THE GENERALISED SYSTEM SOLUTION CLASSES IN THE EKS ENVIRONMENT

Dr. D. Tang

Energy Simulation Research Unit
Department of Architecture and Building Science
University of Strathclyde
Glasgow, U.K.

ABSTRACr

'I'lie Energy Kernel System (EKS) is an energy simulation environment that facilitates the creation, validation and maintenance of simulation programs using the object-oriented programming (OOP) paradigm. This paper introduces the particular aspect of the EKS development concerning the fundamental issues of system representation; that is, the theory encapsulation and system solution which constitute the implementation of the generalised solver classes in the object-oriented environment.

INTRODUCTION

The Energy Kernel System (EKS) is an energy simulation environment that facilitates the creation, validation and maintenance of simulation programs using the object-oriented programming (OOP) paradigm. Ultimately the EKS will be capable of building a large range of environmental performance analysis programs, from reduced order models up to the present day state-of-the-art. The functionality required to build these programs will be built into a collection of classes, each normally addressing one aspect of the overall task. Programs with different functionality based on different mathematical representation structures can also be constructed by choosing alternative classes, or variants of classes. Central to this environment, the computational support is provided by the generalised solver classes which embrace a spectrum of analytical, numerical and statistical solution methods to be used in the process of model-building.

This paper describes the fundamental methodology underlying the the development of the EKS theory representation, encapsulation and system solution. The overall description of the historical background and current development of the EKS project can be found elsewhere (Clarke et al, 1991).

SYSTEM REPRESENTATION

In order to achieve mathematical consistency and minimise numerical error propagation, most of the building performance simulation models adopt coherent model discipline throughout the system, e.g. finite difference, finite element, response factor. For those multi-discipline systems which use mathematically different structures to represent individual components, the individual module has to be treated in isolation and linked by parameter passing. Simultaneous solution can only be achieved by iteration.

What is envisaged within the EKS is the provision of a method of representing different mathematical theories in a consistent format and therefore supporting a wide range of approaches. The easiest solutions may be to simply accommodate all the modules with different input/output interfaces into the EKS, and then:

- provide interfaces to each pair of them to allow an arbitrary combination at run time; or
- use a *switch* construct to allow arbitrary run time module selections.

Unfortunately these are not the appropriate solutions for the implementation in the OOP environment. The former gives rise to the problems of *interface complexity* and *combinatorial explosion*. The consequence of this leads to the classical problem of permutation and combination. The latter solution is fatal for its *code redundancy* which would eventually carry the code burden through to the most simple model being built. This can be illustrated in the following example:

A parent class 'Room' instantiates 5 lower classes: Air_volume, Occupant_gain, Equipment_gain, Lighting_gain, and Plant_gain. This would allow the building of any room model including all or part of these 5 attributes, e.g. a room without Air_volume but with all the other 4 attributes for a steady-state model in which all the gains have an instantaneous effect on the room load. If each lower level class has 3 types of theory which require different data structures as input and output, then, in order to build a room model without redundant code in case of the absence of any number of attributes, the room has to possess interfaces for the following number of combinations:

$$C_5^0 3^0 + C_5^1 3^1 + C_5^2 3^2 + C_5^3 3^3 + C_5^4 3^4 + C_5^5 3^5 = 1024$$

The EKS Approach

The EKS adopts an internal schema which allows the presentation of most of the equation-based theories in the domain of building energy modelling in a consistent manner via the 'vectorised state-equations':

$$\dot{X}(t) = A(X,U)\, X(t) + B(X,U)\, U(t) \qquad (1)$$

Notice that entries in the coefficient matrices A(X,U) and B(X,U) are generally nonlinear functions. The returns from those nonlinear functions are not necessarily restricted to values or definite types of coded algorithms. Table 1 shows the variation when applied to different types of systems:

Equation type	Entries in matrix
Partial differential equation.	Partial differential operators.
Spatially discretised nonlinear partial differential equation.	Pointers to functions.
Linear time variant differential equation.	Time series.
Linear algebraic equation.	Constants.

Table 1 Variation of coefficient representations.

It has been proved that for any nonlinear system which is represented by its original simultaneous nonlinear equations, there always exists an identical representation in the form of equation (1). Furthermore, for any system which is transformed into equation (1) from its original nonlinear representation, it can always be transformed back to the original representation (Tang 1990). Therefore the representation of equation (1) is an identical transformation. This type of representation has been used currently throughout the EKS physical classes concerning thermal discretisation.

Supplementary to the representation of equation (1), the EKS also provides a secondary representation in the form of the bulk nonlinear equation:

$$\dot{X}(t) = F(X(t), U(t), t) \qquad (2)$$

It is obvious that the equation (1) type representation can be identically converted into equation (2) by simple arithmetic operations. For the equation (2) type system, there is no one-to-one reverse mapping back into representation of equation (1). This fact ensures that in the case of time-discrete numerical solution, the system represented by equation (1) can have direct access to both linear and nonlinear solution tools without losing information. For systems in the form of equation type (2), the access is restricted to purely nonlinear system tools.

THEORY ENCAPSULATION

Based on the theory representation outlined above, the theory encapsulation classes have been created in the EKS to ensure data security and at the same time provide the machine-efficient means for further system manipulations. As most of the building energy modelling systems that include building fabric, air flow, plant and control are predominantly represented by highly sparse system equations, sparse techniques have to be applied to avoid the excessive exhaustion of machine resources.

To achieve these, the EKS offers two types of sparse array techniques for theory encapsulation, namely the pointer sparse array and the implicit row-wise sparse array. The pointer sparse array technique is implemented via the five classes of Equation_set_list, Equation_set, Equation, Coefficient, and Region_id, and is used among the EKS physical classes for theory encapsulation. The implicit row-wise sparse technique, which has a more economic storage requirement, is currently implemented within the solver class cluster to support a number of EKS solvers and operators. Both of the techniques store only non-zero entries of the system matrices. An extra array is used to store the information position related to each non-zero entry. The maintenance of the two types of sparse array classes are provided by the generic List class: a doubly-linked list. A number of private functions are created for internal data manipulations.

1 The pointer sparse array

Figure 1 shows the example of the pointer sparse array with maintenance of the generic lists. A header pointer is used to point to the list of coefficients in the equation and an equation identifier for the location of the equation. Each coefficient in the list contains its function entry and an identifier specifying to which region it belongs. The head pointers and the coefficient list pointed to by the head are maintained by two independent generic doubly-linked lists in orthogonal directions.

2 The implicit row-wise sparse array

The implicit row-wise sparse array is one of the most economical schemes for sparse matrices. Figure 2 shows this sparse array with maintenance of a single generic list. Each element in the list has a coefficient entry and an identifier which identifies the relative position of the coefficient entry. For a two dimensional $m \times n$ sparse matrix, for example, the i, j position in the matrix will be identified by:

$$id = i \times n + j \ ; \quad (i=0,1,...,m-1; \; j=0,1,...,n-1)$$

The saving of machine storage requirement by using the above sparse array rather than the conventional two dimensional array can be significant. For example, when finite difference methods are applied to the transient building conduction problems, there will be no less than an n/8 times saving of machine memory requirement. Here, n is the total number of nodes in the system.

A number of private functions has been created for the maintenance of the the above theory encapsulation classes and the list class. Some of these are shown in the following tables:

- Encapsulation manipulation:

Name	Functionality
get_row()	get row number for given relative position.
get_column()	get column number for given relative position.
get_position()	get relative position for given column and row numbers.
get_coefficient()	get coefficient for given relative position.
2D_to_sparse()	get sparse representation for given 2-D matrix.
sparse_to_2D()	get 2-D matrix for given sparse representation.

Table 2 Functions of theory manipulation.

- List maintenance:

Name	Functionality
insert()	insert an element in the head of list.
append()	append an element to the end of list.
remove()	remove an element from the list.
first()	find the first element in the list.
last()	find the last element in the list.
clear()	delete the entire list.
size()	find the total number of elements in the list.
sort()	sort the list in a particular order.
iterator()	sequentially search the entire list.

Table 3 Functions of list maintenance.

SYSTEM SOLUTION

The class structure of the EKS has been designed to be able to accommodate most of currently available solver modules. In doing so, all the solver classes in the EKS will have a common interface i.e. the theory encapsulation classes based on the EKS internal representation. At the present time, most of the available linear and nonlinear solver software pack-

ages are interfaced by two types of systems, the linear algebraic methods for linear systems and the iterative methods for general nonlinear (including linear) systems. As such interfaces conform directly to the EKS internal representation as given in equations (1) and (2), these solver modules can be transferred into the EKS and encapsulated as solver classes with only minor modification. For those proprietary solvers which are embedded in the coded algorithms, they must be made modular by separating the abstract mathematical solver from the algorithms, before they can be replanted into the EKS environment. It is envisaged that the future EKS developer may opt to add new solver classes which do not conform to the internal theory representations; this would require the use of the EKS Metaclasses (Clarke et al, 1991).

Two levels of solver classes are provided by the EKS solver cluster:

The fundamental level includes conventional vector, matrix operators and linear, nonlinear system solvers. These are mathematical tools/ modules usually available in the current software libraries or other sources such as text books. The EKS makes no effort to improve the efficiency of each individual module except by providing the interface for data encapsulation. In this domain of solution tools, the EKS provides general solvers for the solution of linear and non-linear differential equations as well as algebraic equations. This may include a number of well known methods, e.g. Runge-Kutta method, Gear method, PC methods, multi-step methods. In limit cases, analytical solution methods are provided for linear time-invariant systems. For system operation, the EKS provides a wide range of mathematical tools including vector inner product, vector norm, matrix product, matrix determinant, matrix inverse, pseudo-inverse, singular value decomposition, matrix eigen-values, matrix eigen-vectors, to allow direct system parametric studies. This also provides the possibility for users to compose solvers of their own by the combination of different vector/ matrix operators.

In addition to the conventional methods, a number of advanced system operation/ solution tools have been investigated and their implementation is underway in the EKS demonstrator. The tools include:

- Sparse operators and solvers

In the prototype stage, the EKS has developed a number of vector/ matrix operators and direct solvers which are fully operational, based on the 'implicit row-wise sparse array'. This means that these operators and solvers receive, store and process only on the non-zero entries of the vectors and matrices. With additional tools added for pre-processing, e.g. profile reduction and sorting, the sparse solvers are able to ensure that minimal non-zero entries are created during the elimination process.

- Graph interfaces and algorithms

A methodology has been developed within the EKS to enable the mapping of a system network/ graph into state-space representation or vice versa, based on the intrinsic relationship between graph and vector symbolism (Tang 1990). This means that for any given system network/graph, the EKS is able to provide an identical set of equations which are then encapsulated in the theory classes. This is the fundamental tool for system automation. For example, after a system has been drawn on the screen via a graphic interface, the topology of the system network is then depicted by the characteristic matrices of the system graph. The identical set of system state space equations can then be obtained via the methods of graph interface.

Eventually the EKS will provide a wide range of graph algorithms for system optimisation and solutions. In the demonstrator, attention is particularly concentrated on those graph methods which support the advanced solver functionalities. These include the methods of (Pissanetzky 1984):

tree partitioning,
profile reduction,
nested dissection,
ordering,
etc..

Based on these methods, the Decoupling Simultaneous Solution (DSS) technique has been developed in the Energy Simulation Research Unit to allow an efficient solution of general large system networks. By this method, a large system is sorted, partitioned and reassembled into several relatively independent sub-systems. The method guarantees that the independent solutions of the augmented sub-systems are theoretically identical to the simultaneous solution of the complete system. This offers a powerful tool for building simulation and ensures that many existing simulation models can be re-used after minor modification; computing power required for solving the much smaller sub-systems of the combined modelling is much reduced. The first validation effort of this technique has been carried out on the simultaneous solution of the building energy and turbulent air flow system (Zhang, 1990).

Figure 3 shows schematically the basic principles underlying this method. In (a), it shows the integrated building and air flow system nodal network which addresses the 3-D solid thermal diffusion and turbulent air movement. The two parts of the system are physically coupled and reflected by the integrated mathematical structure. In (b) the system network is sorted and partitioned into a quotient tree of sub-systems using nested dissection techniques. Here only two levels are formed. In (c) the coupling parameters are firstly identified and then condensed into encapsulations properly located in the system. Finally in (d) the graph elimination is applied to the dissection tree and the integrated system is reassembled into two relatively independent systems in concatenation. The EKS sparse solvers can then be applied to the two augmented sub-systems to obtain solutions. The method ensures that the results thus obtained from the sequential and independent solutions of the two augmented sub-systems are identical to the simultaneous solution of the integrated system as given in (b).

CONCLUSIONS

The EKS energy simulation environment facilitates the creation, validation and maintenance of simulation programmes using the OOP paradigm. Within it, the solver classes are one of the most important aspects of the EKS development. The central issues involved in the support of such an environment lie in the methodology for embracing a wide range of current modelling structures in a coherent manner, and in providing possibilities for future development.

Within the EKS development, the methodology governing theory representation and theory encapsulation has been developed. This enables the solver classes to offer a spectrum of tools for system manipulation. In addition to the conventional linear and nonlinear system solution methods which are now installed in the EKS solver classes, the EKS has developed its

own features by providing efficient tools for advanced system modelling.

REFERENCES

Clarke, J. A. et al 1991. "The Energy Kernel System." In *Proceedings of BS'91*, Nice, France, Aug. 20-22.

Pissanetzky, S. 1984. *Sparse Matrix Technology*. Academic Press Inc. Ltd. London.

Tang, D. 1990. "The EKS Theory Representation". EKS Project Document. Energy Simulation Research Unit, University of Strathclyde, Glasgow.

Tang, D. 1990. "An investigation into the intrinsic relationships between graph and matrix representation of building physics in the context of the EKS project". EKS Project Document. Energy Simulation Research Unit, University of Strathclyde, Glasgow.

Zhang, Y. 1990. *Combined Building Energy and Air Flow Simulation*. MSc Thesis. Department of Architecture and Building Science, University of Strathclyde.

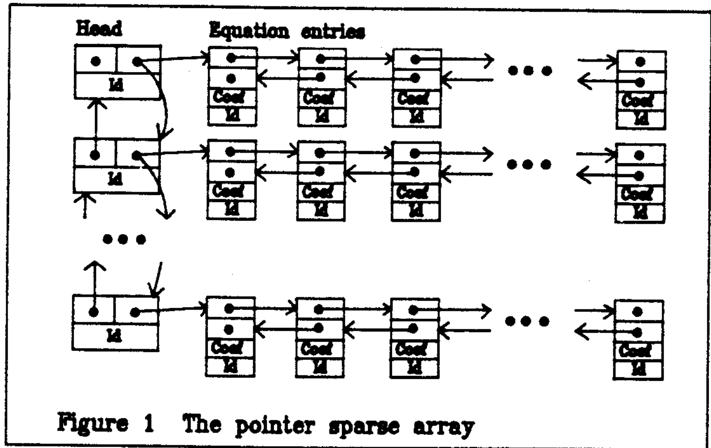

Figure 1 The pointer sparse array

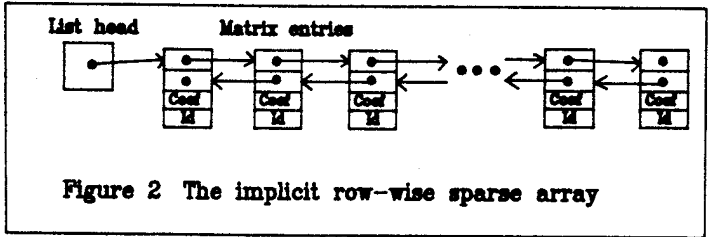

Figure 2 The implicit row-wise sparse array

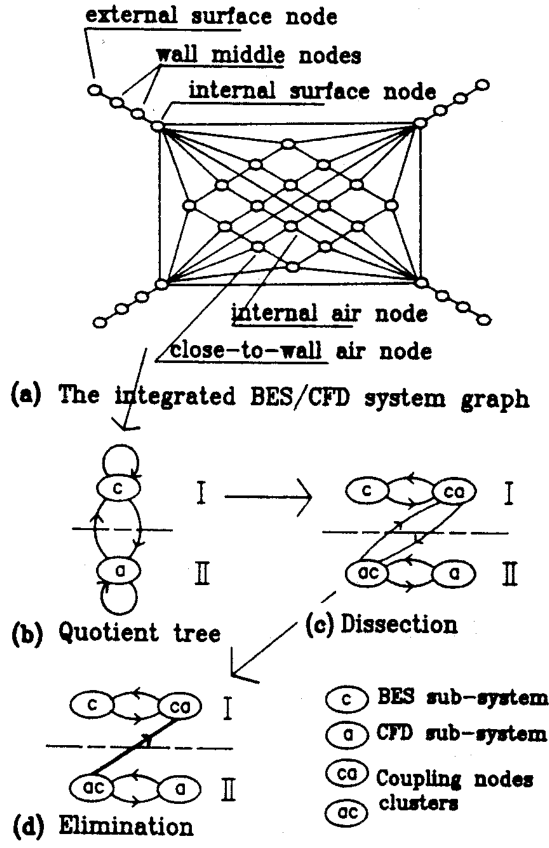

Figure 3. The decoupling simultaneous solution method.