The Energy Kernel System: Form and Content

J A Clarke
Energy Simulation Research Unit
University of Strathclyde, UK

D F Mac Randal
Informatics Department
Rutherford Appleton Laboratory, UK

The Energy Kernel System is an advanced machine environment intended to foster the collaborative development of the next generation of performance assessment programs. Firstly, the overall system architecture and the role of the underlying object oriented database is described and some of the underlying technology issues briefly mentioned. Then the principal classes, the methodology used to identify them and their construction and organisation is addressed. Finally the internal mode of operation of the EKS and the steps required to define a program of specified functionality are detailed. Yhe theory representation used within the EKS and the object oriented encapsulation of this approach is also elaborated.

Project Summary

The EKS project was undertaken as a collaboration between four institutions - the Universities of Bath, Newcastle and Strathclyde, and the Rutherford Appleton Laboratory - funded by the UK Science and Engineering Research Council (SERC) Final reports were submitted in 1992 (Wright el al 1992, Hammond et al 1992 and Clarke et al 1992).

The project set out to explore the feasibility of developing an advanced program building environment, the EKS. The objectives were to:

- Identify the computational methods underlying building energy/ environmental prediction models.
- Develop a procedure for establishing these methods in an organised form within a demonstration system.
- Research the feasibility of adopting the object-orientated (OO) programming paradigm in the representation of the methods and their underlying data structures.
- Identify suitable theories and computation methods for inclusion within the EKS demonstrator.

Having become familiar with OO technology - in terms of an OO language, C++ (Stroustrup 1987), and OO database (ONTOS 1989) - it proved possible to construct a taxonomy of classes from which models of different functionality can be built. The role of these classes is to represent the physical entities which comprise a building (rooms, walls, etc) and the abstract entities which dictate its thermodynamic state (heat transfer theories, numerical methods, etc). These classes are organised into independent 'used by' and 'derived from' hierarchies and placed under the control of an instantiation mechanism. This means that programs possessing different modelling capabilities can be constructed automatically by merely selecting the required class variants - that is no user coding is required. And because each physical entity within a modelled building has a matched object at runtime, an EKS-produced program can be matched to the physical system it is being used to model. In particular the EKS class taxonomy has been progressed to a stage where it can support the construction of demonstration programs which exhibit near state-of-the-art characteristics. These classes have been placed under the control of an OO database to facilitate added security of use and support object persistence.

At the present time an EKS Demonstrator exists and SERC has commissioned an independent review in order to decide on the way forward.

EKS Demonstrator Overview

The Classes which comprise the EKS are organised into a 'taxonomy' as shown in Figure 1. This taxonomy specifies how the classes interrelate and defines the information flow between them. The ever present dilemma between extensibility - that is the ability of any existing class to use or be used even by a newly created class - and security - that is the guarantee that the given classes are compatible - is solved by using special "Metaclass"[#] classes. A "Metaclass" defines the behaviour of its associated class, including which other classes are required for its correct operation. Thus, by insisting that programs can only be built by using "Metaclass" classes, it is possible to have an extensible but secure system without paying the

performance penalty of runtime type checking.

Actual program building is carried out by the "Template" class. Given the chosen program architecture, as defined by a particular "Context" class, the "Template" builds up a collection of "Metaclass" instances which define the program structure. These "Metaclass" objects can then check that the specified program is internally consistent. Furthermore, since the "Template" is in effect a specification for the program, there is no need to generate an executable program at this stage. Instead, once the problem specific data is available, the "Template" can use its "Metaclass" classes to create a customised program tailored to the problem being addressed. This is all achieved internally in the "Template" class; the program builder or user need know nothing about the "Metaclass" scheme.

In its present form (see Figure 2) the core of the system is an Object Oriented Database (OODB). This plays three separate roles. Firstly, it can hold entities such as climatic data and material properties encapsulated as persistent objects*. Secondly, it can hold the problem description and results as objects, enabling the interfaces to be separated from the body of the performance prediction engine. Thirdly, it holds the "Template" and "Metaclass" objects used internally by the EKS environment.

Recognising that reliance on an OODB platform could restrict EKS exploitation, the system was configured so that it is possible to run the system with or without the OODB. In the latter mode some of the program composition checking features are disabled and the absence of the OODB necessitates a system rebuild when a new class is added (although it is still significantly easier than the equivalent work using conventional program development techniques).

As shown in Figure 2, surrounding the OODB are a number of utility modules which are used to specify the architecture of a particular program. Assuming that the user has a well formed modelling hypothesis, EKS operation will entails the use of the following programs:

EKS_cb

A program which allows the definition of a program's context (this is equivalent to the creation of a main program in conventional programming). For example it may be that one program might offer site and building modelling capabilities while another might offer a site, multiple buildings and plant capabilities. Within the EKS two different "Context" classes would be required to express these alternatives, although clearly the latter could be derived from the former. The output from this module is a "Context" class. (The EKS Demonstrator comes complete with several example "Context" classes to demonstrate the program specification process.)

EKS_tb

Given a "Context", this module allows the user to specify the precise capabilities of a program by selecting EKS class variants as required. As each class is selected the "Metaclass" mechanism determines the dependent classes so that the process is automated. This allows program building to be carried out incrementally and with no need for specialist knowledge in terms of the underlying algorithms. The output from this stage is a program "Template" which defines the classes and class connections from which the program will be constructed. A "Template" can either be stored in the OODB for later recall or held as an ASCII file.

EKS_dd

This module takes a "Template" as input and outputs the corresponding OO product model. If the data of this product model are unacceptable, in that they cannot be obtained from the intended end user type, then the previous applications can be revisited and the program architecture modified. At the end of this iterative process, the data requirements would typically be made known to some third party problem definition package such as a separately tailored interface application or a CAD system.

EKS_dm

This module allows the definition of a given problem in terms understandable to the EKS generated program and holds the information in the form of "X_def" objects. (The term "X_def" stands for 'something_definition' where 'something' might be a room, a material, a plant component, a site, etc. Typically a program will require several "X_defs" to define the site, building geometry & construction, plant layout and so on.) These "X_defs" can then be stored within the OODB.

EKS_mb

Program construction can be placed under the control of the OODB in which case the OODB-installed "Template" is consulted and correct class use can be guaranteed. Alternatively, and for use in cases where an OODB is unavailable, module EKS_mb can be used to build the required model from the "Template" as held in an ASCII file. This procedure is equivalent to the conventional link/ load operation.

EKS_rm

Finally, this module is used to associate the program with its data as held in "X_def" form. Again this operation can be placed under OODB control

Throughout this paper quoted names signify classes.

* An object is a specific instance of a general class.

or invoked conventionally.

These EKS interface tools exist in two forms to handle the case where the OODB is present and the case where it is not. In either case they will appear identical to the user, the only difference being that the program "Template" and corresponding "X_defs" will either be stored within the OODB on within an ASCII file on disk.

EKS Classes

Within the EKS classes break down into three basic types - base, principal and intrinsic:

Base Classes

These define the interface of the EKS principal classes and therefore their basic capabilities. Base classes are 'generic' classes which cannot be used because their functions are virtual, being implemented only in the derived principal classes. Essentially base classes exist to allow a future extension of the EKS class hierarchy into other domains without the need to carry the thermodynamic functionality of the existing EKS classes. They also exits so that the entity they represent can be implemented in a variety of ways while guaranteeing that these different implementations will operate with the other EKS classes.

Principal Classes

These are selected by a program builder to define the capabilities of some target program. Each principal class is derived from a corresponding base class and offers a particular implementation of the virtual functions of its parent - that is they represent the physical or thermodynamic states of the entities they represent. In some cases, and with domain theory type classes in particular, several alternative principal classes will exist to represent the alternative theoretical approaches. A later section on this class type gives more detail.

Intrinsic Classes

These are the internal work horses of the EKS serving to transport data between the principal classes, to control the class selection process at template specification time, to contain essential support data such as climatic time series and to dimension all properties of state. Intrinsic classes, because they are not selected by the program builder, are not shown in the EKS class taxonomy. Intrinsic classes include:

- Data classes to encapsulate climatic data, material properties, program results and the like. Instances of these classes may be held within the OODB as persistent objects. These objects, which are equivalent to the data-sets available for use with conventional energy programs, are available to all EKS constructed programs.
- Dimension classes - temperature, pressure, mass flow rate, weight, etc. - used to 'type' all EKS data and so ensure data security (Wright el al 1992).
- Computational support classes - vector, equation, matrix, etc. - to encapsulate support data such as time, location and state variable.
- Infrastructure classes which assist in the program building process and facilitate object control at run-time. Such classes include "Template" to hold the program definition, "Metaclass" to hold the class relationships and "Network" to hold topological information (such as the "Construction" to "Room" relationships).

The separation of the underlying functions in this way gives maximum flexibility and code reuse when creating new programs. The base classes provide a common semantic basis for the principal classes and permit classes derived from them to be reliably used in different contexts. Derived classes are subtypes of their parent, that is a class may be transparently replaced (even at run-time) by any class derived from it. Thus, as shown in Figure 3a, if the functionality of "DynamicConduction" is required, either finite difference or response function approaches can be selected whereas if only the (lesser) functionality of "Conduction" is called for, both steady state or dynamic conduction classes can be selected. It is this facility that gives the EKS its flexibility: in terms of model building (support of theoretical variants), in terms of run-time features such as dynamic model substitution, in terms of support for program validation and in terms of program maintenance. Where different program architectures handle the same task in different ways, derived classes provide a means to incorporate that functionality while ensuring minimal impact on the rest of the system.

Behavioural inheritance also reduces coding and improves reliability since new classes gain access to their parent's code and so need only add the code for the extra functionality they provide. Extensibility is also assisted because new classes and variants of existing classes can be added without requiring any changes to other existing classes.

The EKS Principal Classes

The essence of the OO paradigm is the view that a program can be composed of independent objects communicating via messages. It was therefore well appreciated from the project's outset that the main challenge was the decomposition of building modelling into classes and subclasses and the definition of the properties of these classes in terms of their data members, behaviour and inter-relationship. It was also appreciated that in the

OO research and application field there was still no commonly accepted definition of the OO approach (Kim et al 1990). The project therefore struggled in its early stages to develop a rational basis for system decomposition and class identification.

It has been stated that "the most important concept in the object-oriented approach is data abstraction" (ibid.). To achieve this (and the subsequent data encapsulation), one approach is to undertake a data analysis of the target domains (here building physics and thermodynamic simulation). As the EKS was intended to be a model building environment and not a building model, such an approach was considered to be impracticable in terms of the resources available and the total number of possible modelling approaches. Instead a functional decomposition approach was adopted (Clarke et al 1989) which adhered to the following strategy.

- The domain (building energy modelling) was decomposed into its primitive functionality such as sun position tracking, conduction, convection, radiation, equation solving, polygon operations and the like. These are the functions, albeit at different levels of abstraction, found within all contemporary modelling systems. The initial research task was therefore to undertake a comprehensive functional analysis of existing models for building, plant and control system simulation (Tang 1990). The functions so identified are described elsewhere (Wright et al 1990 and Clarke et al 1990).
- A minimum level data requirement for each function was identified. For example, a one dimensional layer conduction function will require a set of thermophysical properties irrespective of the underlying mathematical model and so its minimum data requirement (in EKS terms) is a "Material" object (or strictly speaking a pointer to such an object) and a "Dimension" object. The matrix inversion function requires the topology and coefficient values (or the means to determine such values) irrespective of the inversion technique to be used. It is also worth noting that only EKS relevant functionality was considered. For example when considering sun-related processes the sun position is relevant while its angular rotation is not.
- At this stage the functions were associated with a physical class which logically would know about the context of the function. For example "Construction" logically knows about thermal resistance, while "Room" logically knows about shortwave response. More contentious perhaps, "Construction" may know about overall, reference U-values (which have prescribed surface overall resistance values) while only "Building" may know about overall, actual U-values because to construct this parameter requires knowledge of the properties and thermodynamic state of different entities (air volumes, surfaces, constructions, exposures and so on). It follows that class functions must relate only to the intrinsic data and properties of a class and not require the existence of, or assume data or properties of, another class. This ensures that a class will encapsulate only data which is truly pertinent to that class and that the data members of each class, as required to support its functions, can be guaranteed to be available at run-time to the object made from the class.
- Abstract classes are now identified by gathering together related functionality as implied by the functions of the physical classes. For example, the convection function of class "Room" requires surface area, hydraulic diameter and heat flow direction all of which are geometrical entities and so are gathered together into an abstract class "Polygon". This ensures that classes will not possess functionality where that functionality could be made more generally available by encapsulation within another class.
- Where a class has one or more domain theory functions (for example convection, shortwave response and occupant behaviour in the case of a class "Room") these functions are implemented as links to abstract classes containing the required functionality of the domain theory. The different formulations of any given domain theory can then be handled by derived classes: this serves to facilitate the handling of the multiplicity of domain theories, without incurring combinatorial explosion in the parent classes.

It is recognised that other grouping mechanisms could have been employed - for example grouping on the basis of computational intensity - but, because the EKS classes should be semantically acceptable to the modelling community, these were not considered appropriate for the EKS. Nevertheless it is thought likely that the foregoing process has given rise to the similar classes as would have resulted from a conventional data decomposition of real world entities#. The important aspect about this approach is that it immediately eliminates all aspects of the problem that are not related to the target domain and ensures maximum class reuse and extensibility. It also provides a mechanism to deal with the physical/ abstract mix so dominant in thermodynamic modelling.

The CEC Combine project which is attempting the development of an integrated data model has also used a technique of data decomposition of existing design tools with a subsequent OO encapsulation (COMBINE 1992).

Computational Support

As a simulation model development platform, the EKS will eventually encounter a wide range of systems represented by different mathematical categories, e.g. hyperbolic partial differential model for aerodynamic systems, parabolic partial differential model for diffusion systems, elliptic partial differential model for wave and vibration systems, ordinary differential models for lumped parameter systems and so on. This calls for an efficient and generic computational support facility which provides the range of mathematical tools and solvers required.

To achieve this, the EKS employs the following techniques:

- An internal mathematical representation format based on vector symbolism for the general system representation.
- A common mathematical interface for communication between the EKS classes via special transport classes.
- Dynamic data structures and sparse matrix techniques for system maintenance.

By these mechanisms access can be obtained to the EKS generalised solver classes which embrace a spectrum of analytical, numerical and statistical solution methods. The solver class mechanism has been designed to accommodate most of the currently available equation solving methods. This is achieved by using the C++ class inheritance mechanism in which a base solver class defines the solver interface by means of virtual functions which are then implemented in a set of derived classes. In this way a given solver can be selected at run time without affecting program structure.

Within the EKS Demonstrator two generic types of solver are provided each with several implementations. The first type includes conventional vector/ matrix operators and linear, nonlinear system solvers. These are mathematical tools usually available in software libraries and the like. The EKS makes no effort to improve the efficiency of such tools, only to provide the means by which the data they require can be encapsulated and supplied. For this type of solver the EKS provides a number of well known methods such as Runge-Kutta, Gear, etc. The second type comprises a direct solver which is based on an 'implicit row-wise sparse array' technique. This solver type receives, stores and processes only the non-zero entries of the vectors and matrices which define the problem and so is able to provide high efficiency by ensuring that minimal zero entries are created during the elimination process.

Class Taxonomy

The EKS classes are organised into a three dimensional class taxonomy which is designed to offer the functionality required to build alternative program architectures while guaranteeing security of use.

Figure 1 shows one plane of this hierarchy, the 'used by' hierarchy, which defines the relationship between the principal classes from which programs are actually built - rooms, constructions, sites, surfaces, layers, plant components, solvers and so on.

Orthogonal to this plane is the EKS inheritance hierarchy used to represent the alternative domain theories. For example Figure 3a shows alternative conduction theories used by the principal class "Layer".

Finally the third plane of the taxonomy represents conventional OO inheritance to enable code sharing even among classes of fundamentally different types (Figure 3b). This inheritance mechanism reduces coding and improve reliability. Inheritance means new, derived classes need only add code for the extra functionality they are providing since they automatically use their parent's code, presumably already 'validated' to some extent.

In this way the principal classes are related, while the orthogonal classes can be considered to be separate class hierarchy rooted on each base class.

Clearly there is no single best way to structure the classes, and any structure for a real-world system will be biased towards a particular view (such as energy modelling as here). However, an efficient structure should be adaptable for other areas (by using the more general base classes), and be easy to use and extend.

Theory Representation

At the present time many alternative algorithms exist for application in a building modelling context. One objective of the EKS is to ensure that future model users are not limited to only those out of date algorithms imposed by a particular system. To achieve this, the EKS offers a spectrum of theory classes with each theory covered by more that one algorithm. At program construction time, the EKS principal classes can then be coupled to any theory class as dictated by the intended application and the required level of complexity. In this way, the principal classes provide the interface to the theory variants. This is achieved by arranging that the alternative mathematical models of any given theory (such as conduction, air flow, room shortwave response, etc) are derived from a single base class and so possess the same interface.

As an example of the process, consider the "Conduction" class for which Figure 3a shows the derived class variants. The "Layer" class possesses

a function which returns the theory for conductive heat transfer. The actual code which represents this theory is not held in the "Layer" but instead is located in a class derived from the "Conduction" base class. In effect the "Layer" object's conduction function asks the "Conduction" object to return the appropriate theory.

In order to achieve mathematical consistency and minimise numerical error propagation, most building simulation programs adopt a coherent model discipline throughout the system, e.g. finite difference, finite element or response factor methods. Such an approach will give rise to a set of equations - algebraic, differential, linear or non-linear equations - which can be simultaneously solved by a variety of techniques. For those systems which use different mathematical approaches to represent different components (rooms, coils, etc.), each component is treated in isolation and linked to the other components by parameter passing. The solution of such systems is effectively by iteration.

Within the EKS a technique was required to represent the many different possible mathematical theories from which programs could be constructed. The requirement was that this be done in a manner which preserved the integrity of the original mathematical model in a mathematically consistent way.

One way to achieve this would be to provide representation schemes for all the different possible approaches - a finite difference layer and a finite difference conduction; a steady state layer and a steady state conduction and so on. Apart from being inelegant, this would give rise to the problems of interface complexity and combinatorial explosion (Clarke et al 1990), result in duplication of code/functionality and create unnecessary maintenance difficulties.

To investigate the system representation format which would best satisfy the needs of the generalised model building environment, a study into the system representation tools matrix methods was carried out. The outcome was the decision to base the EKS internal theory representation on a vectorised state-space equation method. The method was implemented using sparse matrix technique. Using this technique it is possible to represent most of the equation-based theories in the domain of building energy modelling in a consistent manner as vectorised state-equations. The details of this method are given elsewhere (Clarke et al 1992; Tang 1990). Theory in this vectorised form is encapsulated within the EKS via four special transport classes: "Equation_set", "Equation", "Coefficient" and "State_variable".

At present the EKS offers a range of alternative theories for each topic - conduction, convection, air flow, shortwave & longwave response, casual gains, control behaviour and so on. Furthermore these theories have been selected to represent the range of possible models from a low order approach - time invariant, user specified air changes for example - through intermediate order formulations, to (near) state-of-the-art methods. The purpose of this treatment is to demonstrate the capability of the EKS to accommodate a spectrum of modelling techniques and to support class interchangeability. In this way the EKS provides structured alternative theories in support of the alternative modelling approaches while guaranteeing security of use.

EKS in Use

The EKS can be used with or without an OODB. In each case the template building program, EKStb, the data definition program, EKSdd, and the program building and execution program, EKSrm, will appear similar. The only difference if that in the former case the various operations as controlled and coordinated by the OODB, whereas in the latter case the template is stored on disk and the program is built and executed in a conventional manner.

To demonstrate the EKS use, the system comes complete with several example "Context" classes. These define several possible programs progressing from trivial site analysis, through building models of intermediate complexity to state-of-the-art (in terms of theory) multi-zone building models with plant.

In the short term it is anticipated that the EKS will be explored by researchers who are concerned with advancing the state-of-the-art in modelling buildings and the environmental control systems they contain. In the medium term it is possible that software vendors might use the EKS to construct and maintain the theoretical 'engine' of future design support systems. In the longer term it is entirely feasible that end users such as Architects, Engineers and Energy Managers could use the EKS to achieve bespoke software solutions for particular problems.

The intention is that the EKS will improve researcher efficiency by placing model development on a task sharing basis. In addition, by allowing programs of different architecture to be built from a common set of classes, program integrity should improve and the validation process should be better served. In the longer term environments such as the EKS open up the prospect of radical changes to the design support process. For example, consider a design support system which allowed the definition of a design hypotheses by the graphical selection of component parts representing walls, windows, radia-

tors, shading devices, sensors, a sun type, a site type and so on. If these components were related to EKS classes then the designer is effectively constructing, in real time, a model which is matched to the problem. Given the functionality of the EKS classes it would a relatively simple matter to then arrange that the instanced object start to operate immediately on selection. By bringing together hypothesis manipulation and performance appraisal a real-time computer-supported design environment is enabled. This in turn would enable the application of simulation at the earlier stages of the design process where the potential benefits are greatest.

Conclusions and Future Work

To summarise the project's achievements:

- A system has been developed demonstrating the construction of a range of models at different levels of abstraction - from simplified performance assessors to state-of-the-art simulators.
- The project has proved the technical feasibility of applying the OO programming approach to a complex engineering domain.
- The EKS demonstrator is designed to be extensible and portable, and can operate in either stand-alone or OODB modes.
- The EKS concept makes state-of-the-art developments accessible to users without the need to handle source code, providing the means to allow designers to participate in the design tool creation process.
- And the project has demonstrated the benefits to be gained from effective collaboration between the IT and domain communities.

In developing the EKS, an attempt has been made to anticipate related developments in the field. For example, the emerging international Standard for the Exchange of Product data (STEP) is likely to have a major impact on building simulation in the medium to long term. Within the standard, real-world entities are described using the STEP language Express, which has many object-oriented features. The principal medium of exchange of data in the construction industry is still text and drawings, with some de facto industry standards for electronic drawings produced on CAD systems. At present, progress is being hampered by the lack of an agreed standard for describing the geometry and topology of a building, and the lack of standard 'libraries' of generic building components. When the STEP standard becomes established for building data, it is envisaged that software (currently being developed) will be used to translate C++ class definitions to and from Express entity definitions. It will then be possible to map a STEP building description in Express into an OODB data structure of C++ objects, with the potential for interface to other software packages such as CAD, lighting design and so on through the neutral format of STEP. This will go some way to removing one of the major barriers to effective interaction between simulation programs; namely the arbitrary and incompatible data structures currently in use. Clearly, an OO programming approach based on real-world entities, as in the EKS, will facilitate development in this area.

The model building tools provided with the EKS are fully functional, but do not have a particularly sophisticated interface. What is envisaged for the future would be a powerful user interface to the facilities provided. The main feature of this interface would be a browser type facility for examining and selecting the classes making up the program, together with help and guidance on the capabilities and validity of the various classes. There would also be facilities to display the resultant program graphically, and to modify the program by replacing/adding classes.

While the EKS demonstrator has provided proof of concept, several further developments can be identified in order to evolve the system to its full potential. These include a need to subject the EKS demonstrator to field trials to ensure that the conceptual basis is sound; the construction of a better interface to reduce the learning curve associated with the new technologies; the controlled validation of the system's features and functions; and as a major new infrastructure platform, its use to explore new simulation techniques and software engineering paradigms.

References

Clarke J A, K James and D Tang (1989) 'Simulation Methods Concerning Building Performance Prediction: A General Review' *EKS Project Paper* Energy Simulation Research Unit, University of Strathclyde, Glasgow.

Clarke J A, K James and D Tang (1990) 'EKS Prototype Status Review and Issues Arising' *EKS Project Paper* Energy Simulation Research Unit, University of Strathclyde, Glasgow, October

Clarke J A, D Tang, K James and D F Mac Randal (1992) 'Energy Kernel System' *Final Report for GR/F/07880* Science and Engineering Research Council, Swindon.

COMBINE (1992) 'The COMBINE Project' Various publications available from F Augenbroe, Faculteit der Civiele Techniek, Delft University of Technology, Delft, The Netherlands.

Hammond G and Irving I (1992) Validation Studies Appropriate to the Development of the Energy Kernel System for Building Environmental Analysis *Final Grant Report* Science and Engineering Research Council, Swindon.

Kim W and F H Lochovsky (1990) *Objected-Oriented Concepts, Databases, and Applications* ACM Press, New York.

ONTOS (1989) *Object Database Documentation* Ontologic Inc, Three Burlington Woods, Burlington, MA 01803, USA.

Stroustrup B (1987) *The C++ Programming Language* Addison-Wesley Publishing Company Inc.

Tang D (1990) 'The EKS Theory Representation' *EKS Project Paper* Energy Simulation Research Unit, University of Strathclyde, Glasgow.

Wright A J, P Charlesworth, J A Clarke, G P Hammond, A Irving, K A James, D Mac Randal and D Tang (1990) 'The use of Object-Oriented Programming Techniques in the UK Energy Kernel System for Building Simulation' *Proc. European Simulation Multiconference*, pp.548-52, Nuremberg, Germany.

Wright A J, T J Wiltshire and S Lockley (1992) *EKS Project Final Report* Science and Engineering Research Council, Swindon.

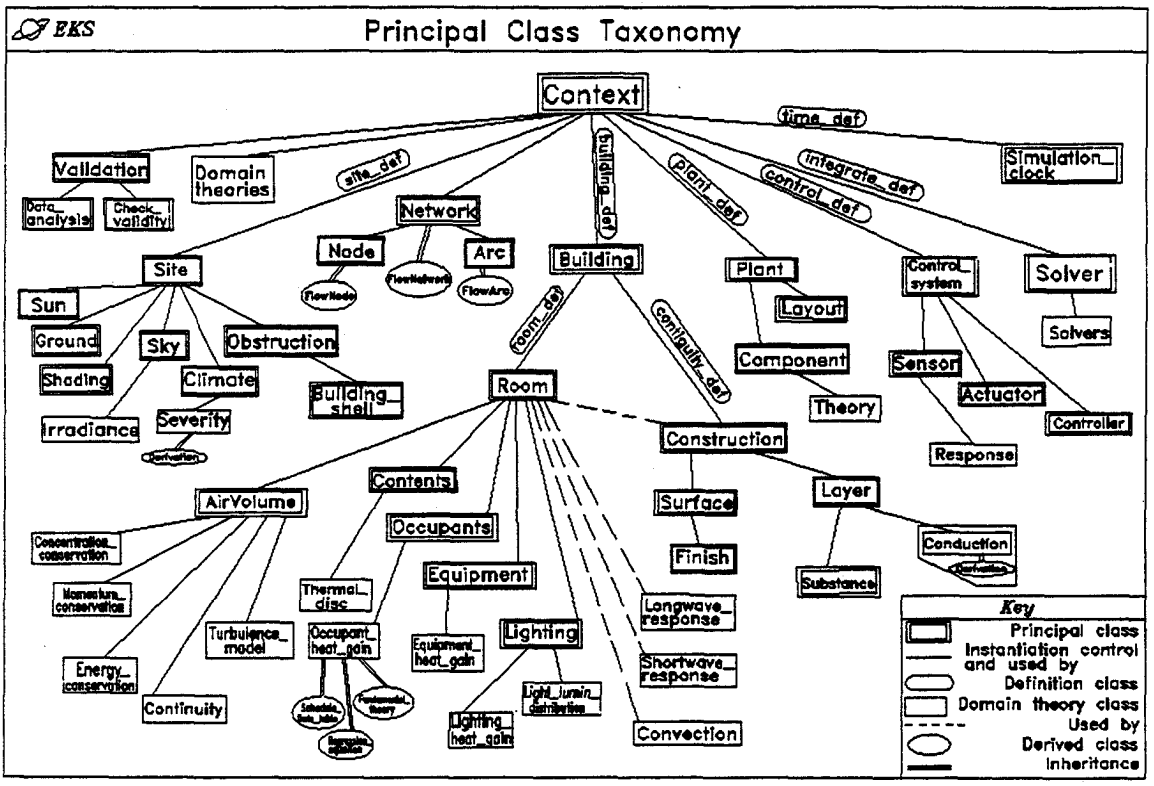

Figure 1 EKS Class Taxonomy

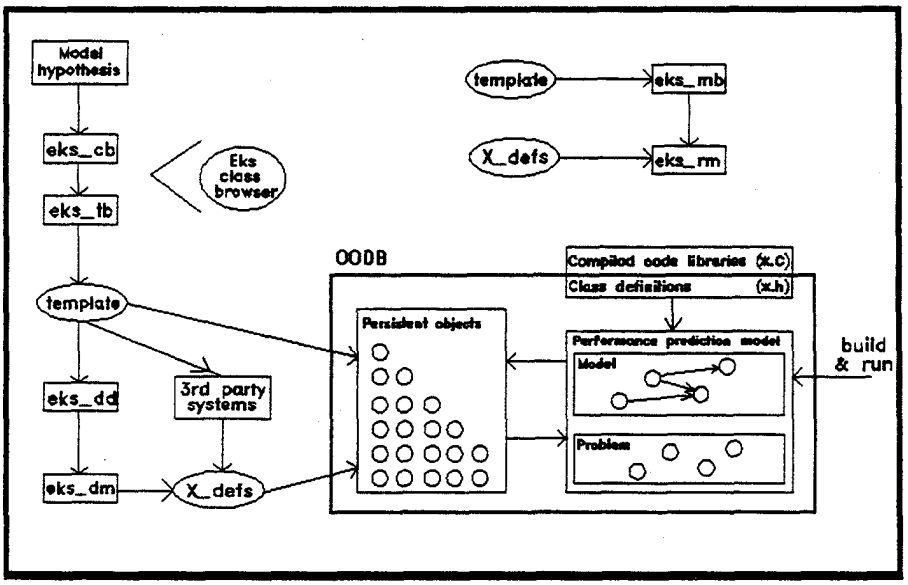

Figure 2 The Energy Kernel System

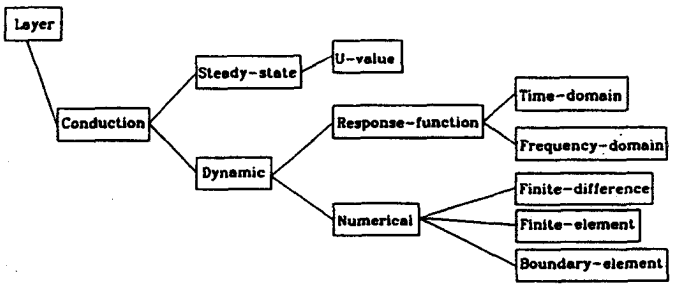

Figure 3a EKS inheritance

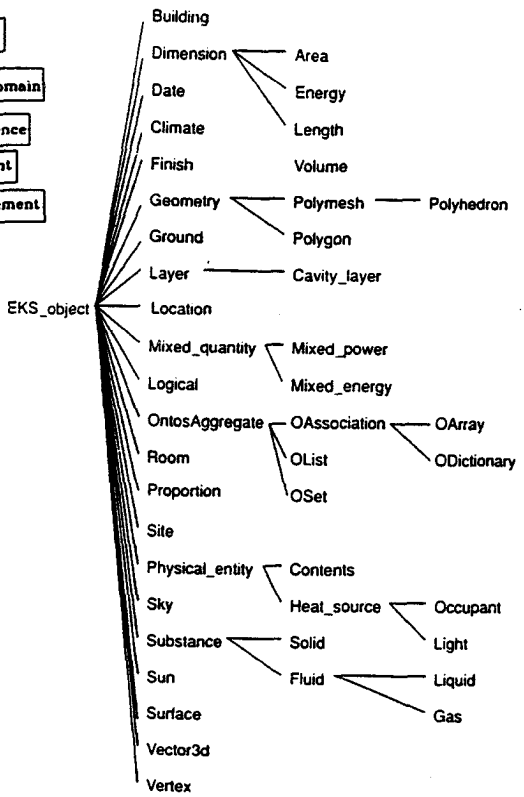

Figure 3b OO inheritance