Position Statement for the
WORKSHOP ON REUSABLE DESIGN SYSTEMS: UNDERSTANDING THE ROLE OF KNOWLEDGE
held in conjunction with the
SECOND INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE IN DESIGN
22-25 June 1992, Carnegie-Mellon University, Pittsburgh, USA.

THE REUSABILITY OF DSPL SYSTEMS

David C. Brown

AI Research Group
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609, USA.

The Design Specialists and Plans Language (DSPL) is a language in which to capture the knowledge of how to produce a design [Brown & Chandrasekaran 1989]. It is intended for routine design situations. It has been tried on mechanical, electrical and civil engineering tasks.

Routine design means that everything about the design process, including the knowledge needed, must be known in advance. Note that this does not mean that the specific design (i.e., the solution) is known in advance. Nor does it mean that the pattern of use of the knowledge (i.e., the design trace) is completely known in advance, but portions of it may be.

The language offers the ability to encode knowledge as different types of agents. Each type plays a specific role (function) in the final design process. As the knowledge represents routine problem-solving then we would expect it to be highly compiled [Brown 1991a].

A key question for this workshop is what are the actual and underlying ingredients of DSPL? This is difficult to determine as they are mostly compiled in. However, some of the agent types supplied by the language provide clues. Numbers in braces refer to the list below.

Specialists act as place-holders for Plans {8} and plan selection knowledge (Sponsors & Selectors). They each represent a subproblem, solving it by plan selection {11} and execution {5}. Plans are precompiled sequences of actions intended to provide the design for a subproblem. A Plan provides ordering {6} of design activity and a subproblem decomposition {3}. Plans also provide the basis for solution recomposition {9}. A Sponsor evaluates {4} the suitability of a plan for use in a particular situation {12}, while a Selector picks {11} a suitable Plan.

Steps represent the building blocks of the design process, providing a value for an attribute of the design. This is done using calculation or decision {1}, or by selection {11} using pattern matching {12}. Tasks group steps, and therefore help to define the problem decomposition {3}. Constraints test values {2} and make suggestions about patches {13}. Redesigners attempt to patch the design {7} in order to correct a constraint failure. Failure Handlers recognize failing situations {12} that might be patchable, or can trigger backtracking {10}.

This analysis suggests the following list of ingredients of DSPL:

1 Basic Synthesis,
2 Criticism,
3 Decomposition,
4 Evaluation,
5 Execution,
6 Ordering,
7 Patching,
8 Planning,
9 Recomposition,
10 Retraction,
11 Selection,
12 Situation Recognition,
13 Suggestion Making.

This is similar to the list presented in [Brown 1992]. It is not intended to be complete, and may not even be correct!

For example, it is clear that some of the list could be ingredients of others on the list. For example, Situation Recognition is an ingredient of several items on the list, such as Decomposition. In addition, in some cases "Planning = Decomposition + Ordering". Thus, Planning is a higher level ingredient than Decomposition, which in turn is at a higher level than Situation Recognition. This problem reappears in many published lists of types of intelligent activity or in lists of ingredients of intelligent activity.

Consequently, a big issue for those producing toolkits, or automatic configurers of problem-solving systems is to decide at what level of complexity the ingredients will be, and to decide if an attempt should be made to produce a set where the ingredients do not overlap.

The more complex an ingredient the more it prescribes how it can fit with other ingredients. Thus, low level ingredients provide more flexibility. Overlapping ingredients make it harder to decide which to use.

Klinker et al [1990] consider something reusable "if it can be employed for several domains and tasks". However, I think domains and tasks are separate issues.

The more compiled domain knowledge there is, the less likely the ingredient is to apply to many problems. With respect to tasks, it seems obvious that some of the ingredients listed above could be used in tasks that have nothing to do with design.

What about reuse of knowledge which is encoded in a task dependent language such as DSPL? It certainly isn't all reusable for other tasks. However, the situation recognition pieces might be. The only way to reuse portions of a DSPL-based design system is to break it apart into individual agents or collections of agents, and then index them appropriately — whatever "appropriately" might mean. The problem will be that much of it will be extremely context-dependent, and will require exactly the same situation in order to produce correct results.

Some of our current work is attempting to reuse design knowledge. It is investigating how evidence from knowledge such as function, structure, and design cases, can be integrated so that an IntCAD system can learn to automatically decompose design problems [Liu & Brown 1992].

In this work, the knowledge is not used as an "agent" but rather inspected and appropriate pieces extracted for use. This requires indexing of problem decomposition knowledge from past design cases. The context is recorded, and is checked before extraction takes place. In this way the design knowledge is used only in its appropriate context.

D. C. Brown (1991) "Compilation: The Hidden Dimension of Design Systems". *Intelligent CAD*, III, (Eds.) H. Yoshikawa & F. Arbab, North-Holland.

D. C. Brown (1991a) "Routineness Revisited". *Mechanical Design: Theory and Methodology*, (Eds.) M. Waldron & K. Waldron, Springer-Verlag, to appear.

D. C. Brown (1992) "Design". *Encyclopedia of Artificial Intelligence*, 2nd edition, (Ed.) S. Shapiro, J. Wiley & Sons.

D. C. Brown & B. Chandrasekaran (May 1989) *Design Problem Solving: Knowledge Structures and Control Strategies.* Research Notes in Artificial Intelligence Series, Pitman Publishing, Ltd., London, England.

G. Klinker, C. Bhola, G. Dallemagne, D. Marques & J. McDermott (1990) "Usable and Reusable Programming Constructs", *Proc. 5th Kn. Acq. Wkshp*., AAAI.

J. Liu & D. C. Brown (July 1992) "The Generation of Decomposition Knowledge for Near Routine Design Problems". *Proc. International Conf. on the Applications of AI in Engineering*, Waterloo, Canada, Computational Mechanics Publications.