The Edinburgh Designer System: An Architecture for Solving Ill Structured Problems

Brian Logan, David Corne

Department of Artificial Intelligence
University of Edinburgh
Forrest Hill
Edinburgh, UK

and

Tim Smithers

Artificial Intelligence Laboratory
Vrije Universiteit Brussel
Brussels, Belgium

Abstract

In this paper we discuss the role of problem redefinition in solving ill structured problems. We argue that conventional CAD systems are ill equipped to deal with problems where both the requirements and the solutions are incomplete and/or inconsistent. We outline an alternative model which views problem solving as an *exploration* process and describe a design support system, the Edinburgh Designer System (EDS), based on this model. The hybrid ATMS-blackboard architecture employed by EDS is briefly described and we illustrate how it supports the solution of ill structured problems. Finally, we discuss its relationship to similar systems and outline some of the outstanding research problems arising from our experience with the architecture.

DAI Research Paper No 595, August, 1992

This is a revised version of a paper presented at the 10th European Conference on Artificial Intelligence, Vienna, Austria, 3–7th August, 1992

1 Introduction

Ill structured problems have been largely ignored in the AI literature. However many real world problems fall into this category and the ability to solve such problems is an important part of any agent's problems solving capabilities. A problem is ill structured if it is not well structured, that is, if there is no complete and consistent problem statement when problem solving begins [13]. For example, in deciding whether to buy a house or a car or where to go on holiday, one's decision may be influenced by a wide range of factors, many of which may only emerge when the various alternatives have been considered in detail, and the initial requirements may have to be revised or abandoned in response to unexpected difficulties or opportunities. In this paper we concentrate on a particular kind of ill structured problem, design problems. Design problems are interesting because part of the problem definition is given in the form of requirements the designed artefact or process must meet. However these requirements are typically incomplete and/or inconsistent and considerable effort is often required to formulate the problem.

In attempting to solve such problems designers explore the space of possible solutions trying out decisions and investigating their consequences in a way which develops both the solution and the problem requirements [16]. This process of exploration frequently results in inconsistencies between the designer's best guess at a solution and the problem requirements or between the current decision and the existing partial solution. In some cases such inconsistencies can be resolved by modifying the proposed solution. However, if no solution which meets the problem requirements can be found, then the problem requirements are themselves inconsistent (relative to the knowledge of the designer) and the problem is insoluble. To 'solve' such a problem it is necessary to redefine it. For example, the problem of designing a house with a floor area of at least $100\,\mathrm{m}^2$ costing less than £10,000 is probably insoluble given existing construction techniques, prevailing statutory requirements etc. without some redefinition of the concept of 'house'.

When an inconsistency arises, it is unrealistic to expect the designer to restore consistency immediately. Indeed it is often only be exploring the implications of the conflicting design decisions and requirements that a choice can be made about which decision to abandon or which requirement to relax and in many cases considerable work is required before the conflict can be resolved.

2 Solving Ill Structured Problems

Previous attempts at supporting this process using computers have tended to adopt one of two approaches: deriving the characteristics of a design given a description of the solution; and generating a solution or part of a solution given the design goals and requirements. Such systems can be effective when the problem and solution are well defined. However they are typically incapable of operating with incomplete or inconsistent information. If the problem requirements are incomplete or inconsistent then no consistent solution can be found. Similarly, if the proposed solution is incomplete or inconsistent no consistent evaluation is possible (assuming there is enough information to produce any evaluation) and the question of whether the solution meets its design goals is meaningless.

The response of most systems to these problems is to ignore them. Existing CAD tools are ill suited to the process of exploration described above, tending to focus on individual design characteristics such as cost, structural stability etc. With no overall framework to integrate the individual tools, the problem of inconsistencies between criteria is not addressed and they provide no support for the process of problem redefinition. The designer is left with the task of reformulating the problem requirements and/or the solution and trying again.

We can therefore identify two important aspects of the support required when carrying out design tasks which are missing from existing systems: *strategic* support for the exploration of the space of possible designs and the overall control of the design process; and *tactical* support for the construction and evaluation of alternative design solutions and part solutions.

At the strategic level the system should provide support for the generation of alternative solutions to particular sub-problems, the comparison of alternative solutions, the selection of the most promising candidates for further development and revising and refining the requirement description. At the tactical level the system should provide support for context specific inference based on 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. In addition the system should, where possible, provide assistance in solving particular design problems, drawing on the large amounts of knowledge encoded in design handbooks, codes of practice and in the expertise of individual designers. Of particular concern are inconsistencies which arise in attempting to satisfy multiple goals, for example when the derived performance fails to meet the design requirements or when no way of achieving the design requirements can be found.

3 The Edinburgh Designer System

In the remainder of this paper we briefly describe the architecture of a design support system, the Edinburgh Designer System (EDS), which embodies the model of design support outlined above. EDS consists of four principal subsystems: knowledge representation, inference, consistency maintenance and context management.[1]

Domain knowledge is represented in a structured knowledge base containing the definitions of domain objects, called *module classes*, related by *part_of* and *kind_of* relations. Each module class declares a set of parameters, variables and constraints which define a particular class or type of object. For example, in the house design problem mentioned above, we would typically have module classes containing information about houses, rooms, walls etc. Relevant parameters for a *house* module class might include the floor area of the house A, the total cost of the house C and the cost/m^2 floor area U, together with constraints expressing dependencies between these parameters, such as $A \times U = C$. Instances of these module classes form the *design description* which contains the requirements, domain knowledge and design decisions relating to a particular problem.

This knowledge of the domain is used by a series of inference engines or *support systems* to infer the consequences of the designer's decisions about problem requirements and possible solutions. There are two main kinds of support system: general purpose support systems and special purpose support systems. General purpose support systems use constraint propagation techniques to infer *necessary* consequences of the designer's decisions. In general these sub-systems are invoked automatically and can be allowed to proceed relatively unhampered by the designer with the sole requirement that they should refrain from re-deriving information already known about the design. Special purpose support systems, on the other hand, advise on how to solve particular design problems. In general these sub-systems offer *possible* consequences, often in the form of one or more design decisions which extend the current design description. They tend to be goal directed and are usually invoked explicitly by the designer who then interacts with the support system to reduce the number of options until a single solution is found.

Control of the interactions between the support systems is in the style of a blackboard system [6]. Each support system is implemented as one or more knowledge sources. At each blackboard cycle, the system uses the knowledge sources to derive the consequences of the design description represented on the blackboard, reviewing any uncompleted work in the light of what is discovered and adjusting its inference priorities accordingly. The designer is viewed as a knowledge source whose 'bids' are always processed first. This allows the system to follow several lines of reasoning as it attempts to infer the consequences of the designer's decisions, while still giving priority to user input.

[1] The design and implementation of EDS are described in more detail in [14, 8, 7].

The system pays particular attention to any inconsistencies derived by the knowledge sources as these are often indicative of inconsistencies in the problem requirements or problems with the proposed solution. Inferences based on inconsistent premises are of no value. However it is important to continue to draw inferences from each of the decisions independently and to allow this the information on the blackboard is truth maintained using an assumption-based truth maintenance system (ATMS) [4]. In EDS, the assumptions represent the problem requirements, knowledge of the domain and the basic design decisions made by the designer. The justification structure records the dependencies between design decisions and requirements and their consequences.

To control the inferences the system performs an assumption-based context management system (ACMS) can be used to partition the information on the blackboard into a number of contexts or *views*, each of which contains information relating to a particular design solution or task. Views are defined either directly by specifying the assumptions they contain or indirectly in terms of other views. The contents of a view is everything which depends on the assumptions forming its definition. For example, a view containing a house might be defined in terms of the views containing the rooms of the house. The set of views forms a tree. Each view is assigned a name corresponding to a unique node in the tree. This allows relative naming of views, for example, the rooms of a house can have the same name in different house designs. The views system provides a level of organisation above that of the dependencies maintained by the ATMS which reflects the designer's interests and goals. Whereas justifications model the fine structure of the problem, views model the broad structure.

At any one time, exactly one view is selected as the current view. Only those items which are in the current view are visible to the knowledge sources and hence can form the basis of further inferences. Switching views can therefore be used as a simple but effective means of process control. By changing contexts, the designer can focus the system's attention on a particular part of the problem or on a particular kind of inference. As different assumptions become visible, so different knowledge sources are activated and different kinds of inference performed.

Design proceeds by creating instances of module classes and assigning values to their parameters to define one or more possible solutions.[2] When the designer makes an assumption one (or more) datum nodes are created in the current view to hold the new information. For example, in attempting to solve the house design problem, the designer might begin by creating an instance of the *house* module and assigning values to some of its parameters: $A \geq 100$, $C \leq 10,000$ etc. This information is examined by the knowledge sources to see if it, together with any information already assumed or derived, can be used to make further inferences. If a knowledge source is able to make an inference, it generates a bid in the form of a knowledge source activation record (KSAR) which is scored and merged into an agenda. At each blackboard cycle, the KSAR with the highest score is executed and the results are added to the design description. In the example above, the assumptions made by the user together with the constraint $A \times U = C$, can be used by a *valuePropagation* knowledge source to infer the additional constraint $U \leq 100$. This information may in turn form the basis for a new round of bids and this cycle continues until no executable KSARs remain in the agenda.

As new values for parameters or bounds on them are assumed or derived, consistency checks are performed between constraints and values by the *valueConflict* knowledge source. Conflicts result in the creation of a justification for the distinguished node $\langle false \rangle$ recording the fact that the assumptions involved are mutually inconsistent and cause the ATMS to partition the assumptions into mutually consistent sets. An inference which is only derivable from inconsistent assumptions loses support and cannot form the basis of further inferences. If there is no conflict, EDS marks this by justifying the datum $\langle consistent \rangle$ and proceeding as usual. For example, we may know that the lower limit on construction costs is £110/m^2. Adding the constraint $U > 110$ to the problem description leads to a justification

[2] This is an oversimplification—the designer can also define new parameters and constraints and assemble novel designs from existing modules.

for $\langle false \rangle$, recording the fact that the three assumptions $A \geq 100$, $C \leq 10,000$ and $U > 110$ are jointly inconsistent (and hence that the problem is insoluble) and invalidating any inferences based on these assumptions. However the constraint is consistent with each of the problem requirements individually and it can be used to derive new bounds on the values of A and C, i.e., that $A \leq 90.9$ and $C \geq 11,000$.

In determining how to overcome this problem the designer may make further assumptions to explore the implications of relaxing one or both of the original problem requirements, for example that $A = 90$ or that $C = 11,000$. These assumptions are inconsistent with the existing design solution and with each other. While these inconsistencies will be trapped by the ATMS, they are of no interest to the designer.[3] By placing assumptions relating to different designs in different views, the designer can avoid the derivation of such meaningless inconsistencies and partition the design description in response to their exploration of the problem.

The designer is therefore free to pursue alternative designs without worrying about inconsistencies except insofar as they indicate shortcomings in the proposed solution. Moreover the opportunistic nature of the blackboard control strategy means that whatever can be consistently derived from the design description will be derived.

4 Related Work

A number of other systems have concerned themselves with problem redefinition. Navinchandra's 'CYCLOPS' [11] employs a systematic search-based method for relaxing constraints until a solution can be found; the original problem is thus 'redefined' but only after an exhaustive initial search to solve the original problem has failed. Work at Rutgers views 'exploratory design' as hillclimbing in problem-specification-space directed by feedback from analyzing failures in candidate designs [17]. Such work makes important preliminary contributions to the area of solving ill-structured problems, but so far has only scratched the surface in that only a few predefined alternative routes for problem restructuring are explored, and there is considerable bias towards exhausting available search methods first, before redefinition operators are applied (contrasting with our view of problem redefinition as a basic activity which pervades exploration). Also, the single-context nature of existing problem redefinition work misses an essential use of this technique in exploration: that of comparing alternative problem formulations, and comparing different solutions from different problem formulations in order to further guide problem redefinition and solution development.

Myers *et al* [10] and Carter and MacCallum [3] describe systems systems in which a central blackboard is used to control the application of a number of support systems. These systems view design as essentially a single context activity where alternative designs exist only as discarded states. Few groups have considered the problems of multiple contexts (which is understandable since the problems involved in single-context design are formidable on their own). Exceptions include Patel and Dasgupta [12] who describe the use of a modified ATMS to perform belief revision among constraints in the context of their "Theory of Plausible Designs", and Banares-Alcantra [2] who describes the use of an ATMS for exploration in a support system for chemical plant design. Morgue and Chehire [9] advocate a blackboard-style system together with an ATMS for problem solving in complex domains and describe basic research into efficiency considerations for hybrid production-rule/ATMS systems. EDS differs from these systems mainly through our emphasis on pursuing alternative solution strategies in parallel and the need to derive information from incomplete and inconsistent design descriptions. It incorporates ideas found in both blackboard and ATMS-based design support systems together with extensions such as the ACMS to support design as a multi context exploration activity.

[3] Note that without the ATMS, these assumptions would result in the derivation of an infinite sequence of values for A and C.

5 Conclusion

EDS is implemented using the Poplog multi-language programming environment. It has been used extensively by several of the collaborators in the Alvey large scale demonstrator project 'Design to Product' (DtoP) which focussed on the design of light electro-mechanical components and formed part of the final DtoP demonstration system [1]. The architecture has also been applied in a number of other domains including drug design [15], nuclear power systems design [5] and option trading. Our initial experience with the architecture has been encouraging. The ATMS-blackboard and ACMS have been successful in supporting *ad hoc* explorations of the trade offs between design criteria in these domains. More importantly, our initial experience has served to highlight some of the shortcomings of the model of design support embodied in the architecture.

Further development of the architecture is limited by our understanding of the design process. A major problem with the current system is that it has no understanding of the design task and consequently which inferences it should perform in a given situation. In an attempt to overcome this problem we are currently investigating a number of extensions to the architecture described above, including the introduction of user-defined preference orderings over problem requirements, knowledge sources and views and the representation of knowledge sources and views as first class objects within the ATMS-blackboard. We believe such extensions can form a framework for a more effective control of inference. We are also investigating modifications to the implementation of special purpose support systems which will allow the system itself to explore the space of possible designs.

Acknowledgements

The development of the Edinburgh Designer System was partially funded by the UK Science and Engineering Research Council as part of the Alvey large scale demonstrator project Design to Product (DtoP), other parts of which were funded by GEC plc and Lucas Diesel Systems (a division of Lucas Automotive Ltd.) whose active involvement in our research through the DtoP project we gratefully acknowledge, together with that of the other DtoP collaborators at Leeds University Department of Mechanical Engineering and at Loughborough University Department of Computing Studies, Department of Engineering Production, and the Human Factors in Advanced Technology research centre. The application of the EDS architecture to drug design is funded by the joint SERC/DTI Information Technology programme under grant number GR/F/3567.8 in collaboration with Logica Cambridge, British Biotechnology Limited and CamAxys Limited. Work on EDS is currently supported under SERC grant No. GR/F/6200.1.

References

[1] 'Design to Product' An Alvey Programme Large-Scale Demonstrator Project: Final Report to the Department of Trade and Industry and the Science and Engineering Research Council on Contract Number LD/004, 1991. (Commercial in Confidence).

[2] R. Banares-Alcantara. Representing the engineering design process: two hypotheses. In J. Gero, editor, *Artificial Intelligence in Design 91*, pages 3–22. Butterworth-Heinemann, 1991.

[3] I. M. Carter and K. J. MacCallum. A software architecture for design coordination. In J. Gero, editor, *Artificial Intelligence in Design 91*, pages 859–881. Butterworth-Heinemann, 1991.

[4] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986.

[5] K. Furuta and T. Smithers. Numerical methods in AI-based design systems. In G. Rzevski and R. A. Adey, editors, *Applications of artificial intelligence in engineering VI, Proceedings of the 6th International Conference on Applications of Artificial Intelligence in Engineering*, pages 45–58, Southampton, 1991. Computational Mechanics Publications.

[6] B. Hayes-Roth. A blackboard architecture for control. *Journal of Artificial Intelligence*, 26:251–321, 1985.

[7] B. S. Logan, D. W. Corne, and T. Smithers. Enduring support: on defeasible reasoning in design support systems. In J. S. Gero, editor, *Artificial Intelligence in Design '92*, pages 433–454, Dordrecht, 1992. Kluwer Academic Publishers. (in press).

[8] B. S. Logan, K. Millington, and T. Smithers. Being economical with the truth: assumption-based context management in the Edinburgh Designer System. In J. Gero, editor, *Artificial Intelligence in Design 91*, pages 423–446. Butterworth-Heinemann, 1991.

[9] G. Morgue and T. Chehire. Efficiency of production systems when coupled with an assumption based truth maintenance system. In *AAAI-91: Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 268–274, Menlo Park, California, 1991. AAAI Press/MIT Press.

[10] L. Myers, J. Pohl, and A. Chapman. The ICADS expert design advisor: concepts and directions. In J. Gero, editor, *Artificial Intelligence in Design 91*, pages 897–920. Butterworth-Heinemann, 1991.

[11] D. Navinchandra. *Exploration and innovation in design: towards a computational model*. Springer-Verlag, New York, 1991.

[12] S. Patel and S. Dasgupta. Automated belief revision in plausibility-driven design processes. In J. Gero, editor, *Artificial Intelligence in Design 91*, pages 447–466. Butterworth-Heinemann, 1991.

[13] H. A. Simon. The structure of ill structured problems. *Artificial Intelligence*, 4:181–201, 1973.

[14] T. Smithers, A. Conkie, J. Doheny, B. Logan, K. Millington, and M. Xi Tang. Design as intelligent behaviour: An AI in design research programme. *Artificial Intelligence in Engineering*, 5(2):78–109, 1990.

[15] T. Smithers, M. X. Tang, N. Tomes, P. Buck, B. Clarke, G. Lloyd, K. Poulter, C. D. Floyd, and E. E. Hodgkin. Development of a knowledge based design support system. *Knowledge Based Systems*, 1992.

[16] T. Smithers and W. O. Troxell. Design is intelligent behaviour, but what's the formalism. *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, 2(4):89–98, 1990.

[17] C. Tong. Knowledge-based design as an engineering science: The Rutgers AI/Design project. Technical report, Laboratory for Computer Science Research, Busch Campus, Rutgers University, 1990.