OpenSeesPy: Python library for the OpenSees finite element framework

Article · January 2018

DOI: 10.1016/j.softx.2017.10.009

CITATIONS
5

READS
1,279

3 authors:

Minjie Zhu
Oregon State University
9 PUBLICATIONS **64** CITATIONS
SEE PROFILE

Frank Mckenna
University of California, Berkeley
16 PUBLICATIONS **1,306** CITATIONS
SEE PROFILE

Michael H. Scott
Oregon State University
59 PUBLICATIONS **945** CITATIONS
SEE PROFILE

SoftwareX 7 (2018) 6–11

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

OpenSeesPy: Python library for the OpenSees finite element framework

Minjie Zhu [a], Frank McKenna [b], Michael H. Scott [a,*]

[a] School of Civil and Construction Engineering, Oregon State University, 101 Kearney Hall, Corvallis, OR 97331, USA
[b] Department of Civil and Environmental Engineering, University of California, 760 Davis Hall, Berkeley, CA 94720, USA

ARTICLE INFO

Article history:
Received 4 October 2017
Accepted 30 October 2017

Keywords:
Interpreter
Scripting language
Structural analysis
Finite element analysis

ABSTRACT

OpenSees, an open source finite element software framework, has been used broadly in the earthquake engineering community for simulating the seismic response of structural and geotechnical systems. The framework allows users to perform finite element analysis with a scripting language and for developers to create both serial and parallel finite element computer applications as interpreters. For the last 15 years, Tcl has been the primary scripting language to which the model building and analysis modules of OpenSees are linked. To provide users with different scripting language options, particularly Python, the OpenSees interpreter interface was refactored to provide multi-interpreter capabilities. This refactoring, resulting in the creation of OpenSeesPy as a Python module, is accomplished through an abstract interface for interpreter calls with concrete implementations for different scripting languages. Through this approach, users are able to develop applications that utilize the unique features of several scripting languages while taking advantage of advanced finite element analysis models and algorithms.

Code metadata

Current code version	Version 2.5.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-17-00072
Legal Code License	https://github.com/fmckenna/OpenSeesInterpreter/blob/master/LICENSE
Code versioning system used	git, svn
Software code languages, tools, and services used	C++, FORTRAN, Tcl, Python
Compilation requirements, operating environments & dependencies	Linux, Windows, Mac OS
If available Link to developer documentation/manual	http://opensees.berkeley.edu/OpenSees/developer/index.php
Support email for questions	opensees-support@berkeley.edu

1. Motivation and significance

Simulation plays a critical role in the design and assessment of civil infrastructure to resist natural hazards such as earthquakes and tsunamis. While simplified approaches can be used for preliminary analyses, the results are often conservative and can be costly for design and retrofit. As a result, design codes allow engineers to develop more economical solutions that satisfy performance objectives ranging from immediate occupancy to life safety. Simulation helps achieve these solutions in engineering practice and also quantify probabilities of exceeding limit states, e.g., using stochastic methods that consider the variability of system properties and external loading.

Developing new simulation models requires a flexible software framework where developers can build off of existing, interchangeable modules. A fully functional finite element analysis software, OpenSees, the Open System for Earthquake Engineering Simulation, was created in the late 1990s as the simulation platform for the Pacific Earthquake Engineering Research (PEER) Center. The OpenSees framework combines state of the art finite

* Corresponding author.
E-mail addresses: zhum@oregonstate.edu (M. Zhu), fmckenna@berkeley.edu (F. McKenna), michael.scott@oregonstate.edu (M.H. Scott).

https://doi.org/10.1016/j.softx.2017.10.009

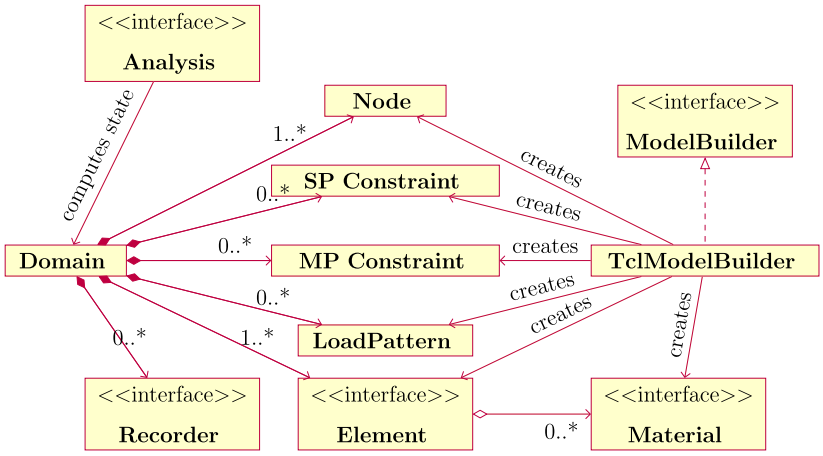


Fig. 1. Class diagram of existing OpenSees framework.

element models and nonlinear solution algorithms and is open source, written primarily in C++ with linkages to some FORTRAN libraries for solving linear systems of equations.

Although OpenSees also has extensions to Graphic User Interfaces (GUIs), such as BuildingTcl and OpenSees Navigator, they typically constrain the user to a particular type of structure and/or analysis [1]. To allow users to take advantage of programming constructs such as conditionals, iterations, and procedures, OpenSees extends the interpreter of Tcl with commands for model building and analysis. While Tcl is very flexible and has many strengths in string processing, it is not well-suited to scientific computing applications. Tcl's use of strings as the only native data type and the resulting cumbersome syntax for mathematical expressions create a steep learning curve for users of OpenSees, many of whom learned MATLAB or Python as undergraduate engineering students.

To remove the limitations of a single scripting language and to permit a wide range of future scripting languages, this paper presents a new interface for the OpenSees framework, the `DL_Interpreter` class to include multiple interpreters making OpenSees accessible to a broader range of users. In addition, users of OpenSees will be able to utilize functionality from different scripting languages, functionality that is not available in Tcl. The focus herein is on OpenSeesPy, an implementation of the Python interpreter for use with OpenSees; however, the underlying framework is general and can easily accommodate other scripting languages such as Ruby, Julia, and R. The interpreter-independent functions in the OpenSees core and their interaction with the new `DL_Interpreter` class are described. An example application of a Monte Carlo simulation of a nonlinear truss structure highlights the multi-interpreter interface. The paper concludes with an assessment of the impact of the use of OpenSees.

2. Software description

The core functionality of OpenSees, including state-of-the-art finite element models and solution algorithms for nonlinear dynamic analysis of structural and geotechnical systems, is described in [2]. In addition to its capabilities for earthquake engineering simulation, OpenSees has been extended for parameter updating and sensitivity analysis [3], fire simulation [4], and fluid–structure interaction [5]. With a wide range of finite element simulation capabilities, the extension of OpenSees with an interface for new interpreters allows users to take advantage of several libraries and packages in developing specialized applications.

2.1. Software architecture

To analyze a structural or geotechnical system in OpenSees, a `ModelBuilder` object populates the `Domain` with the node, element, load, and constraint objects necessary to build a finite element model of the system and `Recorder` objects to record the analysis results [2,6], as shown in Fig. 1. An `Analysis` object computes the state of the finite element model for a particular type of analysis, e.g., static, dynamic, eigenvalue, etc., using interchangeable solution algorithms, linear equation solvers, constraint handlers, and time integration methods.

Users of OpenSees write scripts that invoke an instance of the `ModelBuilder` class to create the finite element model, e.g., the `TclModelBuilder` class extends the Tcl scripting language with a `node` command that creates a `Node` object from user input by calling functions in the Tcl API [7], such as `Tcl_GetInt`, then adds the `Node` object to the `Domain`. To link OpenSees with Python, or any other scripting language, much of this code for reading input, creating finite element domain objects, performing analyses, and recording results would need to be duplicated. To avoid such code duplication, the `ModelBuilder` object is replaced by a core, interpreter-independent OpenSees API that populates the domain and performs analyses, as shown in Fig. 2.

2.2. Software functionalities

Multi-interpreter functionality for OpenSees is provided through the `DL_Interpreter` interface shown at the top of Fig. 2, which defines eight basic input/output (IO) functions for obtaining integer, floating point, and string inputs. Each scripting language will define its own interpreter class that implements the `DL_Interpreter` interface, e.g. `TclInterpreter` and `Python-Interpreter`, for the eight IO functions. Each implementation deals with language related operations, such as initializing the language, printing information, setting up environmental variables, loading startup modules and files, and extending functions or commands.

Since language related codes could be cumbersome, a wrapper class is defined within an interpreter class to share the responsibility of wrapping OpenSees core APIs. Therefore, the core APIs only contain pure OpenSees code for populating the model domain, running analyses, and manipulating internal objects, so that these operations work identically across all interpreter languages. If an IO operation is required, a core API will call the OpenSees IO APIs, e.g., `OPS_GetIntInput`, which are also interpreter-independent and will make polymorphic calls to the eight basic IO functions through the `DL_Interpreter` interface. With this design, the core OpenSees APIs are completely separated from the interpreter and the effort to add a new interpreter is minimized.

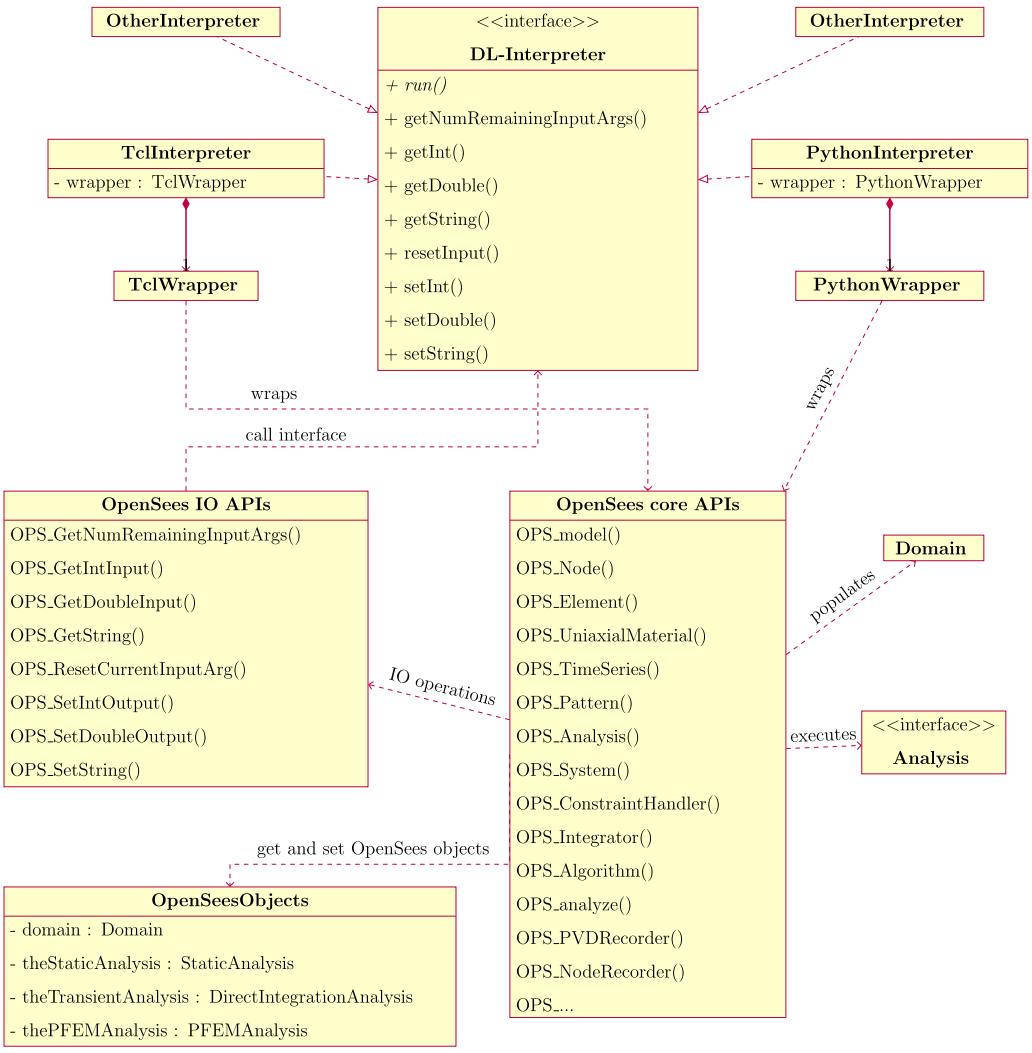


Fig. 2. Class diagram of multi-interpreter interface.

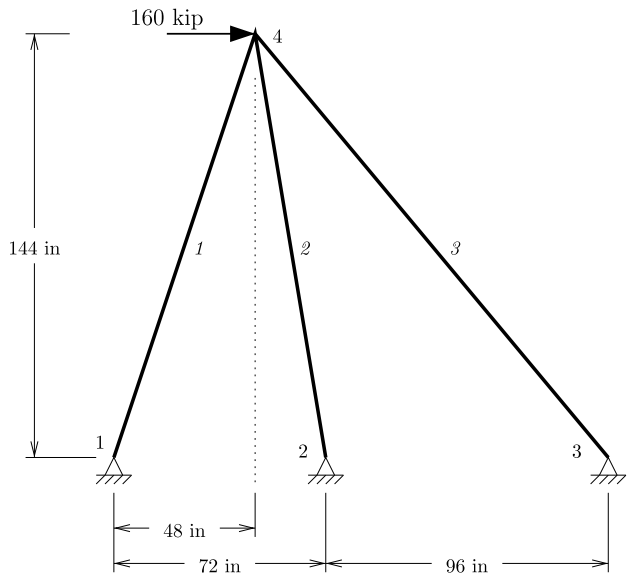


Fig. 3. Truss model for Monte Carlo simulation.

3. Illustrative examples

The power of OpenSees is to create models, execute analyses, and record results dynamically through scripting. The simple material nonlinear truss structure shown in Fig. 3 is created and analyzed. The intent here is not to show a full structural analysis, but rather to highlight the underlying code that enables the use of OpenSees through multiple interpreters.

3.1. Model building

In the truss model shown in Fig. 3, nodes are numbered from one through four and elements one through three. Snippets of Tcl and Python code to create the truss model are shown in Fig. 4. Although the two languages have very different syntax and formats for function calling, the function input parameters are always listed serially, which is true for almost all general purpose and scientific languages.

As previously described, interpreter classes implement the eight basic IO functions through the `DL_Interpreter` interface, which read and record in the same order internally for all languages. For example, the `OPS_GetDoubleInput` function calls the `getDouble()` virtual method to read the next input parameter as a floating point value, e.g. material elastic modulus, E, in Fig. 4. The

```
# Python
x = [0.0, 72.0, 168.0, 48.0]
y = [0.0, 0.0, 0.0, 144.0]
for i in range(4):
  node(i+1, x[i], y[i])


A = 4.0
E = 29000.0
alpha = 0.05
sigmaY = 36.0
uniaxialMaterial('Hardening',
1, E, sigmaY, 0.0,
alpha/(1-alpha)*E)

for i in [1,2,3]:
  element('Truss',i,i,4,A,1)
```

```
# Tcl
set x {0.0 72.0 168.0 48.0}
set y {0.0 0.0 0.0 144.0}
for {set i 0} {$i<4} {incr i} {
  node [expr $i+1] [lindex $x $i] \
           [lindex $y $i]
}
set A 4.0
set E 29000.0
set alpha 0.05
set sigmaY 36.0
uniaxialMaterial Hardening \
1 $E $sigmaY 0.0 \
[expr $alpha/(1-$alpha)*$E]

foreach i {1 2 3} {
  element truss $i $i 4 $A 1
}
```

Fig. 4. Truss model building in Python and Tcl.

```
int PythonInterpreter::getDouble(double *data,int numArgs) {
  int curr = wrapper.getCurrentArg();
  if (wrapper.getNumberArgs()-curr < numArgs) return -1;
  for (int i=0; i<numArgs; i++) {
    if (!PyFloat_Check(o)) return -1;
    PyObject *o = PyTuple_GetItem(wrapper.getCurrentArgv(),curr);
    wrapper.incrCurrentArg();
    data[i] = PyFloat_AS_DOUBLE(o);
  }
  return 0;
}
```

Fig. 5. The implementation of `getDouble` in Python interpreter.

implementations of the `getDouble()` method for the `PythonInterpreter` and `TclInterpreter` classes are shown in Figs. 5 and 6, respectively.

By calling the IO APIs, the core APIs can read the input parameters or record output results. For example, the `OPS_Node` core API reads the node tag and nodal coordinates and creates a `Node` object in the domain. Wrapping the `OPS_Node` function makes it available as the `node` command in several scripting languages. Its implementation in the `PythonWrapper` and `TclWrapper` is shown in Figs. 7 and 8.

3.2. Simulation results

Since the same core API functions are called, separate interpreters should create finite element models that give the same simulation results. For the truss example, the external loads are applied to the structure in multiple steps of a static analysis. Both interpreters report the horizontal displacement of node 4 as $u_4 = 1.510431$ in at the peak load of 160 kip.

To demonstrate a reliability analysis, three structural parameters, truss section area A, material yield stress σ_y, and the horizontal load P_x, are defined and random numbers are generated from normal distribution for the parameters with mean values shown in Fig. 4. The standard deviations are 0.5 in^2, 2.0 ksi, and 10.0 kip for A, σ_y, and P_x, respectively. Monte Carlo simulation is then run and the probability of failure of the truss structure is calculated as shown in Fig. 9. Failure of the structure is defined in terms of the horizontal displacement of node 4 using the following limit state function

$$g(u_4) = 2.5 \text{ in} - u_4, \quad \begin{cases} \text{Safe} & : \text{if } g \geq 0 \\ \text{Fail} & : \text{if } g < 0 \end{cases} \tag{1}$$

```
int TclInterpreter::getDouble(double *data, int numArgs) {
  int curr = wrapper.getCurrentArg();
  const char* argv = wrapper.getCurrentArgv();
  if (wrapper.getNumberArgs()-curr < numArgs) return -1;
  for (int i=0; i<numArgs; i++) {
    if (Tcl_GetDouble(interp,argv[curr],&data[i]) != TCL_OK) {
      wrapper.incrCurrentArg();
      return -1;
    } else wrapper.incrCurrentArg();
  }
  return 0;
}
```

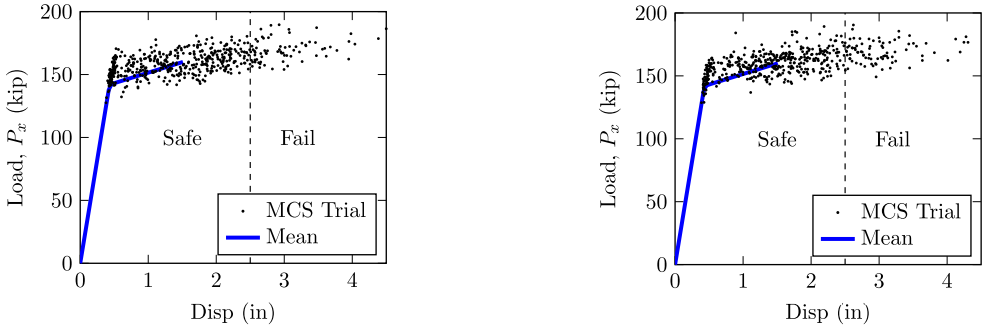
Fig. 6. The implementation of `getDouble` in Tcl interpreter.

```
static PyObject *Py_ops_node(PyObject *self,PyObject *args) {
    wrapper->resetCommandLine(PyTuple_Size(args),1,args);
    if (OPS_Node() < 0) return NULL;
    return wrapper->getResults();
}
```

Fig. 7. Wrapping `OPS_Node` in Python interpreter.

```
static int Tcl_ops_node(ClientData clientData,Tcl_Interp *interp,
                        int argc, TCL_Char **argv) {
    wrapper->resetCommandLine(argc, 1, argv);
    if (OPS_Node() < 0) return TCL_ERROR;
    return TCL_OK;
}
```

Fig. 8. Wrapping `OPS_Node` in Tcl interpreter.



(a) Python. (b) Tcl.

Fig. 9. Monte Carlo simulation of example truss structure using Python and Tcl interpreters for OpenSees.

The solid line is the load–displacement relationship with mean values of the random variables. The dashed line in Fig. 9 delineates the failure and safe domains based on the horizontal displacement of node 4. Each dot is the final displacement for one set of random variable realizations.

Due to differences in their random number generators, the Python and Tcl interpreters lead to probabilities of failure of P_f = 0.1577 and 0.1589, respectively, after 10,000 random trials. This means the horizontal displacement of the truss has a nearly 16% chance of exceeding the prescribed limit considering the random

variable distributions of each parameter. Realistic structures are more complex; however, this example conveys the essential ideas of the multiple interpreter capability.

4. Impact

From its early use by researchers in PEER, OpenSees has grown to a large international user base. In 2016, there were over 1.5 million page views at its website http://opensees.berkeley.edu. OpenSees has been used as the primary simulation tool for numerous graduate theses in the United States and at major universities worldwide. The applications of OpenSees are cross platform (Windows, Linux, and OS X), range from simple serial applications through complicated parallel applications, and are widely used on a range of computing resources, including desktops, Amazon clusters, and high performance computing platforms. The finite element modules of OpenSees can also serve as the analysis engine for developing fragility functions that quantify the resilience of connected systems of civil infrastructure to a wide variety of natural hazards. To address research questions around community resilience, the IN-CORE multi-hazard decision framework [8] uses Python as its glue language for linking structural fragilities generated by OpenSees to investment decisions for bridges, buildings, and electrical transmission systems.

With its extension to Python, users of OpenSees gain access to the plotting library `Matplotlib`, numerical libraries `Numpy` and `Scipy`, interactive server `Jupyter`, 3-D visualization library `Mayavi`, statistics library `pandas`, web development library `Flask`, among others. In addition, OpenSees can now use the Visualization Toolkit (VTK), an open source, permissively licensed, cross-platform toolkit for scientific data processing, visualization, and data analysis [9], as one of its post-processing tools.

5. Conclusions

The multiple interpreter capability of OpenSees reflects its evolution to keep pace with new scripting languages from the software computing community. New interpreters can be added quickly to take advantage of libraries and modules from the scripting languages. With the development of the OpenSees core APIs, which are separate from the interpreters, the finite element modules of OpenSees can be re-structured to utilize new advances from the engineering mechanics and numerical methods communities at the source code level. As an open source framework, it is expected that this capability will keep OpenSees relevant within a large user base and with advances on both fronts of scripting languages and nonlinear finite element analysis.

Acknowledgments

Partial support for this work was provided by cooperative agreement 70NANB15H044 between the National Institute of Standards and Technology (NIST) and Colorado State University through a subaward to Oregon State University. The contents of this paper are the views of the authors and do not necessarily represent the opinions or views of NIST or the U.S. Department of Commerce.

References

[1] AlHamaydeh M, Najib M, Alawnah S. INSPECT: A graphical user interface software package for IDARC-2D. SoftwareX 2016;5:243–51.
[2] McKenna F, Scott MH, Fenves GL. Nonlinear finite-element analysis software architecture using object composition. J Comput Civil Eng 2010;24(1):95–107.
[3] Scott MH, Haukaas T. Software framework for parameter updating and finite-element response sensitivity analysis. J Comput Civil Eng 2008;22(5):281–91.
[4] Jiang J, Usmani A. Modeling of steel frame structures in fire using OpenSees. Comput Struct 2013;118:90–9. http://dx.doi.org/10.1016/j.compstruc.2012.07.013.
[5] Zhu M, Scott MH. Modeling fluid-structure interaction by the particle finite element method in OpenSees. Comput Struct 2014;132:12–21.
[6] Scott MH, Fenves GL, McKenna FT, Filippou FC. Software patterns for nonlinear beam-column models. J Struct Eng 2008;134(4):562–71.
[7] Welch BB. Practical Programming in Tcl and Tk. Perntice Hall, Upper Saddle River, NJ; 2000.
[8] Center for risk-based community resilience planning. 2017 IN-CORE (Interdependent Networked Community Resilience Modeling Environment) NIST, http://resilience.colostate.edu/in_core.shtml.
[9] Hanwell MD, Martin KM, Chaudhary A, Avila LS. The visualization toolkit (VTK): Rewriting the rendering code for modern graphics cards. SoftwareX 2015;1–2:9–12.