A LOGIC-BASED FRAMEWORK FOR REPRESENTING ARCHITECTURAL DESIGN KNOWLEDGE

Kelvin Clibbon
Linda Candy
Ernest Edmonds

LUTCHI Research Centre
Department of Computer Studies
Loughborough University of Technology
Loughborough LE11 3TU
UK

This paper proposes a logic-based framework for representing and manipulating knowledge during Computer-Aided Architectural Design. The framework incorporates a meta-level architecture to represent declarative design knowledge and strategic knowledge used by the designer. It consists of an object layer, a design requirements layer and strategies for navigating through the design space. An extended first-order logic is described which has been used to represent some examples of architectural knowledge. This computational model is being implemented in KAUS (Knowledge Acquisition and Utilisation System), a general purpose knowledge-based system, founded in Multi-Layered Logic.

KEYWORDS: Design Knowledge, Strategic Knowledge, Multi-Layered Logic.

1. INTRODUCTION

Any given design process begins with a problem specification from which a set of requirements is derived. This set of requirements is not usually well defined at the beginning of the design process and does not remain static during it. For example, when designing a new block of flats, the initial requirements might be for a building that is low priced, cheap to maintain and easy to heat. These ambiguous requirements will evolve as design decisions are made, for example by basing the new block of flats on the design of an existing student accommodation block.

Architectural design can be formalised as an activity to derive an object model whose structure meets the design requirements. An object model is the non-procedural representation of a real object (or real problem) and is a representation of an object, possessing structural information to represent functionalities: that is, attributes, properties, behaviour and relations with other objects. The approach proposed in this paper is that design starts with a hypothetical object model. If this object model does not meet the requirements, then it is modified accordingly. In this sense, architectural design can be considered to be the process of identifying problems, specifying functionality and generating appropriate solutions with the support of some basic building blocks. In the approach described here, the object model is a declarative representation of the real object; however, it is not a static. Design is a dynamic activity the aim of which is to find this model. It is also a non-deterministic activity, in the sense that there are an infinite number of possible structures that an object model can be.

2. ARCHITECTURAL DESIGN

Architectural design is the translation of information in the form of requirements, constraints and experience into potential solutions, which are considered by the designer

to meet the requirements. A model of design should reflect the characteristics of the design process both at the cognitive and computational levels of design. The logic-based model discussed in this paper reflects the relationship between the representation of the design and the problem solving processes behind it.

There are three major problems areas associated with Computer-Aided Architectural Design (CAAD): first, the ill-defined nature of the design problem; second, the design knowledge representation and manipulation problem; and third, the design knowledge acquisition problem [1]. In the work reported below, an extended first-order logic is used as a means of representing architectural design knowledge (see section 3.2 for a description of the method).

Most conventional knowledge formalisms provide a computational representation of the design process which does not reflect the designer's strategies. Our primary goal is to facilitate the representation of the designers strategies by means of a logic-based framework. In addition, a secondary aim is to provide an analytical computational tool for addressing CAAD problems.

2.1. STRATEGIC KNOWLEDGE

Strategic knowledge is concerned with the control decisions made during the conceptual design phase as the object model evolves. From a non-epistemological perspective, knowledge is information that can be used to perform some intelligent task. Strategic knowledge is knowledge used by an agent to decide what actions to perform next, where actions have consequences that are external to the agent [2]. Strategic knowledge has more recently been defined as the knowledge used for deciding the course of action when there are conflicting criteria [3]. At a high level of abstraction, strategic knowledge is concerned with knowledge about the approach or problem solving method to be used. At a lower level, strategic knowledge might be concerned with the decisions necessary for a single action [4].

Strategic knowledge is used by the designer to decide what actions to perform in a given situation, where actions are considered to have observable consequences. The more general term, control knowledge, refers to knowledge used to decide what to do next. Actions can have consequences that are observable in the world outside of the agent. An action may be the firing of a rule or an operator. Search-control knowledge is used to choose internal actions that increase the likelihood of reaching a solution state and improve the speed of computation.

Strategic knowledge is distinguished from substantive knowledge of a domain, i.e. knowledge about what is believed to be true in the world. Both substantive and strategic knowledge underlie expertise in many domains. For example a robot uses substantive knowledge to recognise and interpret situations in the world (an obstacle in its path) and strategic knowledge to decide what to do (to go round or over etc.). In general, substantive knowledge is used to identify relevant states of the world and strategic knowledge is used to evaluate the utility of possible actions given a state.

Substantive knowledge is represented explicitly in knowledge bases. Strategic knowledge can be represented both implicitly and explicitly. Implicit behaviour has been achieved using implementation level primitives such as numeric priorities. MYCIN's certainty factors are an example of implicit strategic knowledge [5]. A more explicit representation of strategic knowledge is achieved via control blocks, which are sequential procedures attached to rules. BB1's blackboard architecture is an example of explicit strategic knowledge [6]. A key research issue is how can strategic knowledge be represented and what kinds of strategic reasoning can be expressed during the design process?

Knowledge systems make control decisions at the implementation level. One such decision is conflict resolution, where the choice of which rule(s) to fire from a competing set is made. Strategic knowledge, by comparison, is defined at the knowledge level in terms of the effects on the observable behaviour of the agent. This occurs without reference to the internal symbolic-level organisation of the knowledge system [2]. Substantive knowledge is used to describe or model the domain of endeavour. It classifies entities in the knowledge base in terms of the relationships, attributes and functions between them. The procedure for deciding what to do when operating on the entities in the domain entities, is the strategy. This distinction between strategic and substantive knowledge is not the same as the

difference between procedural and declarative knowledge representation, which is a symbolic-level distinction.

The knowledge elicitation task is inherently difficult for strategic knowledge. Strategy is often tacit and when it is explicit it is not easy to describe it in a form which may be directly translated and implemented in a knowledge-based system [7]. Designers can specify how to classify situations in the world without worrying about the mechanism by which the specifications are interpreted. Specifying knowledge that alters the order and choice of actions, involves building a computational model of the design process.

2.2. REPRESENTING STRATEGIC KNOWLEDGE.

The most straightforward method of representing strategic knowledge is through the use of a procedural formulation. A procedural representation of a strategy specifies the sequence of steps that the system should take. For instance, a procedure might take the form, "do action A then B, then if condition X is true do action C else do action D."

Strategies have been encoded as procedures in programming languages, such as Lisp. In some rule-based systems, for example, Cooper and Wogrin [8], task variables serve as control markers in the rule language, which direct the rule interpreter to execute modular sets of rules as subroutines. S1 [9], for example, integrates procedural control in its representation language as "control blocks", which invoke rules. Another approach to procedural control is to build transition networks [10] or decision graphs that graphically specify the control flow of the program.

Although procedural formulations of control cannot be used to implement a strategy, they can represent it implicitly. The strategic knowledge that underlies the specific sequence of actions is hidden in the procedural representation which specifies what to do as a procedure, but not why. Thus, if the objective is to represent the reasons underlying a strategy, an architecture capable of being driven by this knowledge is required.

Metarules guide the use of knowledge and decide what rules and methods are to be applied. There are essentially two types of metarules: pruning and re-ordering metarules rules. Pruning metarules exclude particular rules from consideration. In terms of the goal tree, this amounts to a decision not to explore a given branch and constitutes a judgement on the overall utility of a rule, as to whether it is of any use in a specific context. The other type of metarule encodes knowledge on the relative importance of rules. At the implementation level the metarule acts to reorder rules relevant to some goal before involving them. In rule-based systems the order in which rules appear in the knowledge base influences the inferencing process. If, for example, one of the requirements for the design of a house was for a spacious kitchen, to the detriment of living room space, then the following is a simple example of a metarule that reorders the rules:

Metarule 1
IF (There are rules which are relevant to living room space)
AND (There are rules which are relevant to kitchen size)
THEN (Do the latter before the former).

Metarule 1 focuses inferencing on rules that relate to the size of the kitchen and hence, ensures that conflict resolution is least likely to be needed in that aspect. Metarules that reorder amount to a less drastic decision as regards the goal tree. The branches are restructured rather than pruned. Different types of meta knowledge can thus be represented, whether they are dependent on exhaustive search, reordering of goals or pruning of goals. This gives rise to a pattern of search which is not 'blind' but which is guided by heuristic knowledge. The seminal work on the explicit representation of control knowledge metarules was carried out in the MYCIN experiments [5,11,12].

2.3. SUPPORT FOR CAAD

The use of metarules to represent strategic knowledge is not restricted to one level of meta knowledge, but can be extended to indefinite levels. Metal level rules can select different types of inference mechanisms (such as forward- or backward- chaining) at appropriate points in the search process. The purpose of representing strategic knowledge with metarules is to tell the system which part of its substantive knowledge to apply in a given situation. In choosing among observable actions and controlling search, metarules are useful if the objective is to control the order and manner in which inferences are made.

Research by the authors reported elsewhere [13], has highlighted the nature of support needed during design. The requirements are for support during knowledge exploration and in evaluating the object model. Knowledge used in the act of designing is dynamic and in order to apply it effectively it must be evaluated during the process. At the same time however, if the design is specified too strictly, then the freedom of the model building activity is restricted and there is no room for innovation. There is clearly a trade-off between the freedom and the efficiency of design.

The designer will develop and use strategies to evaluate the object models that evolve during conceptual design. Strategic knowledge would be used by the designer to identify and formulate the design problem. Having established the requirements, the designer will require strategic support in exploration of the design, particularly as new concepts emerge. This research is concerned with logic-based support, providing tools to guide the designer through iterative model building, model analysis, model evaluation and model modification. It is hypothesised that in having a sound theoretical logic-based foundation, support systems can check for consistency and correctness in the knowledge-base during model building.

3. A LOGIC-BASED FRAMEWORK FOR CAAD

The logic-based framework proposed in this paper utilises layers which consist of nodes. Nodes in the object layer contain descriptions of object models. Nodes in the requirements layer consist of sets of requirements for these objects. Interaction between the two layers can be represented by links. Strategies are needed to support navigation within and between the object and requirements spaces.

3.1 THE MULTIK MODEL

Researchers on the MULTIK (Multi-Layered Knowledge-Based Interfaces for Professional Users) project, are developing a logic-based architecture, in order to support design. A declarative representation of the object model is represented independently from a declarative representation of the requirements. Links between nodes in the layers represent which set of requirements is related to which object model. Static aspects of the design are concerned with the object model and requirements domain knowledge. The strategic knowledge relating to the steps to be taken during the design process are the dynamic aspects. A meta-level architecture facilitates the representation of both the static and dynamic aspects of the conceptual design process. Strategic knowledge is necessary to determine in which layer the next step of the design process is to be made. It is necessary to perform object-level reasoning, with knowledge in the object level, as well as meta-level reasoning, about the knowledge in the object level. According to the meta-level architecture (see figure 1) there are essentially three types of strategic knowledge used by the designer:

(1) strategies used between the layers;
(2) strategies used within each layer; and
(3) strategies used in the nodes, in each layer.

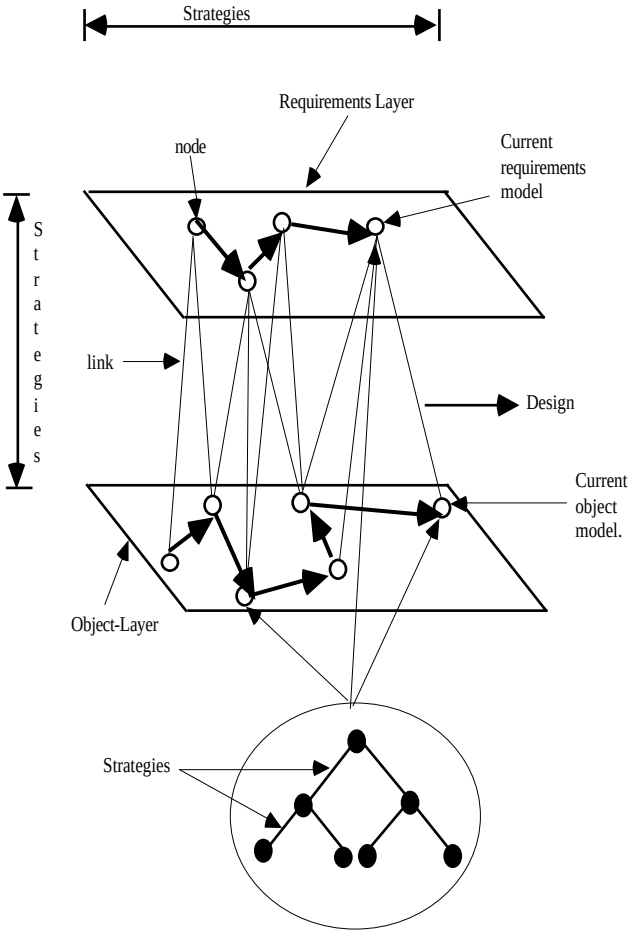

Figure 1 - A Meta-Level Architecture for Design

3.2. Multi-Layered Logic

A Multi-Layered Logic (MLL) [14] knowledge representation, capable of automatic consistency checking, representing complexity and dynamic model building has been used to model architectural knowledge. It is based on the concept of hierarchical data abstraction and is an extension to first-order logic, with data structuring capabilities. A data structure, either human specified or automatically generated, is defined as a set based on ZF (Zermelo-Frankel) set theory. MLL has a set of primitive constructs including 'element-of', 'component-of', 'power-set-of', 'product-set-of', 'union-of', 'intersection-of' and 'pair-of' relations, which are used to represent the object model and its requirements. These data structures can be included as terms in predicates and are defined mathematically in terms of primitive structures according to composition rules. For example, the symbol ' ' represents the power set operator, where *D means the set of subsets of D (excluding the empty set).

The syntax of MLL predicates has been expanded from First Order Logic to include a domain of each variable explicitly in the prefixes. For example, [AX/new_building](expensive X) is the same as the first-order expression [VX]((new_building X) → (expensive(X)). Both mean new_building is expensive, but in the former expression, new_building denotes a set of new buildings. To accommodate the

modified syntax, an ordinary inference algorithm (resolution) has been modified in order to check for equality between data-structures. See Ohsuga [15] for details.

A MLL predicate can also include closed formulae (a formula without any free variables) as the term. For example, in the following [AX/D](P,---,X,---,[AX/C](R,---,X,---)), the evaluation of the inner predicate R can be performed independently from that of the outer predicate P. The set of predicates that do not contain any predicates as terms are located in the object-level, while those that contain object-level predicate(s) as the term(s), belong in the level above the object level, the meta-level.

4. A REALISATION OF THE MULTIK MODEL

A working prototype version of the MLL model is now being developed in KAUS (Knowledge Acquisition and Utilisation System). KAUS has a mechanism to handle meta-level knowledge. MLL rules are indexed and partitioned into several sets (worlds). It is possible to have independent local worlds to form separate knowledge sources. KAUS utilises built-in predicates, called procedural type atoms (PTAs), to specify and change the worlds used during the inferencing process. A meta-level inferencing process can get information from the object level. Hence inferencing in one level is defined in the next highest level. For further details on MLL and KAUS refer to [14, 15].

This section serves to demonstrate the MLL foundations of KAUS, its declarative power in terms of object and requirement level hierarchies and its meta-level capabilities. The architectural examples described provide accessible illustrations of the major substantive and strategic aspects of design but are not intended to be exhaustive.

4.1. KAUS

KAUS was developed at the University of Tokyo. It has evolved since the mid 1970s into a logic programming system and can be regarded as an extension to Prolog [16]. KAUS is based on Many Sorted Logic, where sort hierarchies are described in terms of set relationships, in which each set represents the sort of a set of entities. For example, the sort ‘houses’ represents the set of houses, while ‘flat’ and ‘terraced house’ can be thought of as subsets of the set ‘houses’ - the subsorts of the sort houses. So if a penthouse suite is designated as a flat and a family house as a terraced house, then the penthouse suite and family house are members of the sorts flat and terraced house respectively.

Differing from Prolog, the clausal representation of rules and facts in KAUS are arbitrary AND-OR logical formulas. A KAUS clause has a prefix in which the quantification and type declaration of variables appearing in the clause may be explicitly described. For example, a sentence “If a person Z is a parent of a male X and Y, and X is not equal to Y, then X is brother of Y.” is represented in KAUS and Prolog as follows:

KAUS: [AX,Y/male][AZ/person](|(brother X Y) ~(parent X Z) ~(parent Y Z) ~($ne X Y)).

Prolog: brother(X,Y):-male(X), male(Y), person(Z), parent(X,Z), parent(Y,Z), not(X==Y).

[AX,Y/male][AZ/person] is the prefix part of the clause. This can be read as ‘for all X and Y of type male and for all Z of type person’. (| (brother X Y) ~(parent X Z) ~(parent Y Z) ~($ne X Y)) represents an OR formula which is logically equivalent to ‘(parent X Z) (parent Y Z) ($ne X Y) → (brother X Y)’. ‘|’ denotes the or-connective and ‘~’ is the logical negation symbol.

KAUS, unlike Prolog, has the expressive power to describe rules concerning sets. In KAUS the syntax of Prolog is expanded to include the domain of each variable explicitly in the prefix. Consider the following examples:

All houses have a kitchen.
Terraced house and bungalow are types of houses.
Houses are a subset of accommodation.

These can be expressed in KAUS as follows:
[AX/houses](have(X kitchen)).

```
!ins_e houses terracedhouse bungalow.
!ins_e *accommodation houses.
```

In the above, 'houses' refers to the set of houses and 'A' is the universal quantifier (a reserved symbol). Symbols beginning with capital letters relate to variables, and those beginning with lower-case letters represent predicates or basic objects. !ins_e defines the component_of relation. *accommodation denotes the power set of the set 'accommodation'. The first example is a predicate sentence (a rule). The second and third examples express relations between sets and objects.

In KAUS it is possible to define type hierarchies, which can be interpreted semantically as a representation of ISA structures of objects and PART-WHOLE structures of objects. For instance, houses can be classified into flats and terraced houses and so on. In addition, houses are composed of kitchens, living rooms, sleeping rooms and bathrooms. The ISA and PART-WHOLE hierarchies are described in terms of set theory relationships among objects. The description of ISA and PART-WHOLE hierarchies of objects can be classified into two classes. One is concerned with the hierarchies of objects (in the object level), and another with hierarchies of objects in the meta-level. Objects in the meta-level are program clauses. Hierarchies in the meta-level describe the classification of program clauses. Figure 2 is an example description of part of an object level hierarchy for the house domain.

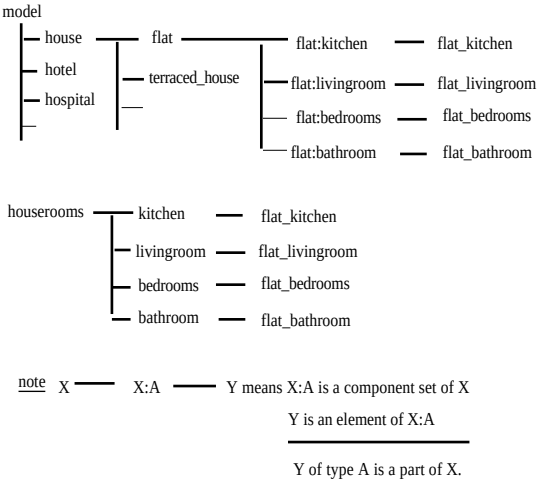

Figure 2 - An Object Level Hierarchy for the Architectural Domain

The above structure is described as follows:

```
a1). !ins_e *model                  house, hotel, hospital;
a2). !ins_e *house                  flat, terraced_house;
a3)  !ins_e *houserooms             kitchen, livingroom, bedrooms, bathroom;
a4)  !ins_e flat:kitchen            flat_kitchen;
a5)  !ins_e flat:livingroom         flat_livingroom;
a6)  !ins_e flat:bedrooms           flat_bedrooms;
a7)  !ins_e flat:bathroom           flat_bathroom;
```

Note that a1 through to a3 describe hierarchies of generic objects. The statements a4 to a7 describe the specific PART-WHOLE structure for a flat. For example, in a1, house, hotel and hospital are described as subtypes of a supertype model. In a2, flat and terraced

house are defined as subtypes of a supertype house, and so on. Figure 3 gives an example description of a hierarchy of objects in the meta-level.

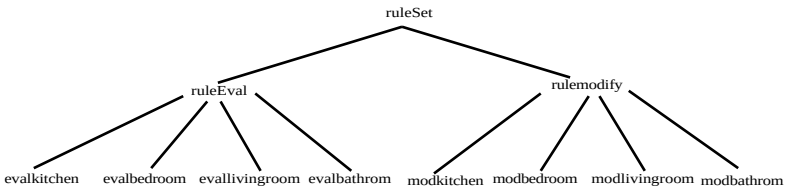

Figure 3 - A hierarchy of objects in the Meta-Level

The example above aims to classify house design rules into evaluation rules and modification rules. The evaluation and modification rules are classified into three rule sets with respect to the objects in the kitchen, bedroom, living room and the bathroom. If, for example, the objects in the kitchen were modified, then they will be so according to the associated modification rules, and evaluated according to the relevant evaluation rules (i.e. rules associated with positions of the sink, doors and so on).

4.2. REASONING WITH STRATEGIC KNOWLEDGE.

Program clauses in KAUS can be classified according to their utility, availability and relevance to specific domain tasks. This results in a hierarchical categorisation of program clauses. A set of clauses which belong to a particular category in the hierarchy may be regarded as a description of a certain inference world. These inference worlds constitute strategies used by the designer during design. In terms of KAUS, strategic clauses can be built to change the current inference world. For example, with a flat, the emphasis might be on living and sleeping accommodation. The kitchen might receive a lower priority, in terms of size, than when designing a terraced house. Consider the following simplified floor plan of a flat and the ground floor of a terraced house (see figure 4).

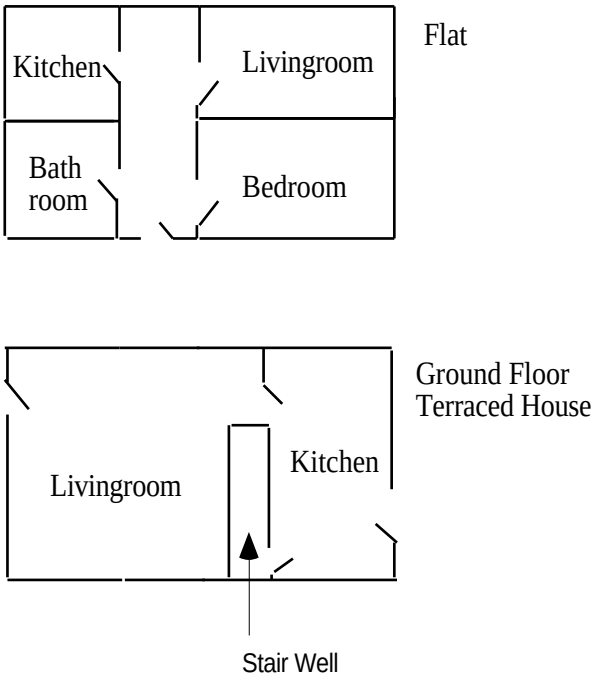

Figure 4 - Design of Flat or Ground Floor of Terraced House

The designer of a flat is attempting to make the best possible use of the floor space available. If the requirements change from a flat to a terraced house, with the same floor space, the layout of the ground floor might be depicted as in figure 4. To illustrate how metarules can be used to represent the designer's strategies for changing inference worlds, imagine that the designer is manipulating the objects associated with the flat. The underlying KAUS representation is as follows:

```
r1:(|(aflat DesignIs) ~(kitchenArea), ~(livingroomArea)).
r2:(|(aterracedhouse DesignIs) ~(kitchenArea), ~(livingroomArea)).

f1:(numberBedrooms 1).
f2:(kitchenArea small).
f3:(numberBedrooms _).
f4:(kitchenArea large).

m1:(know numberBedrooms{\r1, \f1}).
m2:(know numberBedroms{\r2, \f3}).
m3:(know kitchenArea{\r1, \f2}).
m4:(know kitchenArea{\r2, \f4}).
m5:([APredicate/tuplej (| believe Predicate) ~(know Fact) ~($scopeKUs [Fact]) ~($call Predicate)).
```

Metarules m1 to m4 describe what is known (meta knowledge about the facts). For example, m1 describes that the number of bedrooms for the flat is one. The metarule m5 is a generic rule to derive what the object model looks like, given the rules r1 and r2, facts f1-f4, and metarules m1-m4. Metarule m5 has a conditional ($scopkeKUs [X]), which restricts the clauses available to ($call Predicate) to those known by Fact. With this setting,H the

knowledge base is queried to find out what type house (flat or terraced house) the design tends to - i.e. (believe < X Design >)?, X will be instantiated to aflat. If the design requirements were for a terraced house, the designer could be prompted to change the number of bedrooms to be included and the kitchen area.

In the examples above, the object level hierarchy represented in figure 2 is typical of some substantive knowledge in a node of the object-layer in figure 1. The same representation is used for nodal knowledge in the requirements layer. The hierarchy of meta-level objects in figure 3, would be used by the system to support the designer in exploring the design space. Metarule m5 is an example of some strategic reasoning on the behalf of the requirements layer, where the leading object-layer model (node) of figure 1 is checked for consistency against the leading requirements layer node.

4.3. TOWARDS LOGIC-BASED SUPPORT FOR CAAD

In terms of the proposed logical framework, strategic knowledge is concerned with how best to determine the next step or action to be taken by the designer, during the design process. This might amount to a set of rules or heuristics used by the designer in certain situations. For example, if the object layer is densely populated with object models that fit the latest set of requirements, how can the designer be supported in exploring the object space? In addition, how can support for making decisions about which object models to include in a candidate set be provided. Search-control knowledge could be used to constrain browsing to those areas of the object layer that are likely to contain the best solutions.

Multi-Layered Logic provides the means to explicitly represent strategic design knowledge. Strategic knowledge may be provided at a high level, for example representing a high level hint where the designer takes a design decision based on some strategic comments from the project leader. A design support system should actively assist the designer in exploring the design state space throughout the design process. The system should also provide support for context specific inference based on the design decisions, constraints, and the requirements defining the design problem together with physical laws, heuristics and other domain knowledge relating the parameters of the design [17].

5. DISCUSSION

It is generally acknowledged that a detailed systematic analysis of existing design objects is beneficial not only for their improvement but also for the novel design of objects [18]. Whilst the design process can be enhanced by appropriate search algorithms, the automatic generation of complete solutions is not likely to be feasible [19]. The arrival of knowledge-based techniques and the goal of achieving concurrency are playing a role in altering the state of what is thought to be desirable and achievable in terms of support for design as a whole.

With meta-level structuring, the computational system may perform in a less primitive and more sophisticated way, but the underlying model tends to be less clear, especially when dependant on domain specific meta-level concerns. Implicitly meta-level models are based on the hope that a limited number of domain independent meta-levels with corresponding inferencing strategies and knowledge will be discovered. A serious concern with these meta-level models is that effective computational models of a complex task will have to cope with the problem of organising and effectively applying large amounts of knowledge. Therefore, problems of knowledge acquisition, representation and retrieval in knowledge-based systems still pose major challenges if computational systems are to obtain an autonomous capability for design [20].

Design is likely to be enhanced when designers are provided with the capability to explore and expand the design space. Support for CAAD in the MULTIK model involves the addition of new variables and changes to the rules that govern the design process both within and between the object and requirements layers. Exploration of the design space would be supported if the underlying relationships between substantive and strategic knowledge are explicitly represented in a logic-based computational model of design.

6. CONCLUSION

A logic-based framework for design has been proposed which incorporates a meta-level architecture to explicitly represent declarative design knowledge and strategies used by the designer. The KAUS logic programming language has been described with reference to some architectural examples. The MLL which underpins this language has been proposed as a suitable formalism for representing the object model, the design requirements and the strategic knowledge used by the designer.

7. ACKNOWLEDGEMENTS

This work was partly funded by the Science and Engineering Research Council (Grant reference GR/J43769). We wish to thank the members of the Vehicle Concepts Department at Lotus Engineering, UK, for their support and encouragement, in developing the MULTIK model. Our gratitude also to Professor Ohsuga of RCAST at Tokyo University, for his input on MLL.

8. REFERENCES

[1] Liu, Y. Schematic-Designer: a Knowledge-Based CAD System for Schematic Design in Architecture, Design Studies, 1991, Vol 12, Part 3, pp.151-167.

[2] Gruber, T., The Acquisition of Strategic Knowledge, Academic Press, 1989.

[3] Tunez, S., Bosch, B., Bienvenido, F., Marin, R., Soria, F., Canabate, F. and Mira, J., Strategic Knowledge in an Expert System for Agriculture, Cybernetics and Systems, Vol 23, Parts 3-4, 1992, pp.401-415.

[4] Al-Dhaher, K. and Harandi M., Representation of Strategic Knowledge, in Proceedings of Industrial and Engineering Applications of AI and Expert Systems Conference, 1991, Volume 2, pp.482-490.

[5] Buchanan, B. and Shortliffe E., Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, 1984.

[6] Hayes-Roth, B., A Blackboard Architecture for Control. Artificial Intelligence, Vol 26, Part 3, 1985, pp.251-321.

[7] Lanzola, G. and Stefanelli M., A Specialised Framework for Medical Diagnostic Knowledge-Based Systems, Computers and Biomedical Research, Vol 25, Part 4, 1992, pp.351-365.

[8] Cooper, T. and Wogrin, N. Rule-Based Programming with OPS5. Los Altos, CA:Morgan Kaufmann, 1988.

[9] Erman, L., Scott, A. and London, P. Separating and Integrating Control in a Rule-Based Tool, Proceedings of the IEEE Workshop on Principles of Knowledge-based Systems, 1984, pp.37-43.

[10] Georgeff, M. and Bonollo, U., Procedural Expert Systems, Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 1983, pp.151-157.

[11] Clancey, W., The Epistemology of a Rule-Based Expert System - A Framework for Explanation, Artificial Intelligence, Vol 20, Part 3,1983, pp.215-251.

[12] Clancey, W., From GUIDON to NEOMYCIN and HERCULES in Twenty Short Lessons: ORN Final Report 1979-1985, AI Magazine, Vol 7, Part 3, 1986, pp.40-60.

[13] Candy, L. and Edmonds E., Artefacts and the Designer's Process: Implications for Computer Support to Design, Revue Science et Techniques de la Conception, Vol 3, Part 1, 1994, pp.11-31.

[14] Ohsuga S. and Yamauchi H., Multi-Layer Logic - A Predicate Logic Including Data Structure as Knowledge Representation Language, New Generation Computing, Vol 4, 1985, Special Issue on Knowledge Representation, pp.403-439.

[15] Ohsuga S., A New Method of Model Description - Use of Knowledge Base and Inference, in CAD System Framework, K. Bo and F. Lillehagen (Eds.), North-Holland, 1983, pp.285-312.

[16] Bratko, I., Prolog: Programming For Artificial Intelligence (2nd Edition), Addison-Wesley, 1990.

[17] Smithers, T., Conkie, A., Doheny, J., Logan, B., Millington, K., and Tang, M., Design as Intelligent Behaviour: An AI in Design Research Programme, Edinburgh University, Department of AI Research Paper, Number 426, 1989.

[18] Katai, O., Kawakami, H., Sawaragi, T. and Iwai, S., A Knowledge Acquisition System for Conceptual Design Based On Functional and Rational Explanations of Designed Objects, in J. Gero (ed.), AI and Design'91, Butterworth-Heinemann, 1991, pp.281-300.

[19] Pahl, G. and Beitz, W., Engineering Design, London: The Design Council, 1984.

[20] Coyne, R. and Subrahmanian, E., Computer Supported Creative Design: A Pragmatic Approach, in J. Gero and M. Maher (eds.), Modelling Creativity and Knowledge-Based Creative Design, Lawrence Erlbaum Associates, 1993, pp.295-327.