SCRIPTED BUILDING ENERGY MODELING AND ANALYSIS

Elaine Hale (elaine.hale@nrel.gov), Daniel Macumber (daniel.macumber@nrel.gov),
Kyle Benne (kyle.benne@nrel.gov), and David Goldwasser (david.goldwasser@nrel.gov)
National Renewable Energy Laboratory, Golden, CO

ABSTRACT

Building energy modeling and analysis is currently a time-intensive, error-prone, and nonreproducible process. From mundane file management tasks, to repeated entry of model parameters, to the application of often-used design transformations, to the execution of large-scale analyses, workflow automation via user-defined scripts has the potential to reduce costs and improve the quality of energy modeling. Many sophisticated building energy modelers already know this and regularly create various scripts to automate portions of their workflows; however, each practitioner must create custom solutions from scratch, which lead to new inefficiencies and potential errors. This paper describes the scripting platform of the OpenStudio tool suite (http://openstudio.nrel.gov) and demonstrates its use in several contexts.

Two classes of scripts are described and demonstrated: measures and free-form scripts. Measures are small, single-purpose scripts that conform to a predefined interface. Because measures are fairly simple, they can be written or modified by inexperienced programmers. Because measures have a known interface, they can be shared among users and selected for use at several places in the OpenStudio tool chain. We demonstrate the use of measures in an interactive mode from the OpenStudio SketchUp plug-in and in a noninteractive mode from the OpenStudio application. More experienced users can design and write free-form scripts to automate their work. We demonstrate the advantages of conducting large-scale analysis using free-form scripts through a case study. Finally, a vision for future work in which measures are shared through online libraries is described.

INTRODUCTION

The state of the art in building energy modeling and simulation requires the coordinated use of graphical user interfaces (GUIs), spreadsheets, and text editors. Most users prefer to use GUIs when the interface meets their needs. A GUI application may not, however, provide all the functionalities a user needs to modify input data, present output data, and create and compare multiple related models for a particular project. A GUI tool may also make it easy to execute a particular task once and in one place, but not provide a good workflow for repeating that task across a wider range of situations or systems. When a GUI comes up short in one or more of these areas, the user is forced to perform some manual steps in a spreadsheet or with the command line, which often takes additional time, introduces transcription errors, and results in a nonreproducible process.

Expert users may cobble together their own sets of tools for generating models, running simulations, and tracking results for many building energy simulations. This involves writing code for interacting with the simulation engine's inputs and outputs, invoking the simulation engine, and comparing results from multiple models. Most practitioners choose to implement such solutions using an interpreted scripting language (e.g., batch files, Python scripts, Ruby scripts, or MATLAB scripts) rather than a compiled language (e.g., C, C++, C#, or Fortran). This preference is attributable to the ease of use and rapid development/test cycles of interpreted languages. Furthermore, interpreted scripts can be deployed to multiple platforms without having to build a new package for each type of system. One disadvantage of scripting languages, however, is that there is no compile step to catch argument type mismatches and other common problems that can lead to runtime failures.

Such homegrown systems, which often involve a combination of manual and automated steps, certainly provide time savings, reduce errors, and improve reproducibility for the individual investigator. These custom solutions do, however, trade the manual transcription errors for the unavoidable syntatic and semantic errors of computer programming (bugs), especially for code that is not thoroughly tested. The proliferation of homegrown systems also wastes effort across the building energy simulation community, as opportunities for code reuse are lost.

Several software systems aim to fill this niche by better empowering building energy modelers and researchers

to 1) easily and reproducibly perform oft-repeated tasks; and 2) run large-scale analyses. GenOpt provides a library of optimization algorithms, a mechanism to associate variable values with particular manipulations of a text input file, and configuration settings that point to the simulation engine and establish a protocol for exchanging data (Wetter 2001). A generic software tool for multidisciplinary design optimization, Phoenix Integration's ModelCenter, has been applied to building design problems in which energy efficiency, as predicted by EnergyPlus, is one of the objectives to be optimized (Flager, Welle et al. 2009). Both tools automate some low-level tasks, such as file management and new model generation based on algorithm requests, but require users to define low-level manipulations of simulation input files to set up their problems. GUI tools are also available for running large-scale analyses. BEopt optimizes residential building designs simultaneously over an energy and an economic metric. As a GUI tool, it constrains the user by specifying exactly what and how features of the design can be manipulated. This makes the tool easy to use but sacrifices flexibility (NREL 2012).

In this paper we describe the OpenStudio scripting environment specifically designed for building energy modeling and analysis (BEMA). The OpenStudio scripting environment is integrated with the OpenStudio GUI applications so users can combine the benefits of graphical data representation and customized task automation. In particular, measures, which are scripts that follow a certain structure, can be used directly from the GUI applications, which then exercise the set structure to provide on-demand user interfaces. Free-form scripts may also be written and then run outside the GUIs, that is, from a command line. Because the scripting framework contains a formal mechanism for using measures as discrete or continuous variables, measures can also be used within free-form scripts.

In the following sections we describe the architecture of the system and the technologies that enable BEMA scripting. We then describe measures in more detail, giving examples that span several use cases. Next we provide a case study about how free-form scripts were used to develop and revise a Web tool in support of the U.S. 179D federal tax deduction. Finally, we provide a vision for future work in which measures are shared in an online repository and can be used to compose problem definitions for large-scale analysis, including design optimization.

ARCHITECTURE AND TECHNOLOGIES

The software architecture of the OpenStudio project is shown in *Figure 1*. At the center of the project is the OpenStudio software development kit (SDK), which is an open source (GNU lesser general public license) collection of reusable libraries implementing BEMA functionality. The building model library (Building Modeler) is at the center of SDK, and provides classes and methods for creating and modifying building energy models. For instance, the building geometry is

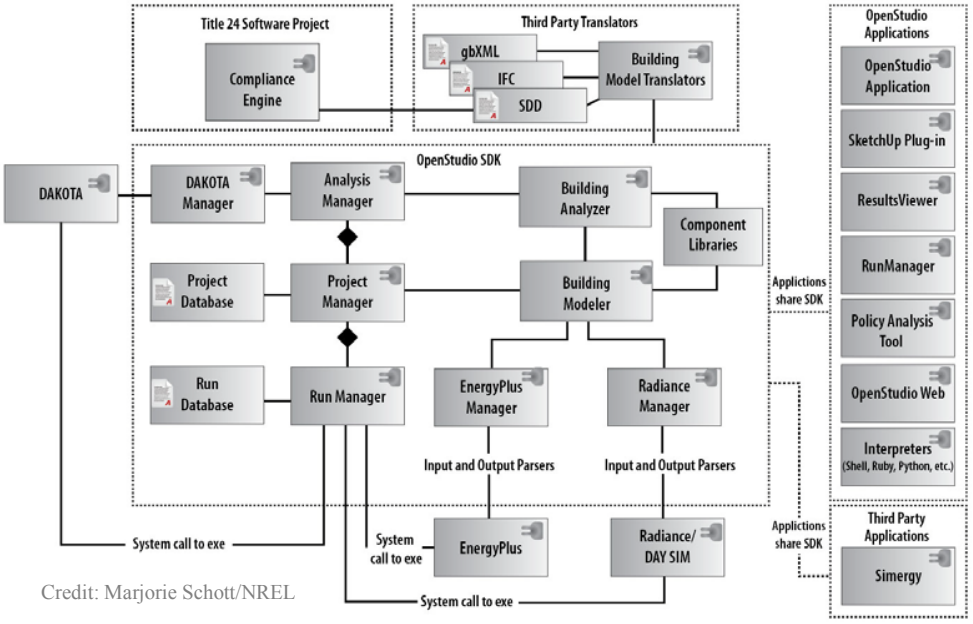


Figure 1: OpenStudio software architecture

represented by a set of spaces, each of which is bounded by a collection of surfaces. Thermal zones are defined by grouping spaces. Each such class (Space, Surface, and ThermalZone) has methods for inspecting and modifying low-level data and relationships. Classes often also define high-level methods which modify many low-level data fields and relationships. As an example, the Surface method for setting window to wall ratio will be described later in this paper.

The building model library is built on top of utility libraries for accessing the input and output data at a low level, opening and parsing related file types, performing geometric operations, etc. These low-level utility methods are also included in the SDK. Finally, the SDK includes modules for running simulations (EnergyPlus, EnergyPlus auxilliary programs, and Radiance), defining and running analyses, and storing results for multiple models and simulations (DOE 2011; Ward 2011). The OpenStudio SDK can be used to build additional applications and projects. Some current examples are shown in *Figure 1*. Several of these GUI tools are available directly from the OpenStudio project.

The OpenStudio SDK is implemented in C++. The functionality of the SDK is tested by an automated system that regularly checks out the source code, builds the project on several platforms, runs a series of unit tests on each, and posts results to a Web-based dashboard. Documentation for the SDK is generated using Doxygen and is posted to the OpenStudio project website for each release (van Heesch 2012). Functionality from the C++ SDK is then exposed to other languages using the Simplified Wrapper and Interface Generator (SWIG) (Beazley, Ballabio et al. 2009). SWIG enables export of C++ code to approximately twenty different languages. We have chosen to support Ruby as our primary scripting language in order to integrate with the Ruby plug-in architecture provided by Google SketchUp. We also export bindings to the C# language because several of our collaborators program in C#. Because C# is a compiled language, we will not discuss it further in this paper. Ruby, on the other hand, is an interpreted language, and we have made significant use of it beyond the basic need for communicating with SketchUp.

PHASES OF A BEMA PROJECT

A building energy analysis typically consists of four phases: 1) define the seed model or models; 2) define the problem to be investigated; 3) run the simulations supporting the analysis; and 4) generate, interpret, and incorporate tables and figures based on the simulation results into reports. The OpenStudio scripting environment exposes portions of the OpenStudio SDK relevant to each phase.

The seed model can be created with the GUI tools, the scripting environment, or a combination of both. In each case, classes and methods exposed in the OpenStudio SDK building model library are used to create objects relevant to the simulation study, assign values to their parameters, and define relationships between objects. A key feature of the building model library is that the objects and their methods are responsible for maintaining model validity. Other models in the analysis are generated from the seed model as described below.

Once the seed model is complete, the analysis problem must be defined. The problem consists of a parameter space, a number of response functions, and a prescribed simulation workflow. The parameter space is a list of variables, each of which defines how the seed model should be mainipulated to set the value of that variable. If the variable is discrete, it is a list of discrete (perhaps even unrelated) changes that can be made to the model; if the variable is continuous, some method for setting the variable to a particular value must be defined. Discrete and continuous variables may be defined using Ruby scripts (measures or free-form scripts). These scripts typically use methods from the building model library. For example, the building model library contains methods to loop over all surfaces in the model and set the window to wall ratio to a particular value. Response functions are typically high-level outputs such as site energy use; however, these may also be implemented using free-form scripts and can be configured to compute any value the user wants to analyze in detail. The simulation workflow consists of a string of jobs to be executed on each new model, such as: 1) translate to EnergyPlus syntax; 2) simulate with EnergyPlus; and 3) run the basic postprocess job to extract high-level results.

Once the problem has been defined, the user may select an algorithm to apply. Two algorithms are currently available directly in OpenStudio: a full factorial mesh over discrete variables, and a bi-objective optimizer over discrete variables. We are also creating an interface to relevant DAKOTA algorithms (Adams, Bohnhoff et al. 2009). We have exposed several sampling algorithms from the DDACE library, including Latin hypercube sampling, orthogonal array sampling, and grid sampling. Once the problem and algorithm are in place, OpenStudio functionality from the analysis driver portion of the SDK may be used to run the analysis. This functionality makes heavy use of the run manager library to queue jobs and keep communications synchronized. After each new building model is generated and simulated (by setting the

variable values and running the simulation workflow), the results are pushed to a database.

When the analysis is complete, the final phase of work begins, that is, figures and graphs are generated. The OpenStudio scripting environment includes functionality for making database queries to pull out high-level and detailed results for each data point, and for automatically preparing figures, graphs, and tables.

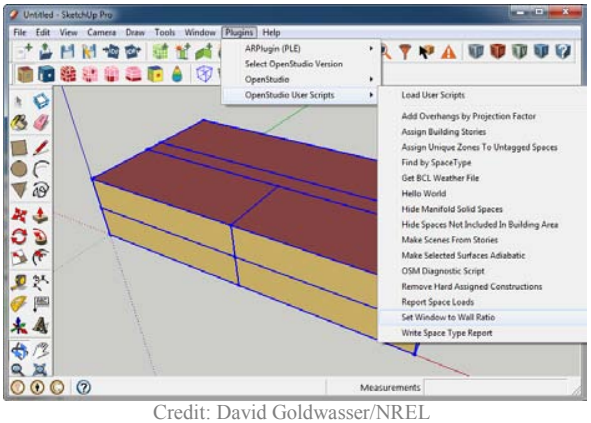
Repeatability is a key benefit of using scripts in each BEMA project phase. If an error is found in some part of the process, the seed model changes, a new variable is proposed, or a different algorithm is to be investigated, the analysis can be rerun with minimal effort. In the following sections we will discuss the use of measures and free-form scripts in automating a BEMA project.

MEASURE SCRIPTS

In OpenStudio, measures are scripts that implement a formal interface. This interface provides a way for host applications to get the name of the measure, identify the required and optional arguments, and run the measure with a given set of argument values. In this section, we provide two examples of applying measures. In the first example, the user works within the OpenStudio SketchUp plug-in and applies a measure interactively to the model. In the second example, the user defines a measure that is applied noninteractively each time the model is simulated. In both examples, the simplicity of the code snippets is directly traceable to the use of high-level methods implemented in the (documented, tested, and open source) C++ code base. These methods are responsible for all low-level tasks, such as file parsing, input file editing, and input file saving, thereby freeing the user to work with higher level abstractions.

Interactive Measures

Measures were initially developed for integration with the OpenStudio SketchUp plug-in, where they are referred to as *user scripts*. The purpose of these scripts is to allow users to automate tedious portions of their workflows. For example, a user may want to apply some property to each surface in a model. Rather than visit each surface individually and set the property through the GUI, the user can write a script that loops over all surfaces and sets the property in one step. To create a new user script, the user writes a Ruby script containing a class that implements the formal measure interface and places this script in the designated folder. The next time she launches SketchUp, the new script will show up next to all the others in the "OpenStudio User Scripts" menu. The user can then run this script at any time by selecting it from the menu, as shown in *Figure 2*.



Credit: David Goldwasser/NREL

Figure 2: Menu showing available user scripts

Several example user scripts are packaged with the plug-in. The packaged examples include a "Hello World" template script illustrating the syntax at its simplest, a summary table generator, a script that blends the OpenStudio and SketchUp APIs to create architectural plan views, and several single-task manipulations of the underlying energy model. In the latter category, the "Set Window to Wall Ratio" example user script removes any existing windows from the selected surfaces, and then replaces them with new windows matching the window to wall ratio and offset (from floor or ceiling) parameters given by the user. Each time the script is run, an input dialog, as shown in *Figure 3*, is dynamically generated to request argument values from the user.

The code that affects these changes in the model, with error checking code removed for brevity, follows:

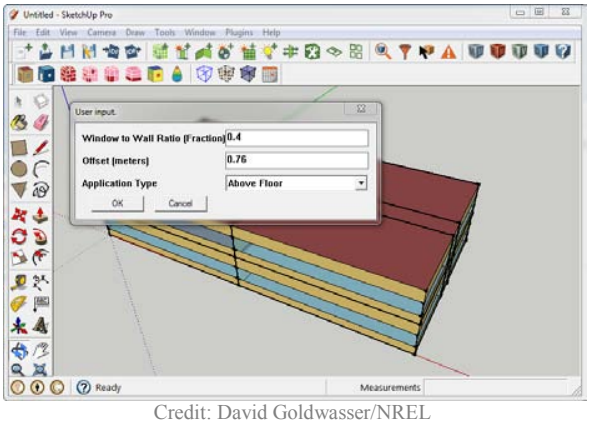
```
def run(model, runner, arguments)
 wwr = arguments["wwr"]
 offset = arguments["offset"]
 application_type =
     arguments["application_type"]

 heightOffsetFromFloor = nil
 if (application_type.valueAsString ==
     "Above Floor")
   heightOffsetFromFloor = true
 else
   heightOffsetFromFloor = false
 end

 model.getSurfaces.each do |s|
   next if not runner.inSelection(s)
   next if not (s.outsideBoundaryCondition ==
                "Outdoors")
   new_window = s.setWindowToWallRatio(
   wwr.valueAsDouble,
   offset.valueAsDouble,
   heightOffsetFromFloor)
 end
end
```

The script makes it easier to perform this modeling task in at least two ways. First, the user's selection within

the host application determines to which surfaces the transformation is applied. This allows the measure to take advantage of particular features of the host application which are useful for selecting objects of interest. Second, the processes of removing the old windows, and creating new ones of appropriate size (balancing the user's inputs and what is physically possible), is encapsulated in one high-level function, setWindowToWallRatio. Because this method is written, tested, and documented as part of the SDK, the user can simply use it without worrying about implementation details. Additional SDK methods and functions can be discovered for other tasks by browsing the building energy model documentation (Ball, Benne et al. 2012).



Credit: David Goldwasser/NREL

Figure 3: Dynamically generated input dialog for a "Set Window to Wall Ratio" measure

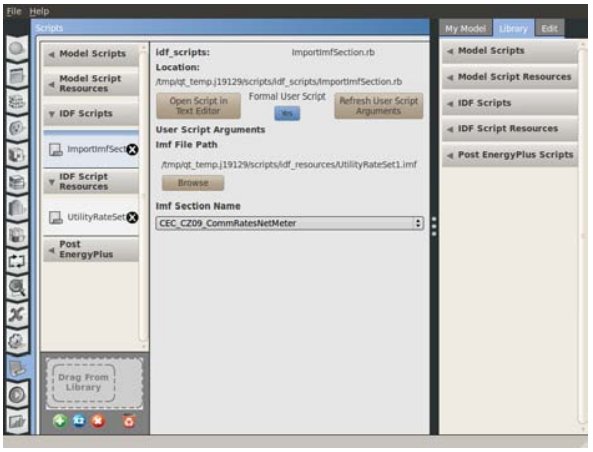
Noninteractive Measures

The interactive measures can be used to automate certain portions of a user's workflow; however, this method of running measures requires user input each time the script is invoked. In contrast, the OpenStudio application can run the same measure many times with the same set of arguments. To use this noninteractive mode, the user fills out the input dialog once and then saves the argument values for future use. The window to wall ratio script example can be run in this noninteractive mode with minor modifications that use a different mechanism for selecting the appropriate surfaces. In this section we will focus on another example.

Like the OpenStudio SketchUp plug-in, the OpenStudio application is built on the OpenStudio SDK; both applications use the run manager library to manage simulation workflows. The OpenStudio run manager library defines a simulation workflow as a tree of jobs in which the output of each job is available as input to any child jobs. The jobs in the basic workflow that the OpenStudio application uses to perform simulations are:

- ModelToIdf: Translate the OpenStudio model (OSM) into an EnergyPlus input data file (IDF).
- ExpandObjects: Call the EnergyPlus auxiliary tool ExpandObjects if necessary.
- EnergyPlus: Run an EnergyPlus simulation.

The scripts tab of the OpenStudio application, shown in *Figure 4*, allows users to call scripts (measures or free-form) at three points in the simulation workflow. Model scripts are run before the ModelToIdf job. At this point in the workflow, the building model library functionality may be applied to the input model. IDF scripts are run after the ExpandObjects job. At this point in the workflow, the input model has already been converted to EnergyPlus IDF and building model functionality cannot be used. The IDF file can, however, be modified using low-level data manipulation methods in the OpenStudio SDK. Finally, Post-EnergyPlus scripts are run after the EnergyPlus simulation, and may be used to generate reports or other outputs from the final OSM model, final IDF model, or the EnergyPlus simulation results stored in SQLite format (eplusout.sql).



Credit: Elaine Hale/NREL

Figure 4: OpenStudio application scripts tab

The particular script shown in *Figure 4*, ImportImfSection.rb, allows users to access their legacy EnergyPlus input macro files (IMFs). This feature is useful to advanced users because it allows them to work on their models in the OpenStudio format using the OpenStudio GUI tools, and directly access EnergyPlus features not yet available in the OpenStudio model.

The motivating example for this section is the need to include utility rate information in the energy simulation. The OpenStudio application does not yet have an interface to enter utility rate information, but many users have utility rate models in IDF or IMF format. To incorporate these utility rates in an OpenStudio model

simulation, the user drags the "Import IMF Section" measure into the model from the application's script library. The user then presses the button labeled "Refresh User Script Arguments". This action dynamically generates input dialogs that the user can use to set arguments for the script. Unlike the previous interactive example, the arguments for the measure are now stored for later use. Each time the model is simulated, this script is executed with the saved arguments. The code this particular script runs to import objects from a named section in an EnergyPlus IMF into the EnergyPlus IDF, with error checking code removed for brevity, is:

```
def run(workspace, runner, arguments)
 imf_file_path = arguments["imf_file_path"]
 imf_section_name =
     arguments["imf_section_name"]

 imfFile = ImfFile::load(
     imf_file_path.valueAsPath,
     "EnergyPlus".to_IddFileType)
     imfFile = imfFile.get

 objects = imfFile.section(
     imf_section_name.valueAsString)

 wsObjects = workspace.addObjects(objects)
end
```

FREE-FORM SCRIPTING APPLICATIONS

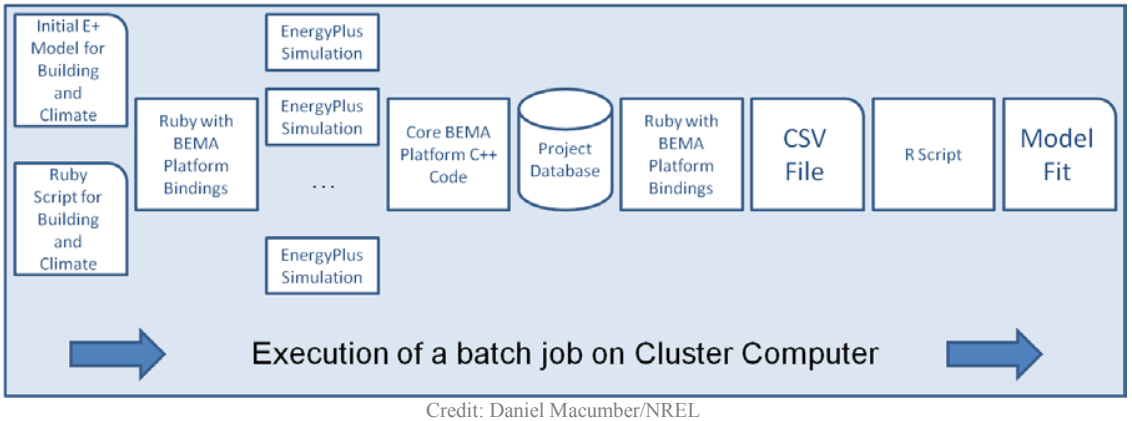
The examples in the previous section show the potential for measures to extend the capability of OpenStudio GUI applications without touching the underlying application code. More advanced users may require solutions that do not fit into the measure script paradigm. They may thus opt to write free-form Ruby scripts that use the OpenStudio scripting environment. These scripts run outside the OpenStudio GUI tools and are designed to meet specific user needs.

179D Case Study

Figure 5 depicts the workflow used to construct a Web tool for prescreening buildings to see if they might be eligible for the Federal 179D tax deduction. This analysis has now been run twice, once in spring 2011, when this analysis was the first large-scale project to use the scripting platform; and again in winter 2012.

In both cases, about 250,000 simulations were run on cluster resources in the span of one to two weeks, after about a month spent developing, testing, and debugging the scripts set up to do the analysis. The scripting platform was key to enabling such a large-scale study and resulting Web application to be set up and run in such a short time by two to three full-time modeler-developers plus one Web developer. Scripts developed on Windows machines were easily adjusted to also work on a Linux cluster; energy simulation data were fit to regression models using another freely available software program, R, with communications between the energy models, R, and the website done using comma separated value (CSV) files (Deru, Griffith et al. 2012).

The study was rerun in 2012, primarily to add variable speed fans and heat pumps, which could not be included the first time because of severe time constraints. Although the scripts were updated to catch up with changes to the SDK during the intervening year, most of the modeling logic was directly preserved and reused. Because the initial study was completed before the building energy model was sufficiently built out, this work uses EnergyPlus IDF (specifically the DOE Commercial Reference Buildings (Deru, Field et al. 2011)) as the primary input. Then, 576 distinct analyses are constructed, run, and evaluated, one for each combination of building type (12), climate zone (16), and system type (envelope, lighting, and HVAC).



Credit: Daniel Macumber/NREL

Figure 5: Workflow for 179D analysis

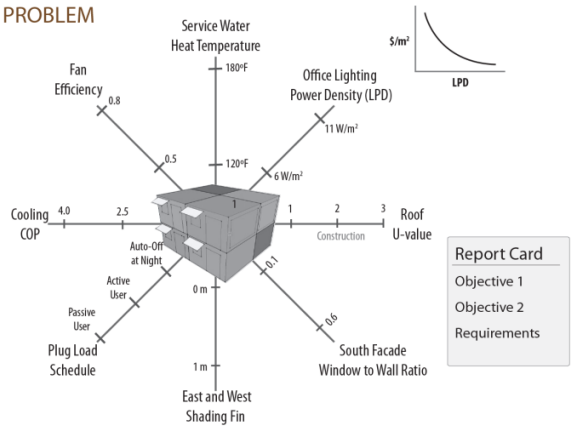
For each building and system type, a problem is defined from discrete variables whose values are either null (which leaves the seed model unchanged), or defined by a Ruby script with some particular argument values set ahead of time. Examples of script arguments include heating coil efficiency, roof U-factor, and lighting power density multiplier. Each simulation workflow, in addition to the normal simulation steps, is concluded by a custom postprocess, also written as a Ruby script. Each analysis runs a full mesh of all the options using our full factorial design of experiments algorithm.

Once the analysis is run, another set of Ruby scripts extracts results from the database populated by the analysis driver, and places them in a CSV format. These files are then consumed by R scripts that calculate one data fit per CSV file. The data fits are then compiled and pushed to the Web application for use in estimating proposed building performance relative to the 179D requirements. Quality control data are also collected and evaluated along the way, which routinely prompts tweaks in one or more scripts, and a partial rerunning of the analysis.

FUTURE WORK

As the OpenStudio SDK matures, more capability will become available for use in measures and free-form scripting. As this functionality increases, we expect that more users will become familiar with the OpenStudio scripting environment and begin to author their own measures and free-form scripts. The next transformative step will then be to add measure scripts to the content available from the online Building Component Library (BCL), currently hosted at http://bcl.nrel.gov. When this is complete, users will be able to search for measures to download and apply using the OpenStudio GUI tools. In this way, one user can benefit from scripts written by another. Similarly, we also envision users downloading measures to compose BEMA problem definitions, which they can then apply to their own seed models.

The problem formulation concept is depicted in *Figure 6*. In this example, the user's model is placed in an eight-dimensional parameter space. Each axis is defined by a measure, or set of measures, downloaded from the BCL. Each point on each axis corresponds to a particular discrete measure or a particular input argument value. Once the parameter space has been defined, the user can choose to run one of OpenStudio's built-in algorithms or one available from the DAKOTA library (Adams, Bohnhoff et al. 2009). In this way, many more users will be able to perform advanced, customized analyses.



Credit: Marjorie Schott and Elaine Hale/NREL

Figure 6: Measure scripts used to compose a parameter space

CONCLUSION

Building energy modelers and analysts typically use several tools. Most projects use a mix of GUIs, spreadsheets, and text editors. Expert users may write custom scripts to implement portions of their workflow. These custom solutions are usually written from scratch and result in significant duplication of effort across the BEMA community.

In this paper we demonstrated the use of a BEMA software platform accessed through the Ruby scripting language. Measures are scripts with a defined interface that allows them to be used in several contexts. Users can also write free-form scripts using the portions of the OpenStudio SDK exposed to the scripting environment.

Both types of scripts give the user more flexibility and power than are typically provided by a GUI, and minimize the risks typically associated with software development. Because the platform is built on modular code for modeling, algorithm-driven model generation, data management, and run management, the amount of code that must be written to do a particular task is greatly reduced. Because the core code is written in C++, documented, tested, and freely available, the scripts run quickly and are widely shareable. Finally, the available functionality will grow over time as functionality is added to the OpenStudio project, reducing duplicate efforts in the BEMA community.

In addition to adding functionality to the underlying code base and enhancing application support of measures, planned work includes extending an online database of building energy data, the BCL, to contain measures. This will allow users to easily search for and apply measures to their own models. Eventually, users will be able to construct customized parameter spaces using downloaded measures, and explore this parameter

space using their own seed models along with advanced optimization, uncertainty quantification, and other analysis routines.

ACKNOWLEDGMENT

This work was financially supported by the U.S. Department of Energy Building Technologies Program, and by the California Energy Commission (Task No. BEC7.1335 and WW2D.1000, respectively). The authors are greatly indebted to our sponsors at those organizations, and also to our colleagues at the National Renewable Energy Laboratory. OpenStudio would not be what it is today without Brian Ball, Larry Brackney, Luigi Gentile Polese, Nick Long, Marjorie Schott, Alex Swindler, Jason Turner, and Evan Weaver. The 179D tax deduction case study work was largely done by Brent Griffith, Matt Leach, Eric Bonnema, Katherine Fleming, and Michael Deru, in addition to a subset of the authors.

REFERENCES

Adams, B. M., W. J. Bohnhoff, K. R. Dalbey, J. P. Eddy, M. S. Eldred, D. M. Gay, K. Haskell, P. D. Hough and L. P. Swiler (2009). "DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 5.0 User's Manual." Sandia Technical Report SAND2010-2183. Sandia National Laboratories.

Ball, B., K. Benne, K. Fleming, L. Gentile Polese, D. Goldwasser, B. Griffith, R. Guglielmetti, E. Hale, N. Long, D. Macumber, M. Schott, A. Swindler, J. Turner, E. Weaver, K. Gowri and R. Hitchcock (2012). OpenStudio SDK Documentation, Version 0.7.0. http://openstudio.nrel.gov/sdk-documentation.

Beazley, D., L. Ballabio, W. Fulton, M. Gossage, M. Koppe, J. Lenz, M. Matus, J. Stewart, A. Yerkes, S. Yoshiki, S. Singhi, X. Delacour and O. Betts (2009). Simplified Wrapper and Interface Generator (SWIG). http://www.swig.org.

Deru, M., K. Field, D. Studer, K. Benne, B. Griffith, P. Torcellini, B. Liu, M. Halverson, D. Winiarski, M. Rosenberg, M. Yazdanian, J. Huang and D. Crawley (2011). "U.S. Department of Energy Commercial Reference Building Models of the National Building Stock." NREL/TP-5500-46861. National Renewable Energy Laboratory, Golden, Colorado.

Deru, M., B. Griffith, M. Leach, E. Bonnema and E. Hale (2012). 179D Easy Calculator Technical Support Document. Golden, CO, National Renewable Energy Laboratory.

DOE (2011). EnergyPlus Energy Simulation Software, Version 7.0. U.S. Department of Energy, Washington, D.C. http://www.eere.energy.gov/buildings/energyplus/.

Flager, F., B. Welle, P. Bansal, G. Soremekun and J. Haymaker (2009). "Multidisciplinary process integration and design optimization of a classroom building." Journal of Information Technology in Construction **14**: 595–612.

NREL (2012). BEopt, 1.2. http://beopt.nrel.gov.

van Heesch, D. (2012). Doxygen, Version 1.8.0. http://www.stack.nl/~dimitri/doxygen.

Ward, G. (2011). Radiance: Synthetic Imaging System, Version 4.1. Lawrence Berkeley National Laboratory, Berkeley, California. http://radsite.lbl.gov/radiance/HOME.html.

Wetter, M. (2001). GenOpt -- A Generic Optimization Program. Proceedings of IBPSA's Building Simulation 2001 Conference. Rio de Janeiro, Brazil.

NOMENCLATURE

BEMA	building energy modeling and analysis
CSV	comma separated value
GUI	graphical user interface
IDF	(EnergyPlus) input data file
IMF	(EnergyPlus) input macro file
OSM	OpenStudio model
SWIG	Simplified Wrapper and Interface Generator