Software Framework for Parameter Updating and Finite-Element Response Sensitivity Analysis

Michael H. Scott, A.M.ASCE[1]; and Terje Haukaas[2]

Abstract: The finite-element software framework OpenSees is extended with parameter updating and response sensitivity capabilities to support client applications such as reliability, optimization, and system identification. Using software design patterns, member properties, applied loadings, and nodal coordinates can be identified and repeatedly updated in order to create customized finite-element model updating applications. Parameters are identified using a Chain of Responsibility software pattern, where objects in the finite-element model forward a parameterization request to component objects until the request is handled. All messages to identify and update parameters are passed through a Facade that decouples client applications from the finite-element domain of OpenSees. To support response sensitivity analysis, the Strategy design pattern facilitates multiple approaches to evaluate gradients of the structural response, whereas the Visitor pattern ensures that objects in the finite-element domain make the proper contributions to the equations that govern the response sensitivity. Examples demonstrate the software design and the steps taken by representative finite-element model updating and response sensitivity applications.

DOI: 10.1061/(ASCE)0887-3801(2008)22:5(281)

CE Database subject headings: Computer programming; Computer software; Finite element method; Nonlinear analysis; Optimization; Parameters; Sensitivity analysis; Structural reliability.

Introduction

Emergent structural assessment methodologies have placed an increased emphasis on applications that update the material, load, and geometric parameters of a finite-element model in order to characterize uncertain structural response. For example, a structural reliability analysis repeatedly updates realizations of the uncertain model parameters in order to find the most probable failure point (Liu and Der Kiureghian 1991). In damage detection and system identification applications, model parameters evolve as a function of observed system characteristics (Soh and Dong 2001; Ozcelik et al. 2008).

Model updating applications can use any finite-element code as a "black box" to evaluate a performance function by repeatedly creating an input file for each realization of the parameters, running the finite-element analysis, and parsing the output file for the desired response quantities. Gradients of the performance function are obtained by finite differences, where the finite-element analysis is repeated with perturbed parameter values. For most sensitivity-based model updating applications, the preferred approach to obtain the response sensitivities is the direct differentiation method [(DDM), Kleiber et al. 1997], where the governing equations are differentiated analytically and implemented as part of the core finite-element code. This approach is significantly more efficient and accurate than finite difference schemes, at the one-time cost of deriving and implementing the analytical derivatives of the response. Zhang and Der Kiureghian (1993) derived the DDM response sensitivity equations for both static and dynamic structural analysis problems, as well as for the J_2 plasticity constitutive model (Simo and Hughes 1998). Since then, a number of contributions have been made to the development of DDM equations for various finite-element formulations and constitutive models (Roth and Grigoriu 2001; Conte et al. 2003; Franchin 2004; Scott et al. 2004; Haukaas and Der Kiureghian 2005).

To incorporate DDM equations in a nonlinear finite-element analysis, Zhang and Der Kiureghian (1997) extended the procedural code FEAP (Taylor 2004) with DDM capabilities, whereas Roth and Grigoriu (2001) did the same for the DIANA software (TNO Building and Construction Research 2003). Silva and Bittencourt (2000) developed an object-oriented code for shape optimization of linear-elastic finite-element models and Gil and Bugeda (2001) implemented finite-element sensitivity analyses for nonlinear material models using an object-oriented approach. However, the development of DDM response sensitivity modules in OpenSees represents one of the first attempts to characterize all major sources of uncertainty in a nonlinear finite-element analysis using an object-oriented approach (Haukaas and Der Kiureghian 2007).

Identifying the parameters that define a structural model is a key component to extending finite-element software with model updating and DDM response sensitivity capabilities. For procedural implementations, parameter updating is efficient as the parameters are typically stored in global data structures or common blocks. However, this approach can lead to unintended results when shared global data are modified and it requires a thorough

[1]Assistant Professor, School of Civil and Construction Engineering, Oregon State Univ., Corvallis, OR 97331 (corresponding author). E-mail: michael.scott@oregonstate.edu

[2]Assistant Professor, Dept. of Civil and Environmental Engineering, Univ. of British Columbia, Vancouver BC, Canada V6T 1Z4. E-mail: terje@civil.ubc.ca

Note. Discussion open until February 1, 2009. Separate discussions must be submitted for individual papers. The manuscript for this paper was submitted for review and possible publication on May 8, 2007; approved on February 5, 2008. This paper is part of the ***Journal of Computing in Civil Engineering***, Vol. 22, No. 5, September 1, 2008. ©ASCE, ISSN 0887-3801/2008/5-281–291/$25.00.

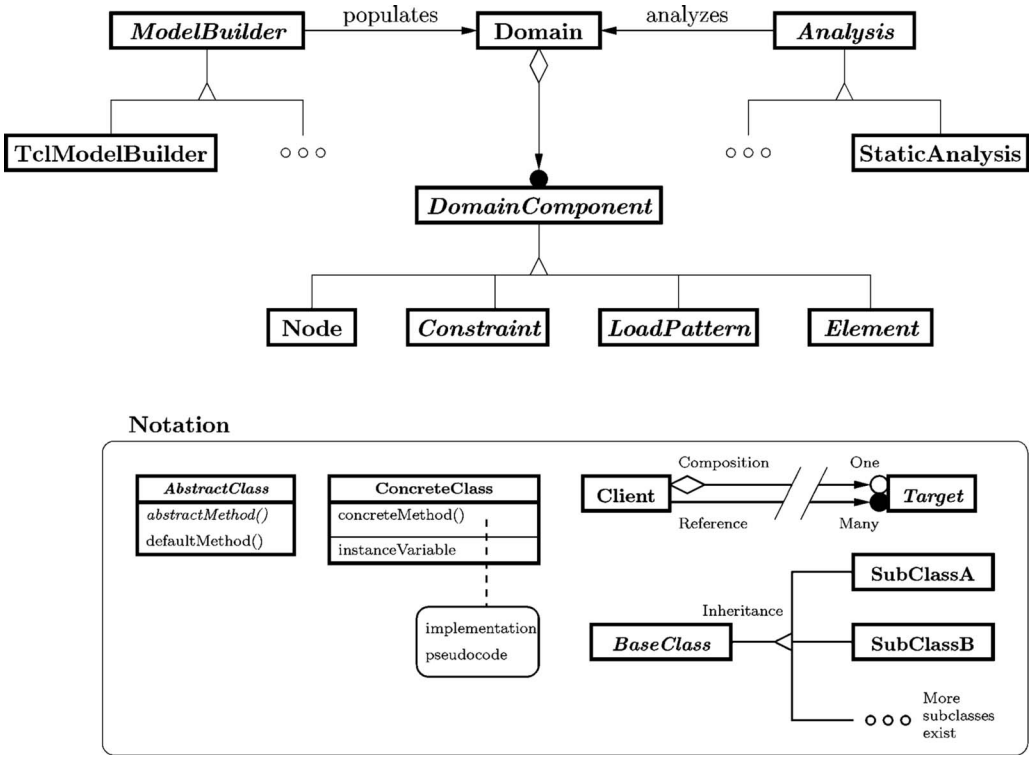

Fig. 1. High-level classes of the OpenSees software framework and notation used for class diagrams

understanding of how the common data structures are implemented. In an object-oriented approach, objects encapsulate parameters such that clients do not need to know all implementation details. Typically, a large number of objects are created during a finite-element analysis, thus extensive logic may be required in order to determine which object encapsulates a desired parameter.

Data encapsulation is one of the primary benefits to taking an object-oriented approach in engineering software development (Fenves 1990; Lee and Arora 1991; Baugh and Rehak 1992). Several object-oriented designs for the finite-element analysis of structural systems have been developed (Forde et al. 1990; Miller 1991; Mackie 1992; Zimmermann et al. 1992; Rucki and Miller 1996; Archer et al. 1999; McKenna and Fenves 2000; Modak and Sotelino 2002; Mackerle 2004). These and other designs have focused on object-oriented approaches to forming and solving the equations that govern a finite-element analysis; however, object-oriented approaches to update model parameters and to compute DDM response sensitivity have not been addressed to the same extent in the literature.

This paper describes an object-oriented framework that has been implemented in OpenSees to support finite-element model updating and response sensitivity analysis. Software design patterns ensure the framework is extensible such that it can accommodate a wide range of model updating applications and that objects in a finite-element model make the correct contributions to the equations that govern DDM response sensitivity. Although the implementation platform is OpenSees, the design patterns and class relationships presented in this paper are applicable to any object-oriented finite-element analysis framework. The paper begins with an overview of the OpenSees software framework. A "top-down" approach to identify and update parameters using software design patterns is presented next, followed by the extension of OpenSees to incorporate DDM response sensitivity analysis. Representative examples demonstrate the parameter updating and response sensitivity capabilities along with their extension to a sensitivity-based damage detection analysis of a reinforced concrete shear wall structure.

Overview of the OpenSees Software Framework

OpenSees, the Open System for Earthquake Engineering Simulation, is an object-oriented software framework developed for the computational simulation of structural and geotechnical systems (McKenna et al. 2000). Most modules in the OpenSees framework are implemented in C++ using an open-source development process. A variety of state-of-the-art finite-element formulations and numerical methods have been implemented. Most users of OpenSees build finite-element models using the fully programmable string-based scripting language Tcl (Ousterhout 1994; Welch 2000), which offers more flexibility in pre- and postprocessing analysis results than fixed-format text files. As the computational platform for NEES, the Network for Earthquake Engineering Simulation, OpenSees has been extended with modules for hybrid simulation (Schellenberg and Mahin 2006). The framework also supports distributed and parallel computing (McKenna and Fenves 2000; Peng and Law 2004).

The high level classes of OpenSees are loosely coupled in order to support a wide range of applications. As shown in Fig. 1, a Domain aggregates DomainComponent objects that represent the nodes, elements, constraints, and load patterns of a finite-element model (McKenna et al. 1997). An instance of the ModelBuilder class populates the Domain based on user input and an Analysis object advances the Domain to a new state based on the type of analysis and the loads applied to the system.

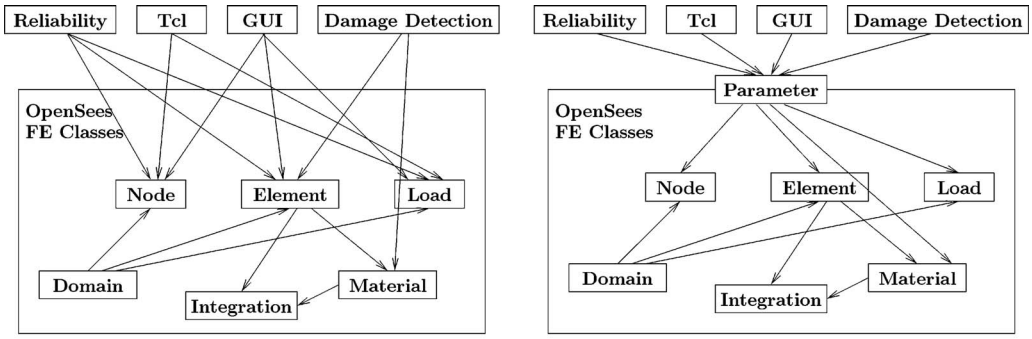

Fig. 2. Client applications and OpenSees finite-element analysis classes: (a) direct access to analysis classes; (b) access to analysis classes controlled by access to Parameter class

Software Design Patterns

Specific implementations of the classes shown in Fig. 1 derive their behavior by composition of other classes in the OpenSees framework. For example, most implementations of the Element class use instances of the Material class as interchangeable algorithms to determine the constitutive response of a finite element. This is a common relationship, or pattern, that has appeared in virtually all object-oriented software designs for finite-element analysis. Gamma et al. (1995) catalog this "Strategy" pattern along with many other generic descriptions of communicating objects that can make a software system more flexible and maintainable. The axiom of design patterns is to favor object composition over class inheritance as a mechanism for code reuse. OpenSees makes extensive use of software design patterns to solve the governing equations of a nonlinear structural analysis (McKenna 1997) and to implement a wide range of finite-element models (Scott et al. 2008). Other applications of design patterns in civil engineering include the development of computational tools for life-cycle assessment of buildings (Ries and Mahdavi 2001) and knowledge systems to promote sustainable construction (Wetherill et al. 2007).

Although the use of design patterns has made OpenSees more modular and flexible, it has resulted in a large number of small classes related by complex inheritance hierarchies and multiple levels of indirection. As a result, there may be several levels of indirection to follow in order to update parameters and to assemble DDM response sensitivity equations for a finite-element model built in OpenSees. This makes it difficult for a client application to use OpenSees as the computational engine for sensitivity-based finite-element model updating applications. The software framework for parameterization presented herein mitigates this difficulty by using the following design patterns:

- "Facade" to provide an entry point for all model updating client applications to access parameters of the finite-element model.
- "Chain of responsibility" to identify the objects that encapsulate model parameters by forwarding identification requests on to component objects.
- "Strategy" to use interchangeable algorithms in the computation of structural response sensitivity.
- "Visitor" to ensure element and material objects compute the correct contribution to DDM response sensitivity equations for each parameter in the finite-element model.

Full details of each above-listed design pattern are given by Gamma et al. (1995) and their contributions to the development of the parameterization framework for sensitivity-based model updating applications are described in the following sections.

Parameterization of the OpenSees Software Framework

There are two fundamental approaches to parameterizing a software framework for finite-element analysis. In the "top-down" approach, objects encapsulate the parameters that define their behavior and new parameter values are pushed down through the framework only when instructed to do so by a model updating application. This is as opposed to the "bottom-up" approach, where objects encapsulate pointers to auxiliary objects that a model updating application updates when necessary. A significant drawback to the bottom-up approach is that it requires objects to check for changes to parameter values at every state determination during an analysis. This overhead can be mitigated by violating encapsulation, e.g., with "friend" classes in C++; however, the following presentation is based on a top-down approach as it preserves encapsulation.

Parameter Class as a Facade

Direct access to the classes of OpenSees that encapsulate model parameters can lead to the situation shown in Fig. 2(a) where there are multiple pointers from client applications in to the finite-element domain. Client application must understand the interface of every class in the finite-element subsystem that encapsulates model parameters. This tight coupling can make it difficult to adapt client applications to changes in the finite-element classes, e.g., when existing classes are split into multiple classes due to the application of design patterns. The Facade design pattern (Gamma et al. 1995) provides client applications a default view of the finite-element domain and alleviates the need to understand every class relationship therein. As shown in Fig. 2(b), the Parameter class acts as a facade, or entry point, for client applications to access the finite-element domain of OpenSees.

The Parameter class is shown in Fig. 3 along with the methods that a client application uses to identify and update parameters in the finite-element domain. Each instance of the Parameter class maintains a list of integer keys and pointers to ParameterizedObject, which is a base class for all objects in the finite-element domain that encapsulate model parameters. A client application constructs a Parameter using a list of DomainComponents and a string that indicates which parameter is to be identified. Each

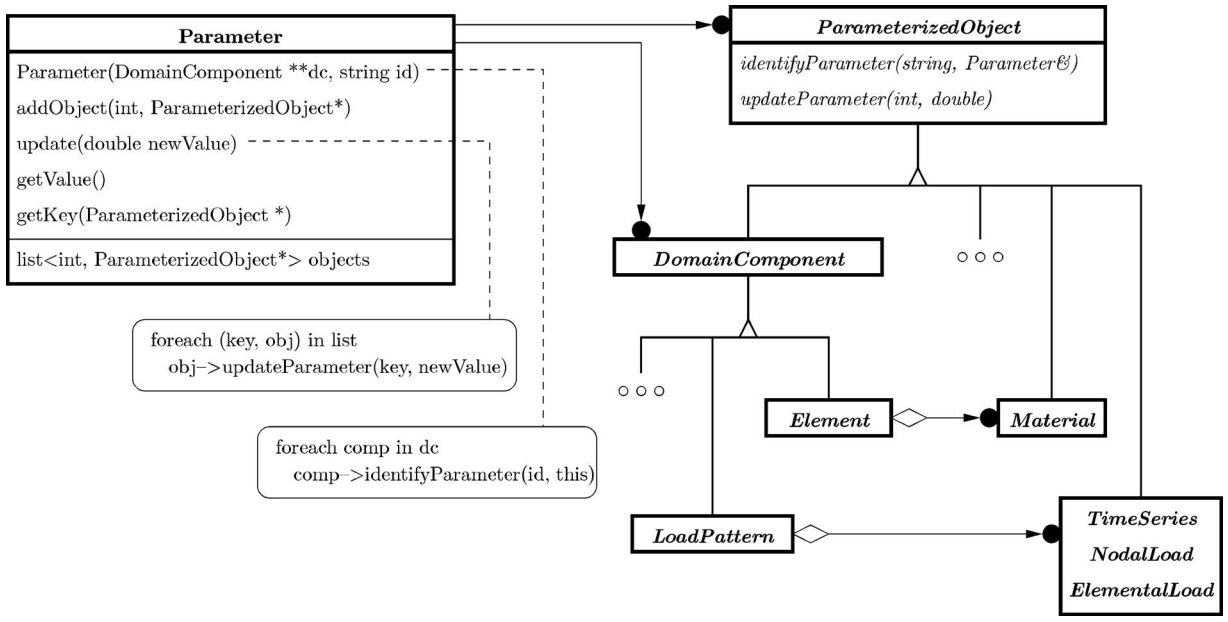

Fig. 3. Class diagram showing the interaction of the Parameter class with the DomainComponent and ParameterizedObject classes of OpenSees

implementation of ParameterizedObject is responsible for associating this string with the integer key stored by a Parameter object. This association is established by implementations of the identifyParameter( ) method, which is described in the following section. When a client application invokes the update( ) method on a Parameter object, it propagates the new parameter value to all instances of the ParameterizedObject class in its list by invoking the updateParameter( ) method, as shown in the implementation pseudocode of Fig. 3.

Identification of Parameters Using a Chain of Responsibility

Each instance of the ParameterizedObject class is responsible for determining if the string passed to the identifyParameter( ) method represents one of its encapsulated parameters. If this is the case, the ParameterizedObject invokes a callback method on the Parameter using an integer key and a pointer to itself ("this" in C++). If the ParameterizedObject cannot make the association, it either forwards the parameterization request on to its component objects or it terminates the search. This approach, based on the Chain of Responsibility design pattern (Gamma et al. 1995), gives more than one object in the finite-element domain a chance to identify a parameter as a client application does not know a priori which class encapsulates the desired data.

The implementation of the identifyParameter( ) and updateParameter( ) methods for a material nonlinear truss finite element is shown in Fig. 4. The InelasticTruss class employs the Strategy pattern to obtain its constitutive response from one of many implementations of the Material class. The InelasticTruss class encapsulates the cross-sectional area, A, of the element, whereas the parameters that define the constitutive response are encapsulated by, e.g., the BilinearMaterial class. When an instance of the InelasticTruss class receives a request to identify a parameter, it first checks if the character string corresponds to the cross-sectional area. If so, the InelasticTruss invokes the callback function, Parameter::addObject( ), with an integer key and a pointer to itself; otherwise, it passes the request on to the BilinearMaterial class, where the process is repeated for its encapsulated parameters. The Parameter stores these (key, pointer) pairs in order to invoke the updateParameter( ) method on the appropriate object when it receives a request to do so from a client application.

Commentary on the Parameterization Approach

The advantages of this approach to identifying and updating parameters are:

- The identification of parameters follows existing links in the framework. New links do not need to be established in order to parameterize a finite-element model.
- Each ParameterizedObject encapsulates the association between integer keys and parameters, thus keys can be duplicated horizontally (through composition relationships) and vertically (through inheritance relationships) in OpenSees. The Parameter class only needs to store, not interpret, the key values.
- Updates are efficient as the Parameter class stores a pointer to each ParameterizedObject. The links in the Chain of Responsibility do not need to be followed in order to update a parameter.

Despite these advantages, the following drawbacks are noted for this approach to the parameterization of a finite-element model:

- Reuse of existing code in other contexts can lead to nonidentification of parameters due to mismatch of identifying strings with encapsulated data. For example, when the BilinearMaterial class is used for moment-curvature response of a beam cross section rather than stress–strain response of a truss element, the string "sigmaY" becomes a nonintuitive way to identify the section yield moment as a parameter. Such problems can be overcome by modifying implementations of the identifyParameter( ) method to associate multiple strings with encapsulated data.
- Finite-element models must be defined in "normal form" with a minimum number of parameters in order to avoid

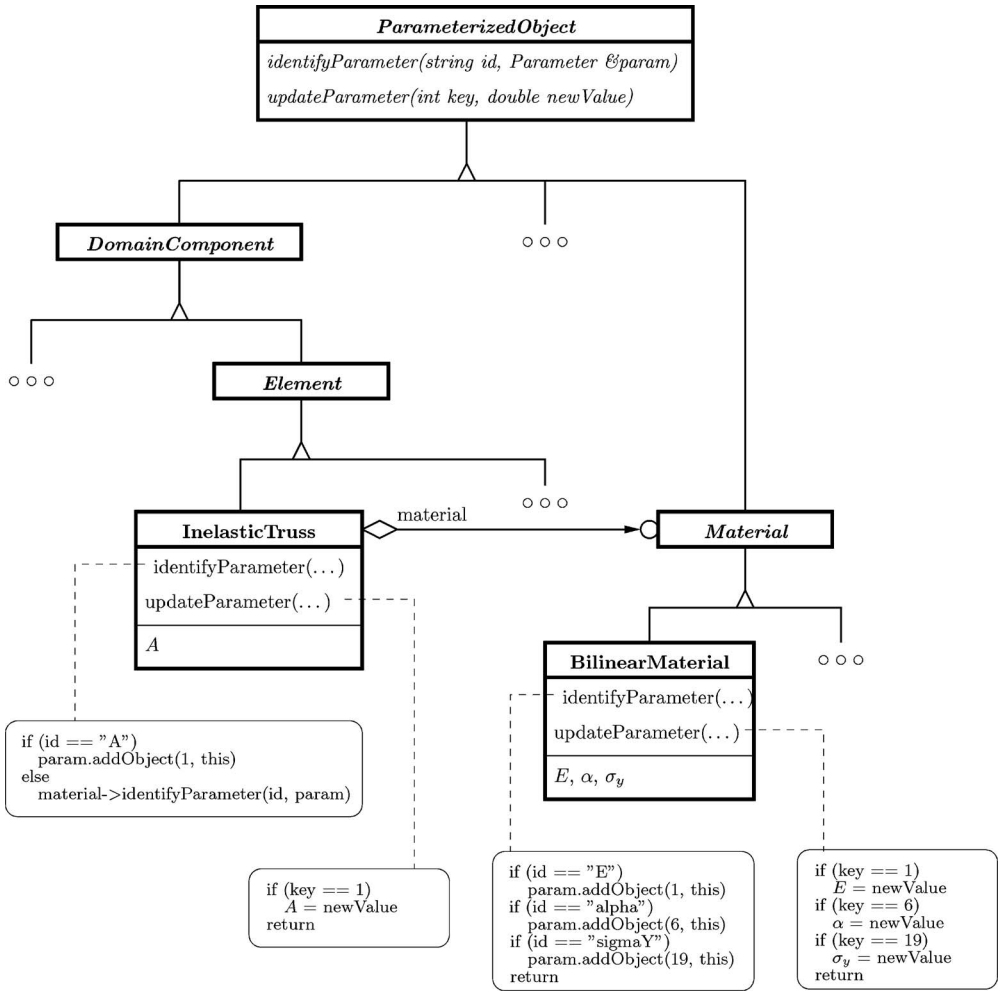

Fig. 4. Class diagram for the InelasticTruss element showing methods to identify and update parameters

data inconsistency of derived parameters. For example, computing and storing the hardening stiffness, $E_h = \alpha E$, in the constructor of BilinearMaterial can lead to an inconsistent state determination if E_h is not recalculated after a call to updateParameter( ) modifies either E or α.

Example Parameter Updating Application

A Monte Carlo analysis of a nonlinear truss structure demonstrates the parameter updating functionality. The truss shown in Fig. 5 is defined using the `node`, `element`, `uniaxialMaterial`, and other commands added to the Tcl interpreter for building models in OpenSees (Mazzoni et al. 2006). These commands are composed into a Tcl script, trussModel.tcl.

To demonstrate the framework for parameterization and model updating, the `parameter` and `updateParameter` commands are also added to the Tcl interpreter. These commands invoke the identifyParameter( ) and updateParameter( ) methods on objects defined for the truss analysis. The sequence of Tcl commands shown in Fig. 6 acts as a client application that performs a Monte Carlo analysis of the truss using three normally distributed random variables. The first parameter is associated with the cross-sectional area of Element 1, the second parameter maps to the yield stress of Elements 1 and 3, and the third represents the horizontal load applied at Node 4. A diagram of the relevant objects created at run time is shown in Fig. 7. The parameters are updated 100 times with a load-controlled static pushover analysis performed in ten load steps for each realization of the uncertain parameters. The results of the Monte Carlo analysis are shown in Fig. 6.

Response Sensitivity Computations in OpenSees

In addition to stand-alone finite-element model updating applications, the framework is designed to support sensitivity-based model updating applications, where the gradient of the finite-element response must be computed in order to find an optimal solution. There are two approaches to evaluate the gradient of the structural response: the finite difference method (FDM) and the aforementioned DDM. Although the DDM is more efficient and accurate than the FDM, it can be difficult to implement for finite-element models with complex state determination algorithms. As a result, some finite-element analysis software packages use a mixture of the two methods, whereby the "high-level" governing equations are differentiated directly whereas finite differences are employed on the "low-level" constitutive equations (ABAQUS, Inc. 2006). The parameterization framework in OpenSees accommodates all three approaches by using the appropriate software

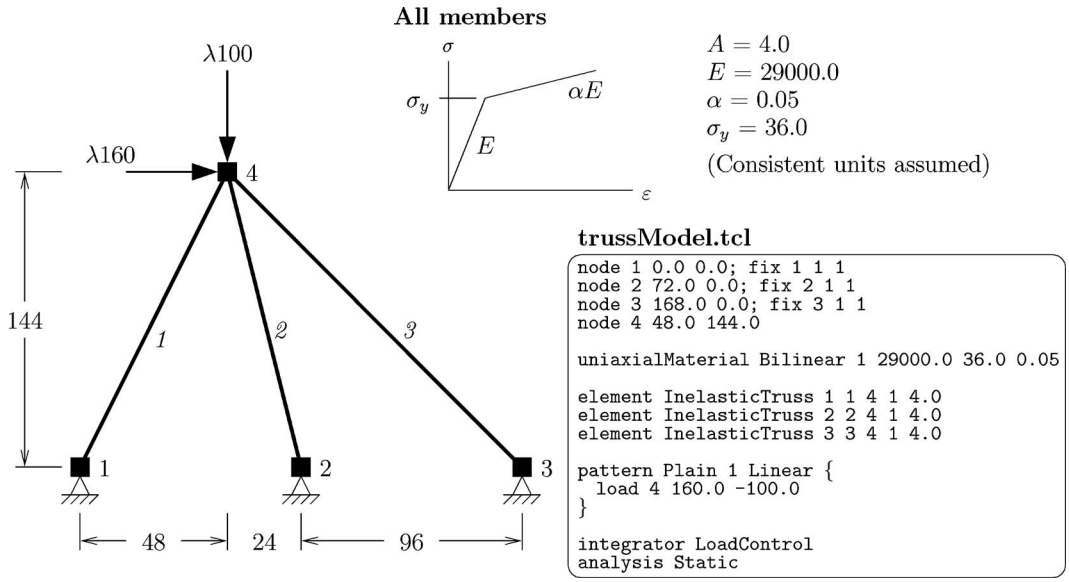

Fig. 5. Three member truss with Tcl script that defines the member properties, applied loading, and analysis type

```
# Define FE model
source trussModel.tcl

# Identify parameters
parameter 1 element 1 A
parameter 2 eleList 1 3 sigmaY
parameter 3 pattern 1 nodalLoad 4 1

# Run Monte-Carlo analysis
for {set i 1} {$i <= 100} {incr i} {
  # Update parameters using normal distribution
  updateParameter 1 [randNormal   4.0  0.5]
  updateParameter 2 [randNormal  36.0  2.0]
  updateParameter 3 [randNormal 160.0 16.0]

  analyze 10
  reset
}
```

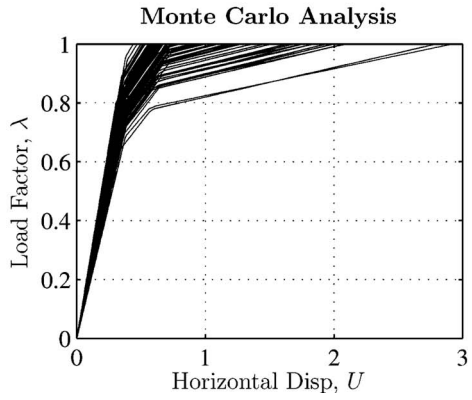

Fig. 6. Monte Carlo analysis of truss load-displacement response for 100 sets of parameter realizations derived from random values of normally distributed cross-sectional area of Member 1, yield stress of Members 1 and 3, and horizontal load at Node 4

design patterns. The extension of the finite-element analysis modules of OpenSees for sensitivity computations is outlined for static problems, noting that the extension to dynamic problems is straightforward.

For the case of static equilibrium where there is balance between the applied nodal load vector, $\mathbf{P}_f$, and the vector of static resisting forces, $\mathbf{P}_r$, the sensitivity of the nodal displacement vector $\mathbf{U}$ with respect to a single model parameter θ is obtained by solving the linear system of equations

$$\frac{\partial \mathbf{U}}{\partial \theta} = \mathbf{K}^{-1}\left(\frac{\partial \mathbf{P}_f}{\partial \theta} - \left.\frac{\partial \mathbf{P}_r}{\partial \theta}\right|_{\mathbf{U}}\right) \tag{1}$$

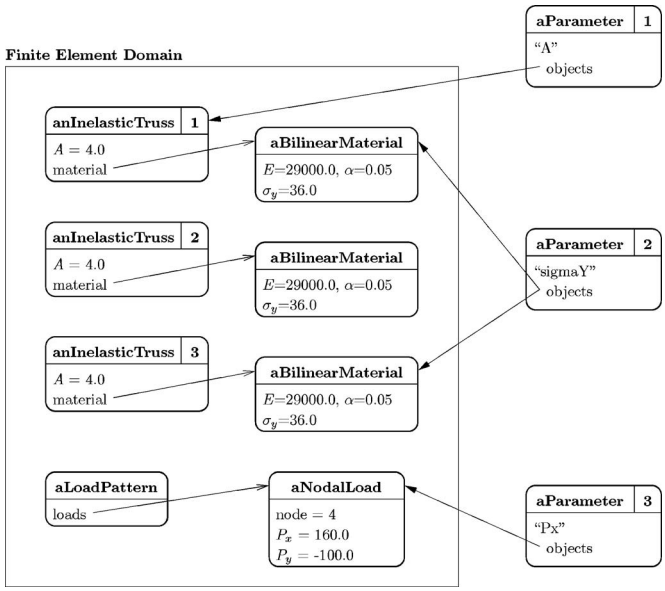

Fig. 7. Object diagram of three member truss model showing the links between Parameter objects and FE objects established at run time

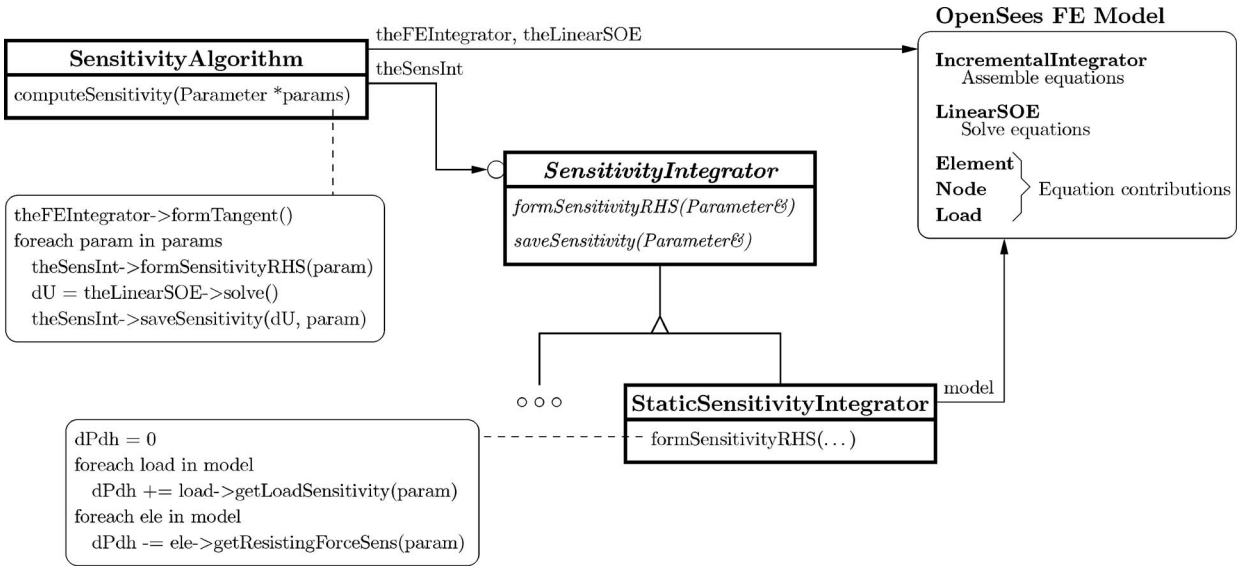

Fig. 8. Class diagram showing the use of the SensitivityIntegrator class as a Strategy by the SensitivityAlgorithm

where $\mathbf{K}$=tangent stiffness matrix of the structure at the converged equilibrium state; $\partial \mathbf{P}_f / \partial \theta$=derivative of the applied load vector; and $\partial \mathbf{P}_r / \partial \theta|_{\mathbf{U}}$=derivative of the resisting force vector under the condition of fixed nodal displacements. The derivation of Eq. (1) and its extension to the case of dynamic equilibrium are found in the aforementioned references, and those cited therein, on the DDM.

The efficiency of the DDM approach lies in the form of Eq. (1), where the factorization of the tangent stiffness matrix can be reused for multiple right-hand side vectors, one for each parameter in the finite-element model. The majority of computational effort for the DDM is devoted to the assembly of the right-hand side vector. Whereas the vector $\partial \mathbf{P}_f / \partial \theta$ is nonzero for only those parameters that represent the applied loads, the vector $\partial \mathbf{P}_r / \partial \theta|_{\mathbf{U}}$ must be assembled from element contributions in the same manner as the resisting force vector itself.

Classes for DDM Response Sensitivity

To assemble and solve the system of response sensitivity equations [Eq. (1)], two classes shown in Fig. 8 interface directly with the core finite-element assembly and solution modules of OpenSees. For each parameter in the finite-element model, the SensitivityAlgorithm class solves Eq. (1) after requesting an instance of the SensitivityIntegrator class to form the right-hand side vector. The SensitivityAlgorithm uses an instance of the LinearSOE class (McKenna 1997), which encapsulates the factorization of the tangent stiffness matrix $\mathbf{K}$, in order to solve Eq. (1). The assembly of the right-hand side vector depends on the integration method used in the finite-element analysis, where separate implementations of SensitivityIntegrator are provided for static analysis and various time integration methods for dynamic analysis, e.g., Newmark (1959). The interaction between the two response sensitivity classes and the finite-element framework is detailed herein.

After convergence of the finite-element solution for a given time step in the OpenSees analysis, a sequence of operations is orchestrated between the SensitivityAlgorithm and the appropriate implementation of the SensitivityIntegrator class in order to compute the DDM response sensitivity, as shown in Fig. 8.

1. The SensitivityAlgorithm requests the Integrator of the ordinary finite-element solution to form the tangent stiffness matrix at the solution point, which is required for Eq. (1). Unless a Newton–Raphson solution strategy was employed in the finite-element analysis, the last matrix assembled by the Integrator may not reflect the tangent stiffness at the converged state.
2. For each parameter in the finite-element model, the following steps are taken by the SensitivityAlgorithm to compute the corresponding response sensitivity in a path-dependent finite-element analysis:
   - Ask the SensitivityIntegrator to form the right-hand side vector of Eq. (1) for the current parameter. The SensitivityIntegrator assembles this vector from contributions of the element and load objects in the OpenSees domain according to the type of time integration method employed in the finite-element analysis. This step implements the first phase of the two-phase process described by Zhang and Der Kiureghian (1993).
   - Ask the LinearSOE, which encapsulates the factorization of the tangent stiffness matrix, to solve the system of simultaneous equations for the response sensitivity, $\partial \mathbf{U} / \partial \theta$, associated with the current parameter.
   - Pass the nodal response sensitivity to the SensitivityIntegrator such that it can iterate over all element objects in the finite-element model and allow them to use the solution to update their history variables for path-dependent behavior and prepare for the next time step in the analysis. This step completes the two-phase process for path-dependent sensitivity computations.

The one-to-many relationship between a SensitivityAlgorithm and implementations of the SensitivityIntegrator class exemplifies the Strategy design pattern (Gamma et al. 1995).

Response Sensitivity Calculations Using the Visitor Pattern

An important part of the software framework for sensitivity computations is the use of the Parameter class to ensure that the correct contributions to the right-hand side of Eq. (1) are com-

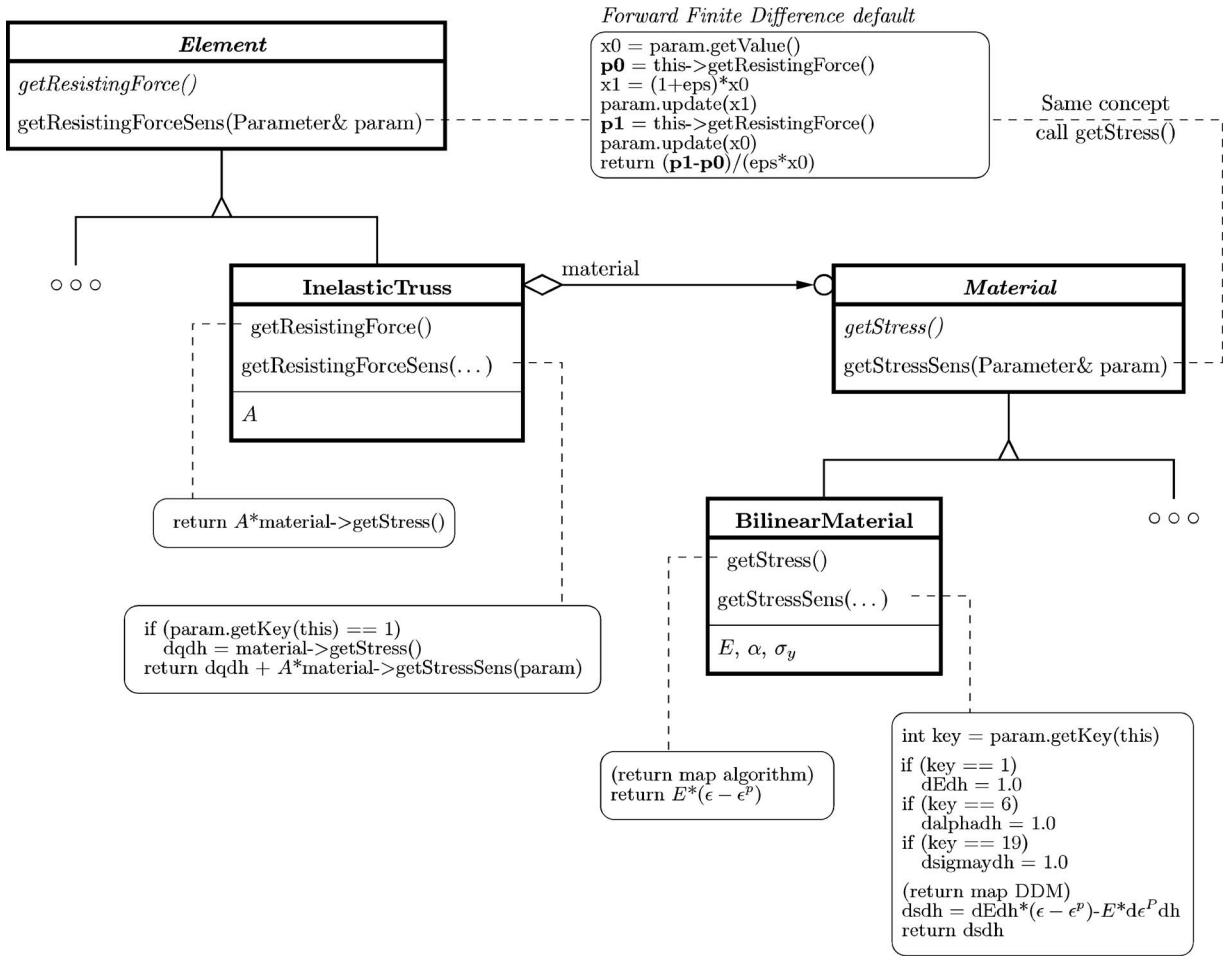

Fig. 9. Class diagram for the InelasticTruss showing methods to compute DDM resisting force sensitivity

puted. To this end, as shown in Fig. 8, the SensitivityAlgorithm passes the current Parameter object through each method call to the SensitivityIntegrator. There are two important design advantages to allowing a Parameter to visit each object that contributes to the right-hand side vector:

1. Each object can invoke the callback method Parameter::getKey( ) in order to determine if it encapsulates the passed parameter, and thus compute the correct contribution to the DDM sensitivity equations. If the ParameterizedObject is in the (key, pointer) list for the Parameter, this method returns the key; otherwise it returns 0 and the finite-element (FE) object can proceed with DDM computations accordingly. In a path-dependent analysis it is possible for a FE object to return a nonzero sensitivity for all model parameters, not just those it encapsulates (Haukaas and Der Kiureghian 2005).
2. A default implementation to compute response sensitivity by finite differences is provided in the base class for element and material models. The forward finite difference calculation is carried out as shown in Fig. 9. This approach enables finite difference calculations to be carried out for element and material models with complex state determination algorithms whereas the DDM is employed for the higher level response equations.

This approach to invoking callback methods during the response sensitivity calculations mirrors the Visitor design pattern (Gamma et al. 1995). The implementation pseudocode in Fig. 9 demonstrates how a Parameter visits each object in order to assemble the correct contribution to the DDM response sensitivity equations.

Example Response Sensitivity Application

A parameter sensitivity analysis of the nonlinear truss structure shown in Fig. 5 demonstrates the response sensitivity framework. For this example, the truss analysis is deterministic and the gradient of the nodal displacement is computed with respect to the three parameters (cross-sectional area of Member 1, yield stress of Members 1 and 3, and horizontal load at Node 4). The analysis is carried out via a Tcl script in order to show the steps taken by a sensitivity-based model updating application. The `computeGradients` command is added to the Tcl interpreter and when issued it invokes the response sensitivity computations described in the previous section.

The parameter sensitivity analysis is carried out using the Tcl commands shown in Fig. 10. The DDM response sensitivity is computed at each of the ten load steps in the pushover analysis. A first-order approximation is made to assess the effect of changes in the parameter values on the load-displacement response of the truss

```
# Define FE model
source trussModel.tcl

# Identify parameter
parameter 1 element 1 A
parameter 2 eleList 1 3 sigmaY
parameter 3 pattern 1 nodalLoad 4 1

# Compute DDM sensitivity at each step
for {set i 1} {$i <= 10} {incr i} {
  analyze 1
  computeGradients
}
```

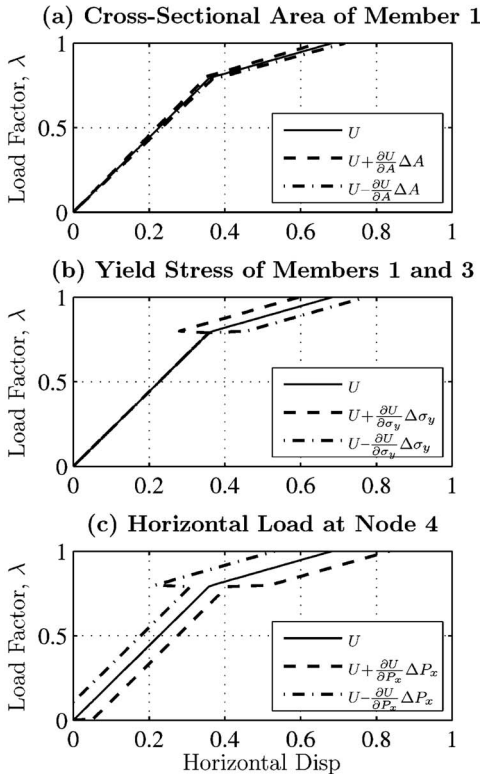

Fig. 10. Parametric sensitivity analysis of three member truss model showing response and first-order change in response using DDM response sensitivity and ±10% changes in: (a) cross-sectional area of Member 1; (b) yield stress of Members 1 and 3; and (c) horizontal load at Node 4

$$\Delta U = \frac{\partial U}{\partial \theta}\Delta\theta \tag{2}$$

where $\partial U/\partial\theta$=DDM response sensitivity computed with respect to each model parameter $\theta=\{A,\sigma_y,P_x\}$. The change in each parameter, $\Delta\theta$, is assumed to be ±10% of its mean value assigned in trussModel.tcl. Sensitivity with respect to each parameter is shown in Fig. 10, where it is noted that the truss response is most sensitive to relative changes in the applied loading. As expected, there is zero sensitivity to the material yield stress prior to reaching the elastic limit, at which point there is a discrete jump in the DDM results (Conte et al. 2003).

Damage Detection Application

A full-scale seven-story reinforced concrete building slice tested on the UCSD-NEES shake table (Ozcelik et al. 2008) demonstrates the application of the parameterization framework to a damage detection analysis. An elevation view of the test structure is shown Fig. 11 and structural details can be found at http://nees.ucsd.edu/7Story.html. A finite-element model updating strategy is applied where the objective function is a combination of residuals in natural frequency, mode shape components, and pseudomodal flexibility matrix components. The elastic modulus of each structural component is used as a modal parameter, giving a total of seven parameters to be identified using ambient vibration data. Four-noded shell finite elements model the web wall (Structural Components C1–C7). As shown in Fig. 11, Component C5 is discretized into four shell elements, and a single instance of the Parameter class maps to the elastic modulus, E, at each of the 16 integration points.

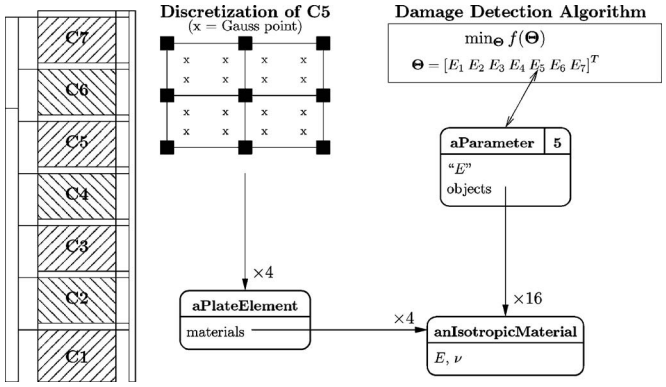

Fig. 11. Reinforced concrete shear wall building and parametrization of Structural Component C5 for the purposes of damage detection

Measured (initial) values of the elastic modulus for each structural component (C1–C7) of the shear wall building are listed in Table 1. After subjecting the building to ambient vibration on the shake table, the finite-element model is updated to reflect the recorded dynamic response. Updated values of the elastic modulus for Components C1–C7 are shown in Table 1, and serve as reference values for the subsequent damage identification analysis of the building for successively higher levels of shake table excitation. He et al. (2006) provide further details of the numerical results of the damage detection analysis.

Concluding Remarks

An object-oriented design to support model updating and response sensitivity applications in the OpenSees finite-element

Table 1. Initial and Updated Values of Elastic Modulus of Structural Components in the UCSD-NEES Reinforced Concrete Shear Wall Building (He et al. 2006)

Structural component	Initial value (GPa)	Reference value (GPa)
C1	24.47	19.83
C2	26.00	24.13
C3	34.84	36.90
C4	30.20	39.26
C5	28.90	33.95
C6	32.14	16.15
C7	33.54	16.77

framework has been developed. Parameters are identified and updated using a top-down approach aided by the Facade and Chain of Responsibility design patterns. Analytical DDM response sensitivity computations are implemented using the Strategy and Visitor patterns. Example applications written in the Tcl scripting language demonstrate the framework for model updating and DDM response sensitivity. The framework is able to support a wide array of future extensions that use OpenSees as a computational engine for reliability, optimization, and damage detection. These extensions include, but are not limited to, high-fidelity reliability analysis to automatically reduce model error and the incorporation of multihazard, nonlinear finite-element reliability analysis in structural health monitoring applications.

Acknowledgments

This work and its implementation in OpenSees have been supported by the National Science Foundation through Grant No. EEC-9701658 awarded to the Univ. of California, Berkeley and a subaward to Oregon State Univ. made by the Univ. of California, San Diego through Grant No. CNS-0205720.

References

ABAQUS, Inc. (2006). *ABAQUS/standard user's manual—Version 6.6*, Pawtucket, R.I., ⟨http://www.hks.com⟩.

Archer, G. C., Fenves, G., and Thewalt, C. (1999). "A new object-oriented finite-element analysis architecture." *Comput. Struct.*, 70(1), 63–75.

Baugh, J. W., and Rehak, D. R. (1992). "Data abstraction in engineering software development." *J. Comput. Civ. Eng.*, 6(3), 282–301.

Conte, J. P., Vijalapura, P. K., and Meghalla, M. (2003). "Consistent finite-element response sensitivity analysis." *J. Eng. Mech.*, 129(12), 1380–1393.

Fenves, G. L. (1990). "Object-oriented programming for engineering software development." *Eng. Comput.*, 6(1), 1–15.

Forde, B. W. R., Foschi, R. O., and Stiemer, S. F. (1990). "Object-oriented finite-element analysis." *Comput. Struct.*, 34(3), 355–374.

Franchin, P. (2004). "Reliability of uncertain inelastic structures under earthquake excitation." *J. Eng. Mech.*, 130(2), 180–191.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*, Addison-Wesley, Reading, Mass.

Gil, L., and Bugeda, G. (2001). "A C++ object-oriented programming strategy for the implementation of the finite-element sensitivity analysis for a nonlinear structural material model." *Adv. Eng. Software*, 32(12), 927–935.

Haukaas, T., and Der Kiureghian, A. (2005). "Parameter sensitivity and importance measures in nonlinear finite-element reliability analysis." *J. Eng. Mech.*, 131(10), 1013–1026.

Haukaas, T., and Der Kiureghian, A. (2007). "Methods and object-oriented software for FE reliability and sensitivity analysis with application to a bridge structure." *J. Comput. Civ. Eng.*, 21(3), 151–163.

He, X., Moaveni, B., Conte, J. P., Restrepo, J. I., and Elgamal, A. (2006). "Damage identification of a seven-story reinforced concrete shear wall building tested on the UCSD-NEES shake table." *4th World Conf. on Structural Control and Monitoring*, San Diego, Calif.

Kleiber, M., Antunez, H., Hien, T. D., and Kowalczyk, P. (1997). *Parameter sensitivity in nonlinear mechanics*, Wiley, New York.

Lee, H. H., and Arora, J. S. (1991). "Object-oriented programming for engineering applications." *Eng. Comput.*, 7(4), 225–235.

Liu, P. L., and Der Kiureghian, A. (1991). "Optimization algorithms for structural reliability." *Struct. Safety*, 9(3), 161–177.

Mackerle, J. (2004). "Object-oriented programming in FEM and BEM: A bibliography (1990–2003)." *Adv. Eng. Software*, 35(6), 325–336.

Mackie, R. I. (1992). "Object-oriented programming of the finite-element method." *Int. J. Numer. Methods Eng.*, 35(2), 425–436.

Mazzoni, S., McKenna, F., Scott, M. H., and Fenves, G. L. (2006). "Open system for earthquake engineering simulation user command-language manual—Version 1.7.3." Univ. of California, Berkeley, Calif., ⟨http://opensees.berkeley.edu/OpenSees/manuals/usermanual/⟩.

McKenna, F. (1997). "Object-oriented finite-element programming: Frameworks for analysis, algorithms, and parallel computing." Ph.D. thesis, Univ. of California, Berkeley, Calif.

McKenna, F., and Fenves, G. L. (2000). "An object-oriented software design for parallel structural analysis." *Proc., Structures Congress, 2000*, ASCE, Reston, Va.

McKenna, F., Fenves, G. L., and Scott, M. H. (2000). "Open system for earthquake engineering simulation." Univ. of California, Berkeley, CA, ⟨http://opensees.berkeley.edu⟩.

Miller, G. R. (1991). "An object-oriented approach to structural analysis and design." *Comput. Struct.*, 40(1), 75–82.

Modak, S., and Sotelino, E. D. (2002). "An object-oriented parallel programming framework for linear and nonlinear transient analysis of structures." *Comput. Struct.*, 80(1), 77–84.

Newmark, N. M. (1959). "A method of computation for structural dynamics." *J. Engrg. Mech. Div.*, 85, 67–94.

Ousterhout, J. K. (1994). *Tcl and the Tk toolkit*, Addison-Wesley, Reading, Mass.

Ozcelik, O., Luco, J. E., and Conte, J. P. (2008). "Identification of the mechanical subsystem of the NEES-UCSD shake table by a least-squares approach." *J. Eng. Mech.*, 134(1), 23–34.

Peng, J., and Law, K. H. (2004). "Building finite-element analysis programs in distributed services environment." *Comput. Struct.*, 82(22), 1813–1833.

Ries, R., and Mahdavi, A. (2001). "Integrated computational live-cycle assessment of buildings." *J. Comput. Civ. Eng.*, 15(1), 59–66.

Roth, C., and Grigoriu, M. (2001). "Sensitivity analysis of dynamic systems subjected to seismic loads." *Rep. No. MCEER-01-0003*, State University of New York, Buffalo, N.Y.

Rucki, M. D., and Miller, G. R. (1996). "An algorithmic framework for flexible finite-element-based structural modeling." *Comput. Methods Appl. Mech. Eng.*, 136(3–4), 363–384.

Schellenberg, A., and Mahin, S. (2006). "Integration of hybrid simulation within the general-purpose computational framework OpenSees." *8th U.S. National Conf. on Earthquake Engineering*, San Francisco, Calif.

Scott, M. H., Fenves, G. L., McKenna, F. T., and Filippou, F. C. (2008). "Software patterns for nonlinear beam-column models." *J. Struct. Eng.*, 134(4), 562–571.

Scott, M. H., Franchin, P., Fenves, G. L., and Filippou, F. C. (2004). "Response sensitivity for nonlinear beam-column elements." *J. Struct. Eng.*, 130(9), 1281–1288.

Silva, C. A. C., and Bittencourt, M. L. (2000). "An object-oriented structural optimization program." *Struct. Multidiscip. Optim.*, 20(2), 154–166.

Simo, J. C., and Hughes, T. J. R. (1998). *Computational inelasticity*, Springer, New York.

Soh, C. K., and Dong, Y. X. (2001). "Evolutionary programming for inverse problems in civil engineering." *J. Comput. Civ. Eng.*, 15(2), 144–150.

Taylor, R. L. (2004). "FEAP: A finite-element analysis program, version 7.5 user manual." Univ. of California, Berkeley, ⟨http://www.ce.berkeley.edu/~rlt/feap⟩.

TNO Building and Construction Research. (2003). Dept. of Computational Mechanics, Delft, The Netherlands, ⟨http://www.diana/tno.nl.DIANA⟩.

Welch, B. B. (2000). *Practical programming in Tcl and Tk*, 3rd Ed., Prentice-Hall, Upper Saddle River, N.J.

Wetherill, M., Rezgui, Y., Boddy, S., and Cooper, G. S. (2007). "Intra-

and interorganizational knowledge services to promote informed sustainability practices." *J. Comput. Civ. Eng.*, 21(2), 78–89.
Zhang, Y., and Der Kiureghian, A. (1993). "Dynamic response sensitivity of inelastic structures." *Comput. Methods Appl. Mech. Eng.*, 108(1–2), 23–36.
Zhang, Y., and Der Kiureghian, A. (1997). "Finite-element reliability methods for inelastic structures." *Rep. No. UCB/SEMM-97/05*, Dept. of Civil and Environmental Engineering, Univ. of California, Berkeley, Calif.
Zimmermann, T., Dubois-Pelerin, Y., and Bomme, P. (1992). "Object-oriented finite-element programming. I: Governing principles." *Comput. Methods Appl. Mech. Eng.*, 98(2), 291–303.