dt 0.01 -factor 386.4"

# calculate the rayleigh damping factors for nodes & elements
set alphaM      1.010017396536;  # D = alphaM*M
set betaK       0.0;             # D = betaK*Kcurrent
set betaKinit   0.0;             # D = beatKinit*Kinit
set betaKcomm   0.0;             # D = betaKcomm*KlastCommit

# set the rayleigh damping
rayleigh $alphaM $betaK $betaKinit $betaKcomm;

# ------------------------------
integrator FHT 0.5 0.25 0.065722 0.0 0.0 0.00062806

recorder Node -file SingleBeam_disp_HybridTest_old.out -time -node 2
    -dof 1 2 3 disp
#recorder Element -file Target_disp_HybridTest.out -time -ele 2
    targetDisplacements

test EnergyIncr 1.0e-20 20 3

#algorithm Newton
#algorithm ImplicitShingSolver -count 10
algorithm FHT

numberer Plain
system BandGeneral

#analysis Transient
analysis FHT
```

```
analyze 3000 0.01
```

D Realtime Validation: Configuration file for new realtime software

```
#
# Test example using a single beam, for testing/validation purposes.
#

model BasicBuilder -ndf 3 -ndm 2

loadPackage OpenFresco

# define a few nodes.  We need only three, two for the cantilever
   and a third for the
# "reaction wall"

#         node #      x         y
node      1           0.0       0.0
node      2           0.0       156.0
node      3           50.0      156.0


# nodal mass
#         node #      xMass     yMass    RotMass
mass      2           1.0       1.0      0.0001

# boundary conditions
# type    node #      constrain:  x?    y?    rot?
fix       1                       1     1     1
fix       3                       1     1     1

# specify steel material properties
uniaxialMaterial Steel01 1 50 29000 1e-6

uniaxialMaterial Elastic 2 5.6


geomTransf Linear 1



# cantilever
#                           ele# NodeI NodeJ   A       E       I
   Transform
```

```
element elasticBeamColumn    1     1      2         75.6    29000   3400
    1

#
# hybrid elements
#

#
# OpenFresco elements
expControl SCRAMNetRaw 1 8 3
#expControl SimUniaxialMaterials 1 2
expSetup OneActuator 1 -control 1 1
expSite LocalSite 1 1
# expElement twoNodeLink $eleTag $iNode $jNode -dir $dirs -site
    $siteTag -initStif $Kij <-orient $x1 $x2 $x3 $y1 $y2 $y3> <-iMod>
     <-mass $m>
expElement interptwoNodeLink 2 2 3 -dir 2 -site 1 -initStif 5.6
    -orient 0 1 0 -1 0 0

#element sNodeElement 11 5 1 1 1 1 1 1 131.19 0 4135.83 0 967.65 0
    4135.83 0 204128.4 1 0 0 0 -0.5 -0.5 0 0.01389 -0.01389 1 0 0 0
    -1 36 0 -1 -36

pattern UniformExcitation 1 1 -accel "Path -filePath elcentro.txt
    -dt 0.01 -factor 386.4"

# calculate the rayleigh damping factors for nodes & elements
set alphaM      1.010017396536;  # D = alphaM*M
set betaK       0.0;             # D = betaK*Kcurrent
set betaKinit   0.0;             # D = beatKinit*Kinit
set betaKcomm   0.0;             # D = betaKcomm*KlastCommit

# set the rayleigh damping
rayleigh $alphaM $betaK $betaKinit $betaKcomm;

# ----------------------------------
integrator Newmark 0.5 0.25
#integrator HHT 0.5
#integrator FHT 0.5 0.25 0.65722 0.0 0.0 0.00062806

recorder Node -file SingleBeam_disp_HybridTest.out -time -node 2
    -dof 1 2 3 disp
recorder Element -file Target_disp_HybridTest.out -time -ele 2
    targetDisplacements

test EnergyIncr 1.0e-20 20 3

#algorithm Newton
```

```
algorithm ImplicitShingSolver -count 10
#algorithm FHT

numberer Plain
system BandGeneral

analysis Transient
#analysis FHT

analyze 3000 0.01
```