Building Performance Simulation in a Commercial Software Environment

by

Robert C. Sonderegger
Sr. VP, Research & Development
Morgan Systems Corporation
2560 Ninth Street, Suite 211
Berkeley, CA 94710 USA

ABSTRACT

Much effort has been devoted over the years to advance Building Performance Simulation (BPS) by improving algorithms and by extending the simulation domain to daylighting, acoustics, and indoor air quality. Yet in several recent relevant ASHRAE forums many attendees asked for transparency, useability, and flexibility of computer programs.
The issues of flexibility, transparency, and ease of use are categories commonly associated with the user interface. Moreover, they relate fundamentally to software architecture. A good user interface is the foundation of a program, not a finishing touch. While seemingly unrelated to BPS, the tedious "bookkeeping" tasks of data entry, data management, and reporting occupy a major part of the user's time and should similarly be near the top of the software developer's priorities. How well these tasks are handled is the key to usability, flexibility, and transparency of a BPS package.
The budgets of most software developers in the area of BPS are limited and usually do not allow the type of development effort necessary to deliver the quality of user interface today's more sophisticated users demand. This paper describes an approach that relies heavily on commercial software packages, such as database managers, spreadsheets, and wordprocessors. Besides yielding more product bang for the development buck, this approach goes a long way to making BPS more flexible, transparent, and less onerous for the user.
The software architecture of a typical BPS system is presented with detailed indications as to which tasks can be handled by commercial software packages, and which are likely to require programming by the specialist.
The capabilities of several types of popular commercial software packages are presented as they might apply to the needs of BPS: 1) Database managers can maintain building component libraries, such as records of envelope assemblies, equipment performance specifications, HVAC control settings; 2) Wordprocessors with merge and sub-document inclusion capabilities can be harnessed to become automatic report generators; 3) Spreadsheets and graphics packages can represent simulation results in graphic form.
In our opinion, only the overall integration of all software components and the real, numbercrunching aspect of Building Performance Simulation should be handled by specialized routines written in high-level code. The challenge to the software developer is the successful integration of commercial software with in-house developments. To the user, well integrated systems look simple, uniform, and provide the required flexibility and transparency.

INTRODUCTION

Today's software user in the field of Building Performance Simulation (BPS) is increasingly sophisticated. No longer satisfied by tinkering with homegrown BASIC programs, our target user has come to expect the type of interface offered by popular software, such as spreadsheets and wordprocessors. There is more to a good user interface than the "look and feel" of a software product -- at more than one recent ASHRAE forums, the main concerns voiced with respect to BPS computer programs were:

- *Flexibility*, i.e. being able to model the widest spectrum and mix of technologies, even beyond what the software designer had in mind;

- *Transparency*, i.e. being able to verify by manual calculations what the software is doing;

- *Ease of Use*, i.e. a consistent help system, uniform menus, random access to all inputs, and a "look and feel" similar to what is becoming common practice in standard commercial software.

To this list should we add some needs suggested by our own experience:

- *Connectivity*, i.e allowing the user to transfer data from one application program to another;

- *Breadth*: to many users, the ability to model the greatest variety of technologies at a uniform level of detail is more important than the ability to model a few technologies in the greatest detail.

- *Comprehensiveness*, related to Breadth: for many applications, BPS software must be able to model all aspects affecting BPS regardless of the user's focus, e.g., solar gains through windows should be adequately modeled even though a particular user may only be interested in comparing a centrifugal chiller to an integrated Ice Storage unit. Modern utility rates are rapidly increasing in complexity, with Time-of-Use charges, block rates with demand-dependent block limits, overall bill limits in $/kWh, etc. Since ultimately the end user of most BPS (the client) will require accurate economics, careful engineering analysis coupled with using only average $/kWh prices constitutes a mismatch of efforts. The building must be modeled as a whole, including such extraneous issues as exterior lights, cogeneration plant, and vacation schedules. Moreover, utility rate modeling must become part of BPS.

- *Downward Version Compatibility*: Most BPS software is obsolete the day it is released: New technologies become available (e.g., household-sized cogeneration units) and old technologies are repositioned (e.g., gas-engine driven chillers), and it doesn't take long for a user to ask for a corresponding update. While some users may welcome added capabilities, all users demand downward version compatibility, i.e. the ability to re-use his or her past work in defining buildings in the new version without major modifications.

There are many possible strategies to address these concerns in developing BPS software. Our approach is to utilize existing commercial software for many aspects of software design. This paper describes this approach and its advantages and drawbacks.

COMMERCIAL SOFTWARE

For lack of a better term, we use "commercial software" to define off the shelf, popular software packages of general (horizontal) application, such as database managers, wordprocessors, and spreadsheets.

Commercial software has come a long way from the pre-IBM/PC and pre-Macintosh days. Among the features relevant to our discussion are:

- *Similar User Interface Style*: Like the gradual convergence in the early automobile industry toward common positions of accelerator, clutch and brake pedals, today's software menus are becoming more similar (sometimes to the point of arousing legal copyright battles).

- *Common Data Formats*: While many equivalent data formats exist, the general tendency of modern software is to provide the capability to read from and sometimes write to competing formats; even cross-application data format compatibility has become available enabling, for example, a Quattro[1] (spreadsheet) user to read and write dBASEIII+[2] (database) files.

- *Hardware Transparency*: Most modern software supports the current plethora of monochrome and color graphics cards and printers. Even data transfers across incompatible hardware (e.g. between IBM and Apple personal computers) are now available. The logical conclusion of this trend will eventually lead to "user transparency" where the user can fully concentrate on the application, without concern for the hardware or software involved.

Unfortunately, including these features greatly increases the "technological overhead" inherent to any software development. Even simple software requires provisions for printer and video card drivers, file format definitions, and, above all, documentation. This necessary overhead usually has little or nothing to do with the main purpose of the software, for example BPS. In our experience, this overhead and other aspects of the user interface account for up to two thirds of the entire program code.

One promising avenue to meet users' needs within a reasonable budget, is to integrate commercial software with specialized, application-specific code. Though this approach is not without drawbacks, there is considerable advantage to standing on the shoulders of giants.

What are the "nuggets" in commercial software that we can put to good use in BPS? What is the proper mix of in-house software development and software integration? To answer these questions we focus on three popular types of commercial software: Database Managers, Wordprocessors, and Electronic Spreadsheets.

In Fig. 1 we review schematically the major components of typical BPS software and where use of commercial software is most applicable.

DATABASE MANAGERS

Many objects inherent to BPS are database naturals: wall descriptions, equipment performance specifications, even control strategies. In a cooling equipment database, for example, each chiller is represented by a record whose fields might be capacity, efficiency, part load performance coefficients, etc.

Every software writer in the field of BPS business sooner or later must define his or her databases and decide on how to store the information -- either directly in the program code, or in external tables, or as direct user entries. In doing so, he or she will also have defined yet another database standard.

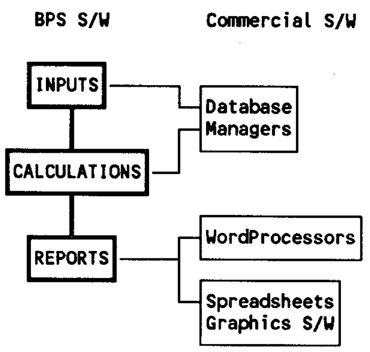


Fig. 1: Schematic representation of major components of typical BPS software and suggested interface points with commercial software.

At the cost of offending the experts, a brief definition is in order: databases can be thought of as large collections of objects of the same kind, e.g. mailing lists, stock quotations, or car specifications. A stack of resumes doesn't qualify as a database, unless all applicants write theirs according to the exact same format, i.e. Name, Date of Birth, Schools attended, Salary requirements, etc. This is an example of a database *structure*, where each *record* represents one applicant, and each item of information is one *field* in a record. The database structure is separate from the *file format*, e.g. whether the records and fields are stored on disk as ASCII characters or in a binary format, etc.

Popular and powerful DataBase management systems (DBMS) are available on the general software market. The main purpose of the DBMS is to allow the user to define database structures, to fill the databases with numbers and text, to manage the information, to delete unwanted records, or to sort by value or alphabetically (e.g., "give me all chillers with capacity greater than 50 Tons and efficiency better than 0.7 kW/Ton").

Generally, DBMS don't have very good calculation ability. For instance, they are not very efficient at manipulating data as required by BPS -- integrations, matrix inversions, polynomial evaluations, etc. At this time these algorithms are better implemented through traditional programming. Yet DBMS shine in managing large amounts of information. The avenue we recommend combines the best of both worlds. We manage all data with a popular DBMS (e.g., dBASEIII+, dBASEIV, or R:BASE), write the algorithms that require this data in a high-level language (e.g. Pascal, C, or Fortran), and connect the two by building an interface that can read and write DBMS files directly.

There are at least three different software architectures to accomplish this:

- Write an overall data management *shell* using the DBMS' "command language" which can then call the special-purpose BPS routines written in a high-level language;

- Write the main *shell* in the high-level language and *execute* a version of the DBMS as a subroutine, customized to handle the BPS application's needs and data.

- Write the main *shell* in the high-level language and add *application drivers* that can read from and write to DBMS files;

In our newest product we are using the last of the strategies described. We have organized the data required for BPS along the following categories (see Fig. 2):

BPS LIBRARIES
General Info
Walls, Roofs, Floors
Windows, Shading
Lighting & Equipment
Fans, Pumps, Motors
Ventilation
Distribution
Cooling Plant
Heating Plant
Cogeneration
Operating Schedules
Energy Rates

Fig. 2: Databases required for BPS.

Each line in the figure represents one or more databases with a unique structure, each describing one class of *objects*. Objects are divided into *devices* (lighting fixtures, chillers, etc.), *components* (walls, windows, etc.), *assumptions* (Ventilation controls, Operating Schedules, Energy Rates, etc.). In total, we identified 26 databases.

To add new records (e.g., a new chiller) or edit or delete existing records we created a total of 58 *screens* in the DBMS command language. There are more screens than unique databases because many objects share the same database structure (e.g., walls, floors, roofs, windows, and shading devices all share the same *Envelope* database structure). For each "individual" screen there is a companion "collective" screen to enter, edit, or delete many devices at the same time. Both types of screens are reproduced side-by-side in Fig. 3, using the Lighting database as an example.

LIGHTING FIXTURES	
Name: 180LPS	
Usage (Res/Ind/Com): C	
180 W Low Pressure Sodium	
Total Capacity (W):	220
Lamp Capacity (W):	180
Output (Lumens):	33,000
Lifetime (hr):	18,000
Heat to Space (%):	100
Lamp Cost ($):	70.00
Ballast Cost ($):	175.00

NAME	USE	DESCRIPT	TOT	LAM	LUMEN	LIFETM	PC	LAMPCO	BALLCO
F40T12	RIC	Standard	52	40	2,770	20,000	70	1.75	25.00
2F40T12	RIC	2-lamp st	95	80	5,540	20,000	70	3.50	25.00
4F40T12	IC	4-lamp st	176	160	11,080	20,000	70	7.00	50.00
F40T12ES	RIC	High eff.	46	34	2,420	20,000	70	3.00	25.00
2F40T12ES	RIC	2-lamp hi	82	68	4,840	20,000	70	6.00	25.00
4F40T12ES	IC	4-lamp hi	152	136	9,680	20,000	70	12.00	50.00
F40/U	IC	4' U-tube	52	40	2,455	12,000	70	8.00	25.00
2F40/U	IC	2-lamp 4'	95	80	4,910	12,000	70	16.00	25.00
F40/UES	IC	High eff.	46	35	2,265	12,000	70	10.00	25.00
2F40/UES	R	2-lamp hi	82	70	4,530	12,000	70	20.00	25.00
F14T8	RIC	15" fluor	18	14	550	7,500	70	5.00	20.00
F15T8	RIC	18" fluor	20	15	765	7,500	70	5.00	20.00
2F15T8	RIC	2-lamp 18	38	30	1,530	7,500	70	10.00	20.00
F20T12	RIC	2' fluore	25	20	1,250	9,000	70	5.00	20.00
2F20T12	RIC	2-lamp 2'	50	40	2,500	9,000	70	10.00	20.00
4F20T12	RIC	4-lamp 2'	100	80	5,000	9,000	70	20.00	40.00
F30T12	RIC	3' fluore	46	30	1,955	18,000	70	5.00	25.00

Fig. 3: Individual and Collective Screens for Lighting devices; note that for space reasons only part of the description field is shown in the Collective screen.

Data screens are only a convenience -- since all this BPS data is stored in a standard file format (DBASEIII+ or DBASEIV) supported by many other programs, any user is automatically enabled to edit, add, and delete records from these databases using DBASEIII+, DBASEIV, or any one of a plethora of available clones.

On the BPS side, an *application driver* has been written that can read and write the DBASEIV format (and a number of other popular formats). When, in the middle of defining a zone, the user asks for a list of available long-life, fluorescent lighting fixtures for residential applications (i.e., USE=R or RIC, and LIFETM>10,000 hrs in Fig. 3), he or she is presented with a "window" menu schematically reproduced in Fig. 4. Note the presence of a "special," user-defined light fixture created using the external DBMS.

NAME	DESCRIPT
F40T12	Standard 4' fluorescent
2F40T12	2-lamp standard 4' fluorescent
F40T12ES	High eff. 4' fluorescent
2F40T12ES	2-lamp high eff. 4' fluorescent
MY LAMP	Typical Lamp found in my house
2F40/UES	2-lamp high eff. 4' U-tube fluorescen
F30T12	3' fluorescent

Fig. 4: Example of a BPS window for selecting lighting fixtures.

This approach enables the user to customize his/her data with a totally open architecture, yet run any specialized BPS software designed to read this file format.

WORDPROCESSING

The most obvious use for a Wordprocessing program in the context of BPS is to format reports generated by BPS software. Typically, the BPS program can write results to a file which can then be edited by most Wordprocessors. For purposes of this discussion, two definitions are in order: The output from any BPS software can be thought of as composed of *formatting information* and *data*. Data is actual numeric results, building name and address, i.e. anything specific to the building. The formatting information is the boilerplate text within which the results are embedded, and any borders around tables, spaces, empty lines, pagination, etc. For the BPS software developer, there are at least three choices of how to mix formatting information and data in an output file:

- Formatting information and data in ASCII format;

- Formatting information and data in Wordprocessor native format;

- Formatting information in Wordprocessor native format, data in Wordprocessor merge-file format.

The most common choice is the first, whereby the output is written to an ASCII (text) file. This file can be further edited, formatted, and printed by any Wordprocessor. This process is labor-intensive, especially if the user desires a professional looking report. Every analysis will require a similar amount of formatting labor.

We believe that the second choice was introduced to BPS in 1985[4]. A *template file* is written in the native format of one of a number of Wordprocessors. The template file contains all boilerplate text and formatting instructions and looks essentially like the finished output, except for the data, which are represented by *symbolic references*, akin to placeholders. At run-time, the BPS software replaces these symbolic references with the *actual data* produced by the analysis, generating a finished report. The user may edit the template file at any time to change the report format or boilerplate text. Any reports produced thereafter will reflect these changes.

The third choice is a further generalization of the second, and is represented schematically in Fig. 5. The template file, instead of containing BPS-specific symbolic references, uses Wordprocessor-specific *merge field references* (numbers or names, as the particular Wordprocessor permits). The BPS software writes the data to a *merge file* in the native Wordprocessor merge file format. To write the merge file, a *template merge file* (not shown) is used which contains the correspondences between the merge field references and the BPS symbolic references for each piece of data.

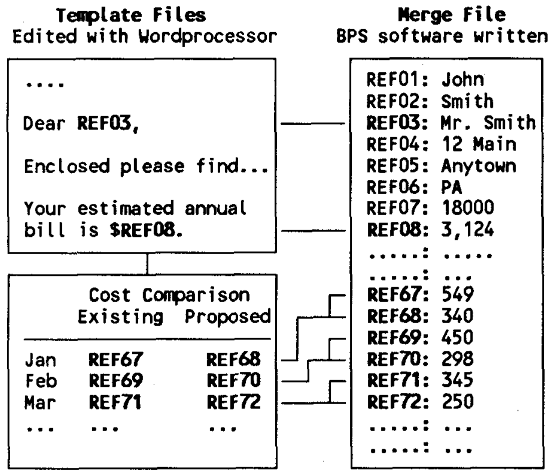


Fig. 5: Example of Merge File Utilization for Report Generation. Merge Field references in Template File are highlighted.

As for DBMS, the point is to divorce, as much as possible, the BPS-specific part of the code, the *Algorithms*, from the generic part, the *Presentation*.

SPREADSHEETS

To further process the data produced by BPS software or to produce a graphic of tabular results an electronic spreadsheet is particularly useful. Again, it is useful to distinguish between *data* to be transferred and *formatting instructions* that tell the spreadsheet how to represent the data. There are at least three different ways to write the results of BPS to an electronic spreadsheet:

- Write data only in a "Delimited ASCII" format;

- Write data only in a native spreadsheet format;

- Write both data and formatting instructions in a native spreadsheet format.

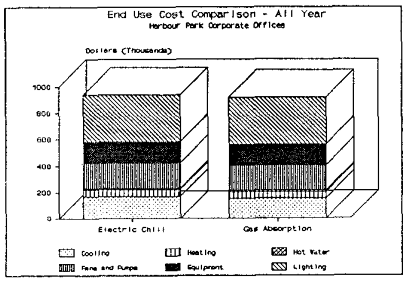
Again the first is the most common way to accomplish data transfer to spreadsheets, and again it is the most tedious. The second way only saves one "cleanup" step in the process. In both cases raw, BPS-produced numbers are in an electronic spreadsheet, ready for further calculations, for graphic representation, or for combining with results from other runs. All of the above, however, require a fair amount of user skill and time. The expert user may automate some of the steps involved by writing "Macros" which is about as simple as programming a VCR several weeks in advance with multiple segments to be recorded.

The third way was first introduced to BPS, to the best of our knowledge, in 1984[5]. Here, in addition to the data, all formatting instructions necessary to generate BPS-specific spreadsheet menus are written directly to a file in native spreadsheet format. All the user needs to do is to run the spreadsheet program, and he or she is presented with spreadsheet-style menus of the type of results that are available. The most popular menu items are those that generate graphics of the results. An example of such graphics is presented in Fig. 6. Notice that the wordprocessor used to prepare the figure can freely mix graphics and text, a feature that is rapidly becoming commonplace.

Metropolitan Gas & Electric Page 6

Section 3: END USE SUMMARY (cont'd)

The following graph and chart outline the operating costs of each end use for the Electric Chiller and the Gas Absorption cooling systems analyzed in this report.



Annual Operating Cost Comparison

End Use	Electric Chiller	Gas Absorption
Cooling	$ 168,507	$ 154,405
Heating	53,751	48,290
Hot Water	10,320	10,264
Fans & Pumps	195,311	194,450
Electric Eqp	155,836	155,863
Process Eqp	0	0
Lighting	355,647	355,642
Elec Subtot	875,300	705,927
Gas Subtot	64,043	212,959
Total	$ 939,343	$ 918,886

Fig. 6: Example of BPS graphics produced by a commercial Spreadsheet and Graphics program.

There are a number of commercial software programs that utilize the native format of the most popular spreadsheet program, Lotus 1-2-3[6], most notably graphics programs and other spreadsheet programs. BPS software that supports at least the

Lotus 1-2-3 format becomes automatically compatible with any new software product that utilizes one of the supported file formats.

DISCUSSION

The advantages of using commercial software for BPS in terms of user requirements and developer constraints discussed in the Introduction are:

- Supporting accepted data interchange standards (DBASE, WordPerfect[7], LOTUS 1-2-3) promotes hardware and software compatibility, connectivity to other applications, and flexibility;

- Transparency is improved by facilitating the essential task of documentation by the developer: Most commercial software is sold with a number of utility programs that facilitate this task.

- Every type of commercial software discussed in this paper is particularly well suited to a few specific tasks needed for building performance simulation: DBMS for storing performance specs for devices, components, assumptions; Wordprocessors for presenting actual results in user-customizable report formats; spreadsheets for graphic presentation and post-processing of tabular results.

- By integrating commercial software with his or her BPS software, the developer or researcher automatically benefits from the efforts of countless programmers who have previously written printer drivers, graphics programs, etc.

There are, of course, disadvantages to writing in a commercial software environment:

- Most popular database formats are lacking certain specialized features required for BPS, such as *field units* (e.g., energy vs. temperature, SI vs. IP), the concepts of a ***not set*** value, of a *calculated field* (e.g., the COP of a Heat pump for which both capacity and input kW have been entered), and more.

- The commercial software data storage format is almost always less efficient (occupying more disk space, requiring more time to read and write to disk) than what could be accomplished by clever programming optimized for BPS requirements;

- The disk space required by the commercial software modules of an integrated BPS system can add up to several MegaBytes; however, many users already have the required software resident on their disks.

CONCLUSION

As researchers, we may be most interested in optimizing Building Performance Simulation (BPS) from an engineering point of view; but in our users' eyes BPS is just one important component in a larger work context. Take the example of modems used for communications. Modems are sophisticated and powerful, treating the user to banks of dip switches and leaving him stranded between odd and even parity. Modems are far outsold by FAX machines which only require the user to stick a telephone plug into a wall jack. All FAX machines contain a modem.

Database Management software can be used to store, sort, and update libraries of buildings and their components. Wordprocessors can automatically format results in a software environment already familiar to many users. Spreadsheets enable the user to analyze results and present them in a graphic format for incorporation in reports.

While we picked three familiar types of commercial software for our discussion, many more promising candidates exist suitable for integration in BPS. Among the most obvious are Computer Aided Design Software, Outliners, Communications software, and Statistical Packages.

For the BPS user, the advantages of working with popular data formats and using commercial software he or she already owns are considerable. For the BPS software developer, supporting well-established standards is both a time saver and insurance against early software obsolescence.

The minimum software requirements by today's users are rapidly outpacing what one company or research group alone can do with limited resources. We have learned in our own practice to "stand on the shoulders of giants," even if they speak the BPS language with an accent.

ACKNOWLEDGMENTS AND REFERENCES

[1] *Quattro* is a Trademark of Borland International, Inc.

[2] *DBASEIII+* and *DBASEIV* are Trademarks of Ashton-Tate.

[3] *R:BASE* is a registered Trademark of Microrim, Inc.

[4] *TrakLoad Energy Audit System*^{TM}, by Morgan Systems Corp.

[5] *TrakLoad*^{TM}, by Morgan Systems Corp.

[6] *Lotus* and *1-2-3* are Trademarks of Lotus Development Corp.

[7] *WordPerfect* is a Trademark of the WordPerfect Corp.