OPPORTUNISTIC SOFTWARE DEVELOPMENT

Robert C. Sonderegger
SRC Systems, Inc.
1300 Clay Street, Suite 850
Oakland, CA 94612

ABSTRACT

Modem, Message-Based operating systems with Graphic User Interfaces have spurred tremendous advances in software development. Usability, connectivity, and transparency of software has increased, as has the transferability of the skills learned in one application to another.

Building Simulation software developers have ample opportunity to profit from the advanced technology that is now available, such as sophisticated edit controls, helpful concepts such as Cue Help and Wizards, and the powerful modularizations made possible by Dynamic Link Libraries.

A number of these concepts are discussed in detail and applications to building energy simulation are presented. To embrace these applications will require a change on the part of building simulation software developers away from vertical integration (developing HVAC calculations *and* all data 1/0 aspects), toward opportunistic integration of commercially available modules for all data 1/0 to better be able to focus on the issues pertaining to HVAC.

INTRODUCTION

Users of the Apple Macintosh and of computers running Microsoft Windows and OS/2 have for some time enjoyed powerful and consistent user interfaces in office applications such as wordprocessing, spreadsheets, and database systems. Many users have grown accustomed to switching among applications, sharing data, and helpful *Wizards* to guide them through unfamiliar territory. To switch from one of the popular *Office* Suites to a traditional building energy analysis program written for DOS is like crawling after you've grown accustomed to flying.

The reasons usually given for this disparity are multiple: the complexity inherent to building energy analysis, the small market size, and the need for detailed simulation. After all, what does simulating a complex control strategy in a building have in common with managing a sales database or writing a product presentation?

This paper reviews the concepts and paradigms that have become standard in modern Graphic User Interfaces (GUI) and examines how they have been or should be adapted to the tasks of building energy simulation. The list of GUI concepts that will be discussed are:

- Hierarchical outlines (e.g., the tree-like connections used by Windows' File Manager);
- Libraries;
- Templates and Wizards (pre-made formats to guide you when starting new work);
- Advisors (smart programs checking on your work);
- Cue Help (context-sensitive instructions that tell you not *what*, but *how to*)
- Dynamic Link Libraries (specialized program modules linked at run time);

HIERARCHICAL OUTLINES

Much research has gone into better modeling of heat transfer processes in building components. Detailed equations govern the latent and sensible heat transfer between a cooling coil and the air streaming past it. The emphasis on detailed modeling at the component level frequently masks the difficulty of handling the complexity inherent to buildings as a heterogeneous collection of components. Many objects of different types (zones, systems, components) are interconnected by means of a building-specific, logical and physical structure. Each *object* is characterized by input and output data (surface areas, control strategies, thermal properties) and by *methods* that describe the relationships between subsets of such data. When creating or editing a building, the user of a simulation tool may modify the *connections* among building objects (e.g., which zone is served by which HVAC system), the *data* describing individual objects, or the *parameters* specific to the methods. To a large extent, buildings are simply Relational Databases. An important subclass of a Relational Database structure is the hierarchical or tree structure, ubiquitous today as the file structure on

most data storage media. For example, the *root directory* on a logical drive may contain one or more subdirectories (or *folders*, in Mac-speak), each of which may contain on or more subdirectories, and so on.

The elements of a building can also be viewed as a hierarchical structure, with the *Building* proper at the top, containing one or more *Plant* items, each of which may contain one or more *Systems*, which are made of *Zones* composed of a number of *Walls*, *Roofs*, *Floors*, etc. But if components in buildings are like files on a disk, why not avail ourselves of the graphical representation and the tools commonly associated with files? File managers are ubiquitous in GUI environments and their function readily adaptable to buildings. An example of the hierarchical representation of a building is shown in Figure 1. There are many advantages to the tree approach:

1. Representation and navigation of the building data structure are combined in one intuitive picture;
2. Skills and concepts that the user already knows from file management are directly transferrable;
3. Many commercial software modules are available to implement this type of representation.
4. Readily available editing paradigms for use in copying and pasting building elements: for example, a zone and all of its elements are copied *en bloc* from one system to another in Figure 3 using the popular *drag-and-drop* operation.

LIBRARIES

Most knowledge-based programs feature some sort of library, as a collection of individual components. For buildings, a library can contain large selections of such components as window types, lighting fixtures, and heating or cooling equipment. In addition to physical components, abstract objects such as operating schedules, thermostat schedules, utility rate tariffs, and economic assumptions are also useful as library contents.

Unfortunately, the more detailed a library, the harder becomes the task of finding the right building component. To minimize this inconvenience, a good *Query* facility is essential. Again, examples of libraries with related queries abound in commercial software. Figure 4 shows an example library of ready-made building components in a popular architectural drawing program for the home market. To select from a fixture library, the user picks *function* (toilets, sinks, appliances, utility), *style* (water heater, furnace, circular stairs, fireplace), and *sizes* the chosen item either by entering dimensions or by stretching or shrinking the object once dropped into the plan.

TEMPLATES

Sometimes referred to as structured libraries, templates combine a subset of library components in a way meaningful to the application. In wordprocessors, for example, when starting a new document the user may select from a list of templates, such as *Business letter*, *Agenda*, *Report*, *Proposal*, etc. By selecting one of these choices the user causes formatting, styles, preset boilerplate phrases, and other esthetic and organizational details of the document to be set in a manner appropriate to the type of document to be prepared. Applied to buildings, an obvious use of templates is to specify building use: *Small Office, Large Office, School, Hospital, Place of Worship, Industrial Facility*, etc.

After selecting one of these templates the user is presented with a "generic" building of this type, with some elements preset according to the selected building type:

- Topology of plant, systems, and zones
- Types of HVAC system, lighting, and other internal gains
- Building type-specific defaults (e.g., W/sqft of lighting, cfm/person of ventilation, etc.)
- Occupancy schedules

While the user still has to enter data specific to the project, doing so on a skeleton building already pre-configured speeds up input and minimizes accidental omission of crucial inputs.

WIZARDS

Similar to Templates, Wizards are invoked whenever a new document is to be prepared. Where Templates are limited to passive presentation of the elements needed for a specific task, from drafting a business letter to configuring an office building, Wizards add an active element of helping the user through some key decisions. When selecting a *Business Letter Wizard*, for example, the user may be prompted for Address, Subject (e.g., *Thank You*, *Promotion* or *Dunning Letter*), Tone (*Collegial, Formal,* or

Diplomatic), and Content (in the form of directed queries, specific to the *Subject* selected before). Once this task is over, the program addresses, formats, and phrases the business letter. The user is left to edit and fine-tune the result.

Applied to buildings, a *Office Building Wizard* may ask the user about:

- Type(s) of Building Usage;
- Floor Area (possibly by building usage), number of stories, and height;
- City or Climate Zone;
- Typical Defaults (W/sqft of internal gains, cfm/person of ventilation, thermostat settings);
- Type(s) of HVAC system(s);
- Principal zone layout and envelope types.

Based on the user's answers, the program constructs a generic building conforming to these specifications, leaving the user to fine-tune the specifications where needed.

The Wizard capability frees a creative designer from getting too bogged down into details too quickly. If the question is "What sort of energy use can we expect from a typical school in Peoria," a well designed Wizard capability overlaid on a simulation program may be the fastest way to an answer. Whereas looking up the appropriate figure from a energy use survey[1] may give a more immediate answer to the same question, using the Wizard feature to drive a detailed building energy analysis program enables a seamless transition over time from the generic questions at the conceptual design stage to the detailed specifications required by a complete project.

ADVISORS

Where Wizards are front-end assistants that fetch necessary components and put them together in a meaningful way, Advisors can be invoked to check your work and suggest better alternatives.

Many home-finance programs feature advisors which volunteer advice, from the mundane to the thoughtful, from steering a user with a credit card habit to cheaper alternatives to offering year-end tax tips based on the user's particular financial situation.

For building programs, the most simple-minded type of advisor is an *Input Check* where common input errors like missing walls, overdefined schedules, and undersized equipment are spotted on demand. This is the equivalent of *Spellchecking* a text document.

At the next level of sophistication is a *Configuration Check* where the inconsistency between, say, a VAV system and a constant volume air handler is flagged, or the missing cooling tower fan in a water-cooled chiller plant. This is the equivalent of having your wordprocessor doing a *Grammar* check.

At the highest level to date is a *Design Advisor*[2] which not only flags standards out of compliance but suggests ways to improve building performance or meet the required standards. For standards compliance, see the section on *Dynamic Link Libraries* below.

CUE HELP

The traditional on-line help system is a relatively passive affair that answers whatever you may ask, but does not tell you what questions to ask. It may tell you all about part-load coefficients of a heat pump, while you still have not figured out how to add a heat pump to a zone in the first place.

Enter *Cue Help*, also dubbed *How To* help, an excellent example of which can be found in Microsoft Access. Cue Help is different from traditional help in that it is task-oriented. Where traditional help gives you extensive information on each command available, Cue Help leads you through just those commands you need to accomplish a given task. The Cue Help window is always "on top" and stays visible no matter how deep you get into a command tree. The learning curve is shorter because you immediately use what you learn.

When it comes to building simulation, a well-designed Cue Help system has many applications:

- Learning a software tool or refreshing usage proficiency after a hiatus between projects;
- Learning building energy analysis on-the-job, so to speak;
- Helping the user steer through design alternatives in a context-sensitive manner.

Cue help opens new vistas on the integration of the heretofore distinct aspects of education and practice

[1] NBECS: Commercial Buildings Consumption and Expenditures, Energy Information Administration (1983).

[2] See, for example, the *Energy Design Advisor* under development at Lawrence Berkeley Laboratory.

in building energy analysis. With modern software developers kits, the skills required to implement Cue Help are 10% programming and 90% writing like those required of any textbook author.

DYNAMIC LINK LIBRARIES

A Dynamic Link Library (DLL) is a module of compiled, executable code that is linked at run-time with the main program (EXE). A DLL communicates with the EXE program or with other DLL modules thorough a standardized interface. For all intents and purposes, the interface of a DLL is little more than an indexed list of subroutines, each with an optional list of arguments. The DLL source code looks like any other module written in one of the many languages that support DLL's, plus a few special statements declaring this module to be a DLL. As a library, the DLL lacks a "Main Program;" it has no beginning or end, just a lot of subroutines waiting to be called by other modules or the main program. The routines *published* by the DLL[1] are part of a standard, language-interpreted interface. That is, a main program written in C++ can reference subroutines written in Fortran, Pascal, or Basic compiled into a DLL.

Run-time linking and language-independent interface make DLL's a powerful tool of software developers. Among the many uses of DLL's are:

- Distribution of graphics engines or report engines: Software developer "A" develops a simulation package that uses a graphics engine by Software developer "B" and a report engine by Software developer "C".
- Different applications developed by one software developer may all need the same database management and interface management routines: an excellent use for one or more DLL's common to all applications of that vendor.
- Database servers: many commercial applications read and write data from files of different formats and located on different storage media, sometimes even on different computers linked by wide area networks; DLL's are an ideal means to hide all this irrelevant detail from the user of the data, providing instead a consistent, standardized interface to the calling program.
- Encapsulated Data Interchange Formats: hiding the details of a common data interchange format from different client applications by encapsulating them into a DLL server.

LEGACY CODE[2] PRESERVATION

Many programs designed for DOS are in need of new interfaces. DLL's provide an excellent way to accomplish this task. Old algorithms and logic that have been stable for some time are left untouched, while the interface is replaced with a thin layer of subroutines accessible from other programs through the DLL interface. It's like pulling a 380 in^3 V-8 from an old car, fitting it with new engine mounts, ready for installation into a late model body that meets all current safety standards.

STANDARDS COMPLIANCE

The performance-based energy budget approach contained in both ASHRAE Standard-90 and California's Title-24 require running an actual design against a reference design. The actual design passes if its energy use or operating cost is less than that of the reference design. The approach sounds deceptively simple, but beware of the fine print.

First, certain exceptions apply to what are permissible inputs for the proposed design. For example, minimum levels of insulation are required for most envelope elements regardless of the overall design performance.

Second, generating the reference design from the proposed design is a complex process subject to many rules which not infrequently leave logical loopholes. Once face-to-face with such a loophole, what is a programmer to do? Certifying the compliance shell of third-party software is a non-trivial task of a standards committee. As the standards grow in complexity, so does the software to support it and so do development costs.

Currently, portions of the ASHRAE Standard are available in software form[3] as a stand-alone compliance checking tool. The user enters a number of inputs about the proposed building and the

[1] The DLL code may include other, unpublished routines for its own internal use. However, these are not visible to any outside module or executable program.

[2] A term coined by clever managers to protect the egos of programmers. The term designates code that *has been around longer than the most senior programmer* and is therefore better left alone.

[3] See for example: American Society of Heating, Refrigerating and Air-Conditioning Engineers, ASHRAE Standard-90 Compliance Calculation Programs ENVSTD and LTGSTD.

software issues a Pass/Fail result. Additional partial results showing the "thinking" behind the Pass/Fail result may also become available. Independent software developers have to replicate the pass/fail functionality in their own code if they want to offer a broader range of applicability of their software and avoid the inconvenience to their users of using different software to do related tasks and of re-entering some data twice.

In this author's opinion, DLL's provide a better way to accomplish this goal without the cost of replicating the complex compliance logic. Together with the printed standard, the compliance checking software equivalent should be published as a DLL. A stand-alone "Demonstration Interface," physically separate from the DLL but mated to it, could be distributed at the same time for illustration purposes. A software vendor seeking to add standards compliance to its energy analysis program simply has to add a query to the DLL's *Pass/Fail* subroutine. Of course, this subroutine will feature a large number of parameters to be supplied by the querying energy analysis program, essentially, all the inputs that would be entered by hand in the demonstration interface.

In addition to the compliance DLL's Pass/Fail subroutine there will be other subroutines supporting the definition of the reference design, such as tradeoff functions and reference design generators.

The key idea is that through run-time linking and language independence, DLL's are the ideal way to provide a neat division of labor between "Algorithm development" and "Interface design" (see Figure 5). Teams of scientists, engineers and software designers can each focus on what they do best using the DLL interface standard to both communicate and to delimit areas of responsibility.

ENERGY ANALYSIS AND CAD

The interaction between CAD systems and energy analysis programs is another area of application of DLL's and also an illustration of another buzzword -- the *Client-Server* paradigm. To a designer using a CAD system, an energy analysis program is mostly a server of energy calculations -- to the HVAC engineer client a CAD system is simply a convenient way to enter or review the building and mechanical systems plans. Either view is possible if the CAD functionality or energy analysis are put into a DLL form.

CONCLUSION

A brave new world of software technology is upon us. There is tremendous opportunity to use readily available Software Developers' Kits (SDK), DLL libraries, Resource Workshops, and more, all to bring the functionality and user interface of building energy simulation programs to the standards that today's users have come to expect.

With the event of powerful, message-based, operating systems with graphic user interfaces, the persona of the lone programmer hacking long nights is rapidly giving way to the well-read, productivity-minded software integrator. If a small industry like building energy simulation is to keep up with today's expectations of usability, connectivity, and transparency of software, it is imperative for its software developers to change from lone inventors to technological opportunists.

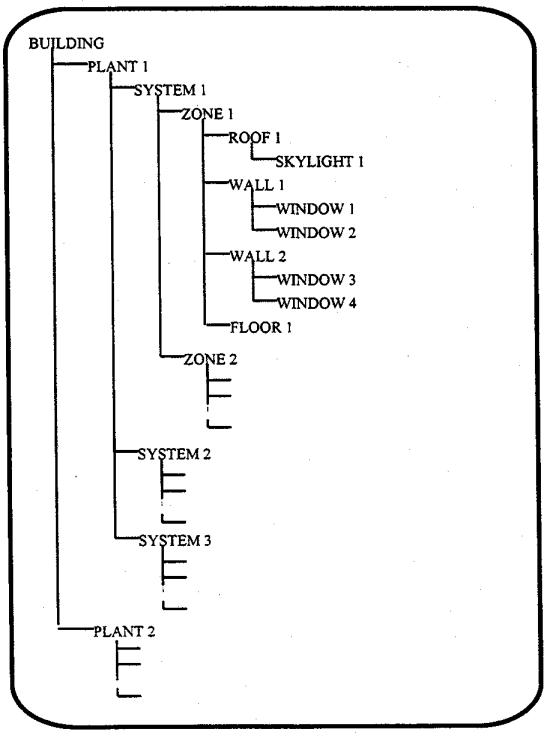


Figure 1: Tree representation of Building Components

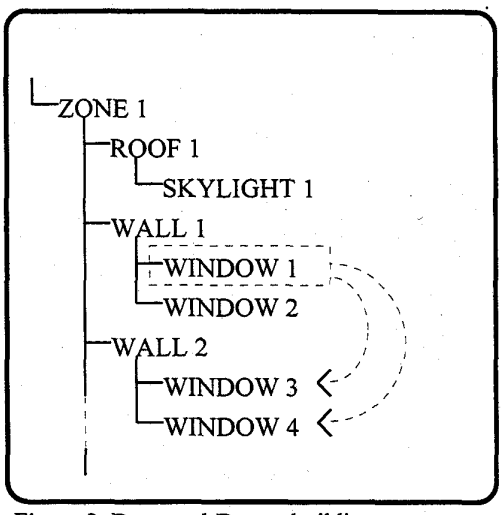


Figure 2: Drag-and-Drop a building component

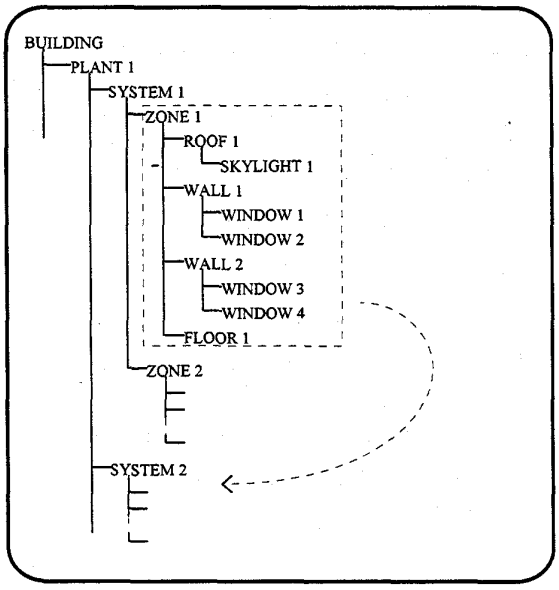


Figure 3: Drag-and-Drop of building components

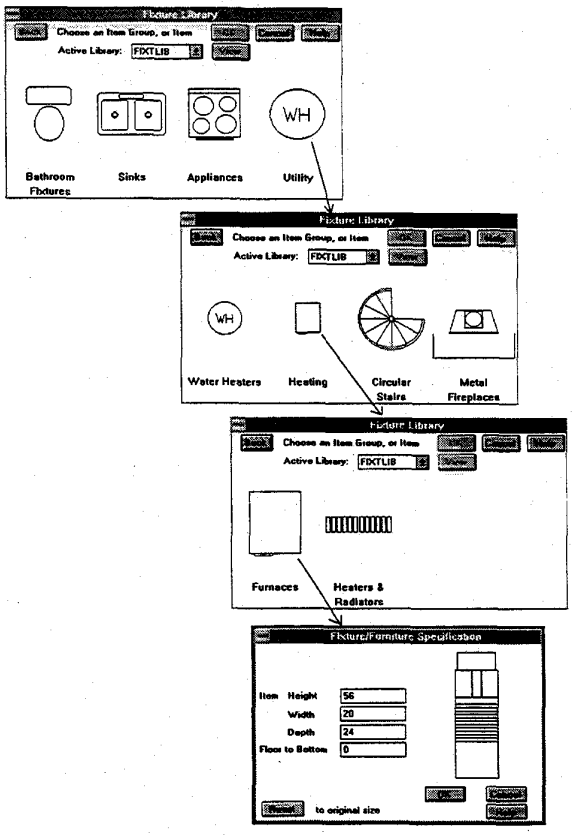
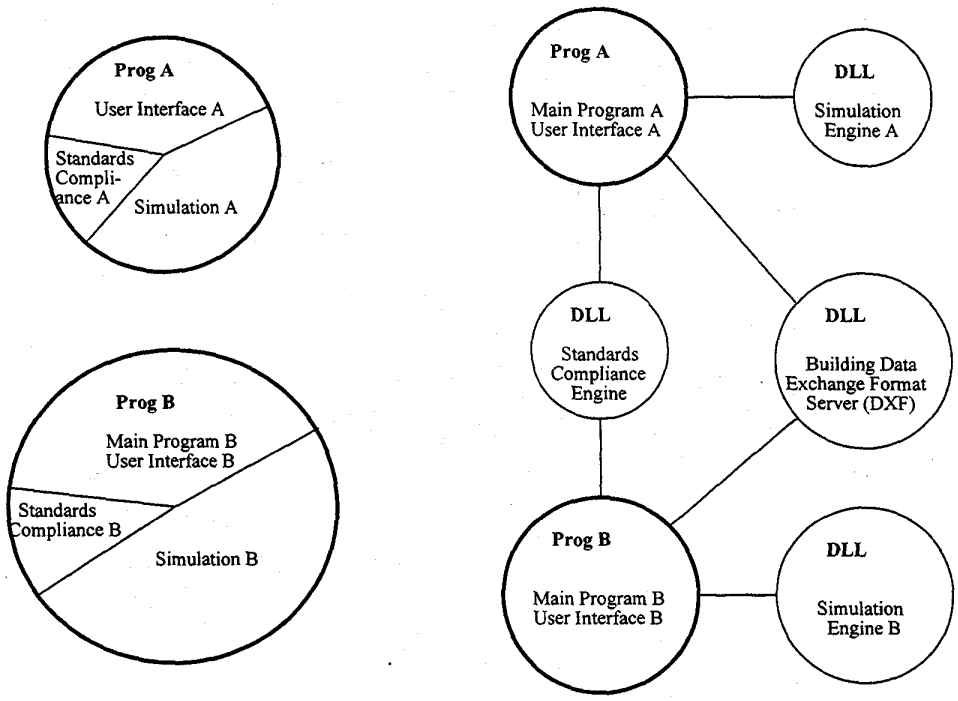


Figure 4: Library of construction elements in home architecture drawing program



Traditional S/W Architecture

Modern S/W Architecture

Figure 5: Examples of Dynamic Link Libraries used for Building Energy Analysis