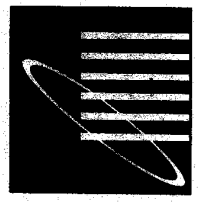

Joining Forces in Building Energy Simulation

J L M Hensen[*], J A Clarke[#], J W Hand[#], P Strachan[#]

[*] Eindhoven University of Technology [#] University of Strathclyde

As a result of the progression of computing power and the increasing demandsfor detailed thermal performance assessment users are shifting from simplified design tools to comprehensive, dynamic thermal appraisal tools which are able to handle the complexity of design. From the standpoint of researchers and developers the days of a single person or even a small group maintaining and advancing such appraisal tools is problematic. Those who wish to advance the state-of-the-art in a particular facet of simulation are given the choice of creating their own tools from scratch or adopting an existing tool and extending it. This paper argues that the latter option can yield significant benefits to the researcher as well as the simulation community. Such a choice does, however contain numerous pitfalls if the core developers and interested third party researchers do not develop a support structure and means of collaboration. The paper will use the evolution of one global consortium related to the ESP-r system as a means to explore the way ahead.

INTRODUCTION

Both with respect to environmental impact and economics, the ability to make sensible and well based decisions regarding the design of buildings and the heating, cooling and ventilating systems which service them, is of the utmost importance.
With current computers, performance analysis by simulation of complex building and plant configuration became available for most of the research community. Indeed, one might predict that the days of simplified design tools are numbered. But so to is the possibility that a single researcher or even a small group will have the time and resources to introduce new products to the design community. Many of the implications of the waste of reinventing the same facilities or being constrained in the exploration of new topics by the massive support code which is required to track the flows of energy have been explored in the recent IFE programmes in the United States and in the United Kingdom. Without doubt, such paradigms are the way ahead and represent the future. This is little comfort though, for the simulation community which wishes to address the needs of the design community in the short to medium term.

Currently a non-trivial investment on the modelling of the thermal interaction of building structure and heating and ventilating system is required to use the current generation of tools. Organisations such as IBPSA provide a forum wherein such technology transfer, training, simulation methodology and validation issues can be explored and thus enhance the applicability of current generation of simulation tools.

For those who are attempting to develop such tools and for those who wish to contribute new ideas and facilities within the current economic climate the problems are daunting indeed. The results are unpleasant but none the less illuminating: The community now has access to dozens of simplified design tools but most are at what one might call the 80 percent stage. They can be used by their authors, but time and resource constraints mean their scope is limited, their facilities are limited and by their bespoke nature they rarely communicate and share data with other systems. If we turn out attention to comprehensive design tools, most are hobbled by

[*] *Eindhoven University of Technology*
Group FAGO, P.O. Box 513
NL - 5600 MB EINDHOVEN, Netherlands
jahe@fago.bwk.tue.nl

[#] *University of Strathclyde*
Energy Simulation Research Unit, 35 Montrose Street
GLASGOW G1 1XJ, Scotland
esru@esru.strathclyde.ac.uk

their interfaces and their monolithic code structures. It requires much passion to evolve such systems and thus change happens more slowly than either the simulation or design community would desire.

At the most disadvantage is the researcher who wishes to further simulation in a particular area but wants not only to publish such ideas but to implement them. One might guess that this is how many design tools began - a researcher writes a package to contain a particular idea or process, which gradually grows into either a simplified design tool or into a comprehensive tool. Given the current economic and research climate it is unlikely that this traditional process can continue, or indeed ought to be repeated.

Thus we arrive at the crux of this paper - how can the community best evolve the current generation of tools, particularly in light of the talents of the many isolated researchers who have much to contribute but no vehicle to do so and groups evolving the current generation tools which can benefit from the freshness of third party contributions. This paper concerns a basic contribution in the area of enabling research consortium.

THIRD GENERATION SIMULATION TOOLS

Early 3rd generation approaches to building energy simulation, often focussed on either the building side of the overall problem domain (especially workers with building engineering backgrounds), or on the plant side (often mechanical engineering backgrounds). In the former approach the influence of the plant system is more or less neglected by over-simplification of the plant; it was/ is common practice to base the estimation of energy consumption on some presumed, imposed indoor air temperature profile. In the latter approach the complex building energy flow paths are usually grossly simplified, and the building (or each building zone) is commonly regarded as just another component which in this case imposes a thermal load on the plant.
It is now felt that neither approach is preferable for the majority of problems which are affected by the thermal interaction of building structure and auxiliary system.
We want to start from the principle that both building and plant have to be approached on equal levels of complexity and detail while taking into account all major fluid flow and heat transfer couplings. If one has access to such a system it would be a particularly powerful and appropriate platform on which to introduce novel facilities. Within such systems there is also the possibility that existing facilities which might rely on a pragmatic engineering approach could be upgraded to use (thermodynamic) first principle approaches.

Starting from an established and internationally recognised platform offers vast advantages for any individual research group. The most important ones are:
- economical; due to the complexity involved and the sheer size of the software to result, it is practically impossible for any (small) research unit to develop and maintain such a system as an independent product,
- academical:
  - as an individual group it is not necessary to have expertise in all areas,
  - areas not addressed within a specific research project will still be state-of-the-art,
  - results transfer to the international research community is implicit and therefore very efficient,
- practical; as more people are using the system, any bugs or flaws are likely to surface - and be solved - sooner.

THE ESP-r SIMULATION ENVIRONMENT

At this point it is useful to move from the generic to the specific. We are working with the ESP-r building and plant energy simulation environment, which originates from the building side of the problem domain, but has been extended to incorporate plant and fluid flow simulation. In terms of research potential, it may be characterised by:
- it is clearly a research orientated environment, with the objective to simulate the real world as rigorously as possible to a level which is dictated by international research efforts/ results on the matter in question.
- it sets out to take fully into account all building & plant energy flows and their inter-connections. It also offers the possibility to assess building & plant performance in terms of thermal comfort. Thus it is specifically suited to do research on subjects in which inter-weaving of energy and mass flows plays an important role.
- the source code is both available and well accessible, because the system is highly modular in nature and offers important features like inbuild trace facilities.
- the system is well documented: it is heavily commented within the code itself, it comes with training and inbuild tutorial material, and there is an extensive manual which is updated on a regular basis (Aasem et al. 1993), and there is also comprehensive background material available (eg Clarke 1985, Hensen 1991).
- the system is now in use by various international research groups.
- the system has been - and still is - the subject of various international validation programmes (see eg CEC 1989).
- the system offers extensive graphics facilities.

- because it runs on UNIX workstations, all other UNIX utilities (for software engineering, numerical techniques, documentation, data retrieval, data reduction, data analysis, etc) which come with the system - or are in the public domain - are 'automatically' available to anyone using the system.

It should be noted though - and this is clearly not meant in any negative sense - that because of its research orientated and evolving nature, ESP-r is not as slick as one would demand of for instance a commercial package. Instead the system expects - and deserves - a pro-active approach of the user.

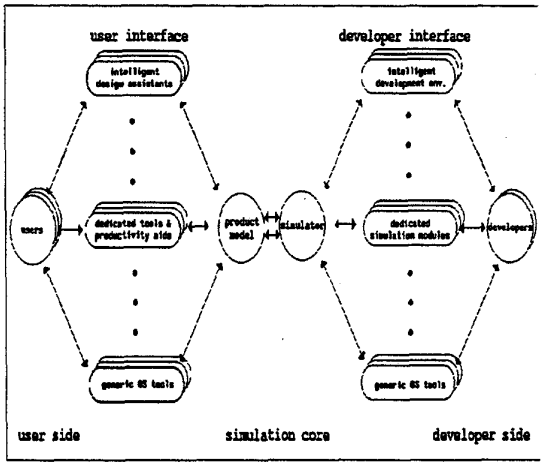

Figure 1 Diagrammatic representation of the ESP-r simulation environment

Figure 1 shows a diagrammatic representation of the various actors, components, and interactions involved in the current environment. For a comprehensive description the reader can be referred to (Clarke 1985, Clarke and Hand et al. 1991, Hensen 1991), and numerous topical papers and other publications.

The core of the system is the **simulator**. It performs the actual simulations using a **product model**. The latter is the complete collection of data describing the model; ie in the current context: building, plant, fluid flow network, occupancy, site, outdoor climate, etc.

Users define and act on the product model using various tools. The tools making up the user interface, generally fall into one of the following three categories :*

- **intelligent design assistents**; high-level user interfaces utilizing state-of-the-art Information Technology techniques, ie an 'intelligent front end' like the *IFe*); this is an important development area;
- **dedicated project tools and productivity aids**; eg the general project manager *esp-r* which - 'hidden' for the users - controls: database managers, specialized pre-simulation analysers (for eg view factors, internal surface insolation prediction, and external surface shading analysis) and the simulation results analysis module;
- **generic tools**; eg as provided with the operating system (text editors, file system managers, etc), assembled from various operating system tools (ie shell scripts), or provided by 3rd party suppliers (either public domain like *grtool*, *xfig*, and *psraster*, or proprietary software like *ww*, *ten*, and *ralbrowse*).

The developer side of the system is formed by the **developers** who, via a **developer interface**, introduce new or change/ expand existing parts/ modules of the simulator. The tools making up the developer interface can also be divided into three categories :*

- **intelligent development environments**; high-level developer interfaces utilizing state-of-the-art Information Technology techniques, ie an object orientated energy kernel system like the *eks*; this is also an important development area;
- **dedicated simulation modules**; which are specialized in simulating a particular aspect of the overall problem domain, like *bld* for building, *plt* for plant, *mfs* for flows, etc;
- **generic development tools**; eg as provided with the operating system (editors, debugging tools, program verifiers, etc), build from these (ie shell scripts), or software engineering tools provided by 3rd party suppliers (either public domain like *toolpack*, *floppy*, and *f2c*, or proprietary software like *forchk*).

The above sketches the context in which the system is 'growing'. It also indicates why more and more people are becoming involved with the (software) development of the system, which itself is becoming increasingly comprehensive and complex.

MANAGING JOINED FORCES

Due to the growing complexity of the system and the increasing number of people involved, there is also an increasing risk of: duplication, negatively interacting parts of the system, undocumented features, unsupported (partial) 'releases', persistent bugs, varying coding styles, etc.

The idea now is, that confirmation to certain accepted rules (or accepted codes of behaviour if you like) will decrease the risks of events indicated above, and will be beneficial especially in the

* As indicated by the dots in Figure 1 there may be tools which fall between these discrete categories.

medium and long term. In the short term, however, this obviously involves getting used to and also a certain degree of self-discipline of all people involved.

Organisational Structure

As elaborated above, we are clearly dealing with a research orientated simulation environment in which changes are taken place at a rather high pace, and from different angles. This is reflected in the organisational structure surrounding the system, which is shown in Figure 2.

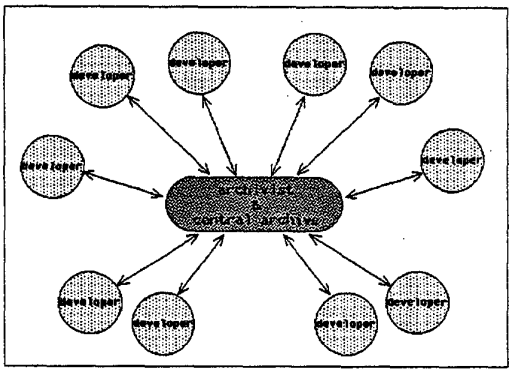

Figure 2 Diagrammatic representation of the organisational structure around ESP-r

The organisational structure is like a star. In the middle there is the central archive which holds the most recent archived version of the system. This archiving is carried out in a centralized location (the Energy Simulation Research Unit at the University of Strathclyde), and the archive is maintained by one person only: the archivist.

Around the archive, there are several developers which are based at various geographical locations. Each developer may - with the permission of the archivist of course - take part or all of the archived version of the system, in order to do the envisaged developments or changes etc. When 'completed', the affected parts of the system are transferred back to the archivist, who will subsequently move them back into the archived version.

It is the archivist's responsibility to enforce some sort of source code control in order to keep track of the system's development path etc. In the future we may decide to use some CASE tool to aid in this task.
It is also the archivist's responsibility to ensure that new source code developments do not adversely affect the other parts of the system in terms of structure; ie to ensure that the system remains coherent. It is each developer's responsibility however, to ensure that his/ her developments are valid in terms of contents and function.

It is important to note that the archived version is basically also the current release of the complete system. This version will incorporate all known and obvious bug-fixes, and will be updated with extensions which were previously developed, tested and approved by both the developer(s) and the archivist. Of course, any new additions to the system will have to be accompanied by a fair amount of documentation regarding objective, how it works, how to use it, examples, etc.

At certain points in time, the archivist may decide to 'officially' announce and release a new version of the system. As indicated above, this version will basically be the prevailing contents of the archive at that point in time. Between two official system releases, the archived version just keeps on 'maturing'. The archive will thus incorporate any recent developments and enhancements. So when a developer plans to make additions, he/ she should 'always' start from the then archived version.

It is also important to be aware that the system does not go beyond what is generally regarded as the β-release level. The reason for this is that all people involved are into research, and not into e.g. commercialisation of source code. Research funding is usually not available for creating say an α-release of the system, although we sometimes - eg in case of developing a system for educational purposes - would like to do so.

Electronic Discussion Platform

To keep eachother informed about ongoing and planned developments, new releases, etc. some regular schedule of general meetings might be useful. This would involve travelling and costs. In our case, a natural alternative is to do it via some sort of electronic discussion platform for ESP-r.

The purpose of this facility is to keep colleague researchers informed about ongoing and planned developments, system related publications, new results, emerged problems, upcoming releases, etc. One of the main benefits of this would be the improved efficiency ie. increased output of us the researchers, but it is quite easy to see many more advantages.

With respect to what form this communication platform should take, one might consider something like a printed newsletter. But since we are all working with UNIX based systems it seems much more sensible to exploit workstation technology and use some electronic form. A central bulletin board would do,

but this needs to be maintained by someone who probably already has several other tasks etc. The alternative which has been chosen for now, is to set up a conferencing facility using electronic mail.
The way in which this mechanism can be used is as follows: each researcher who wishes to subscribe to this facility or who wants to communicate some message or file to colleagues sends this contribution by e-mail to: *esp-r@esru.strathclyde.ac.uk*. This 'user/ machine' then sends a copy of the incoming e-mail message to all subscribers to the conferencing facility. In this way a form of automated electronic bulletin board is established. Currently about 40 researchers in 18 countries world-wide are subscribed to this service.

GOOD PRACTICE PROCEDURES

To re-iterate what was suggested above, it is important to note that we are engineers and scientists - not trained in computer science - for who solving a scientific problem is more important than applying sophisticated software development methodologies. We are writing our own applications and have to rely on the most elementary software tools - editors and compilers - to develop applications which can be large and complex. We do so because the methodologies supported by modern software-development tools, which are derived from commercial environments, do not fit our work habits or our budgets. By trial and error, we have developed a methodology of our own which is based on incremental development and focuses on code production. In a way, we usually put together a still incomplete software version and try it out using ad hoc debugging. We like to achieve rapid solutions, and this may overshadow concerns about requirement specifications, design phases, or documentation.
If we are not very careful, the result might be a monolithic program or module which is difficult to debug, modify, reuse, and port - and is understood only by the developer. In short, the end product has poor quality, which becomes apparent when code must be reused for new purposes or bugs start appearing 'mysteriously'.
To maintain and improve the quality of the software we write and use, we adopted some 'good practice guidelines' which are based on current 'group practice' and seem to give reasonable results.

The first and most important rule is that each developer is **responsible** for his/ her **own work**. This obviously implies properly testing and documentation of own developments. Failing to do so will not only affect him/ herself but effectively all the colleague researchers as well. In the short term because the colleagues will be suffering from somebody elses 'bugs or flaws', and in the long term because it is very hard to get a good name, but very easy to get a bad name (with all related consequences).

As indicted in Figure 2 the organisational structure is like a star. Source **code transfer** to colleagues should **only** take place **via** the **central archive**; ie. a developer should not take code directly from a colleague. Also, if a new development is started, start from the archived version. This is to prevent any mix-ups.

Of course, a developer may choose to have his/ her current work used and tested by a local group of pro-active users, before submitting it to the archive. This will help to make the software more robust and will prevent premature general release of the software. Obviously in those circumstances, documentation is allowed to still be somewhat course.

Before being put into the archive any - new or altered - source **code** should be **checked thoroughly** for errors with some code cheker. Obviously, such a package can only spot semantic or structural errors. Functional or algorithmic errors are much harder to track down. There needs to be another way of achieving this; ie. the important issue of verification and validation as described in eg the PASSYS final report (CEC 1989).

When new or altered code is submitted to the archive it needs a short **high-level description of changes**. This information is to be used by the archivist in order to keep a record of the system's development path.

When new - or altered in terms of user functionality - code is submitted to the archive it should be accompanied by a new - or updated - **manual page** - and if possible also a **tutorial** - for each affected module of the system. This kind of documentation is primarily aimed at the users but is also very useful for other developers since it gives a 'verbose' description of any changes in the system.

It seems like a good idea to have some reference problems (documented in the manual) associated with each module, on which subsequents release can be tested in order to see the impact - on various problems - of the new changes when compared with the previous release.

Good Practice Coding

It is important that the source code has some sort of common coding style*. This has an esthetic func-

* It is certainly not the intention to enforce certain new elements, it is more a case of describing of what seems to be accepted (implicit) rules/ style etc at the moment. This can obviously change in the future.

tion, but more importantly it increases the readability of the code and makes it more accessible to other (future) developers.
Partly for the same reasons, it is also crucial that the source code is well documented.

Style

In terms of code style, one of the most important things to do is to write code in a clean and concise fashion; ie stick to the KISS (Keep It Simple, Stupid) approach. Make sure that code is easy to read and understand (again) for yourself (even after some months not being involved with that particular code) and your colleagues. This implies that in case of evaluating an equation say, it is usually better to write it down the same as you would write down the corresponding scientific equation. Do not combine terms - and disstort the clearness - just because this will save some CPU time. In our context, we are far more interested in saving development time; ie kissable code.

Try to make code as structured as possible. Obviously this is easier - and more natural - in C than in FORTRAN. But even in the latter case, there are several - sometimes only visual - possibilities to achieve this. As a starting point one probably wants to avoid usage of GOTO statements in FORTRAN.
Note that some FORTRAN compilers - like eg the Sun FORTRAN compiler - support some additional features which give you the possibility to use high level structure mechanisms like for instance DO WHILE loops. It is very advisable to **not use** such non-standard (relative to ANSI 1978) features, because they are very likely to impose problems whenever the system is ported to other platforms or when other user/ developers use different compilers. There are however some non-standard features which are supported by all compilers (for example usage of INCLUDE files) and may therefore be used without risk.

Try to use - as much as possible - self-explaining variable/ parameter names. Don not change the meaning of a particular variable/ parameter somewhere else in the code; for example if say *wspeed* is used for windspeed don't change the intrinsic meaning to 'windspeed squared' just because it is used as such in most places. If you choose for the latter it is far better to use a variable called say *wspd2* in the first place.

Frequently, modules in scientific applications like ESP-r have some sort of common format because they perform similar tasks. For instance the coefficient generators for the different plant component types differ usually only in the equations calculating the actual matrix coefficients. In case some additional modules are developed for some feature for which there is already a 'master format' (eg plant component coefficient generators, fluid flow calculators, etc) it is advisable to conform to that 'master format' because that is probably more clear for a (future) 3rd developer.

Documenting

The kind of documentation this is about, is intended to be used by developers; ie. here we are not addressing the user manual pages. We are also not addressing the 'high-level description of changes' as indicated in the previous chapter, which is used for keeping a record of the system's development path.

Nobody wants to create documentation, but everybody needs it and complains when it is poorly done. Documentation should be complete and consistent. There are two basic reasons for this. First, developers who go back to their code after some time cannot remember details. Second, the personal context disappears when the code is transfered to colleagues.
As with style, code documenting is very much a question of commonly adhered to style. Again this also implies that this is something which will probably change continuously with time.
Some general points worth noting with respect to code documenting are:

- use proper English sentences to document;
- each subroutine starts with a high-level explanation of its purpose and how this is achieved;
- each subsequent (major) task or activity in a subroutine is preceded by some documenting comments;
- in case of FORTRAN, COMMON variables are documented in the subroutine where they are first introduced.

When these simple rules are adhered to subroutines are easy to follow and understand, and if we were to collect all embedded comments this would almost lead to a 'report' of what's going on.

CASE Tools Aiding Good Practice

There exist a number of Computer Aided Software Engineering (CASE) tools which may prove helpful for maintaining good practice. Some of these tools come with the UNIX operating system, like: *dbx* and *lint*. Other CASE tools are so-called 3rd party products which are either proprietary software or in the public domain. An noticable example of the latter is the *toolpack* suite of FORTRAN CASE tools, incorporating features like syntacs and semantic checking, static analyses, run-time performance analyses, code tidying and polishing, transforming single-precision into double-precision or 'long variable names' into FORTRAN standard names, rebuild the flow of control

in standardized form, etc.
The first release of this software tools suite, was the result of an international collaborative effort started in 1979. The original project aims were twofold: (1) to provide a suite of tools to assist the production, testing, maintenance and transportation of medium-sized mathematical software projects written in standard-conforming FORTRAN 77; and (2) to investigate the development of extensible programming support environments built around integrated tool suites.
The Toolpack project, was supported in the U.S.A. by the National Science Foundation and the Department of Energy, and in the U.K. by the Science and Engineering Research Council. Because of the nature of the project funding, the resulting software is in the public domain.

CONCLUSION

It has been argued that collaboration in the field of building energy simulation yields significant benefits for individual researchers - or research groups - as well as for the simulation community as a whole. The main benefits are increased efficiency and output, and more rapid developments. This cannot be achieved by some 'enforced' collaboration, but only on the basis of individuals sharing a common goal or belief. This seems to be equally true with respect to a collaboration support structure and control. Instead of trying to enforce certain new elements or a rigid structure, it is much better to start from 'accepted group practices' and to set up some sort of flexible means of interaction and control. This may then all change over time, if new possibilities arise or if certain opinions/ views alter. We feel that the organisational structure, the communication means, and the 'good practice procedures' concerning ESP-r prove to be well workable in this respect.

Acknowledgements

Joining forces in building energy simulation only works on the basis of individuals willing to do so and having a pro-active attitude in this respect. The results achieved thus far, would not have been accomplished without the continuing support of many people. For this we wish to express our sincere gratitude to all our ESP-r colleagues.

References

Aasem, E., J.A. Clarke, J.W. Hand, J.L.M. Hensen, C.E.E. Pernot, and P. Strachan 1993. "ESP-r A program for Building Energy Simulation; Version 8 Series," Energy Simulation Research Unit, ESRU Manual U93/1, University of Strathclyde, Glasgow.

ANSI 1978. "American National Standard Programming Language FORTRAN," ANSI X3.9-1978, American National Standards Institute, Inc..

CEC 1989. "The PASSYS Project Phase 1. Subgroup Model Validation and Development Final Report 1986-1989," 033-89-PASSYS-MVD-FP-017, Commission of the European Communities, DG XII of Science, Research and Development, Brussels.

Clarke, J.A. 1985. *Energy simulation in building design*, Adam Hilger Ltd, Bristol (UK).

Clarke, J.A., J.W. Hand, P. Strachan, J.L.M. Hensen, and C.E.E. Pernot 1991. "ESP^R A building and plant energy simulation research environment," Energy Simulation Research Unit, ESRU Manual U91/2, University of Strathclyde, Glasgow.

Hensen, J.L.M. 1991. "On the thermal interaction of building structure and heating and ventilating system," Doctoral dissertation Eindhoven University of Technology (FAGO).