Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1996), *10*, 157–159. Printed in the USA.

RESEARCH ABSTRACT

S-SALT: A problem-solver, knowledge acquisition tool and associated knowledge base refinement mechanisms

PIETRO LEO, DEREK SLEEMAN, AND AUGOUSTOS TSINAKOS
Department of Computing Science, University of Aberdeen, Scotland, U.K., AB9 2UB

(RECEIVED October 2, 1995; ACCEPTED October 25, 1995)

1. INTRODUCTION

SALT is the highly influential system implemented by Marcus (1988), which is able to acquire knowledge and solve problems using the propose-and-revise strategy. The primary task to which this has been applied is the design of elevators, or lifts (Marcus et al., 1988). Recently we have re-implemented and extended the system at Aberdeen, S-SALT, the new system, has the following enhancements:

- The knowledge checking phase has been made a distinct module.
- A knowledge refinement module has been added that detects some kinds of incompleteness and inconsistencies, given a set of previously solved tasks and their solutions. Additionally, a refinement mechanism is activated when all the current "fixes" fail to solve a *particular* constraint violation;
- Facilities have been added to the Problem-solving module to help the user guide the search (this is particularly helpful when the space of possible configurations is large).
- The system user has an option of using either a graphical or a textual interface.
- The system has been reimplemented in Common LISP, thus making it assessable to a larger community.

From the perspective of Machine Learning, the Knowledge Refinement module is the most significant enhancement.

Reprint requests to: Derek Sleeman, Department of Computing Science, University of Aberdeen, Scotland, U.K., AB9 2UB. E-mail: sleeman@csd.abdn.ac.uk

2. DESIGN OF THE S-SALT SYSTEM

We have made the system very modular, so it is relatively easy to change/redesign a module as long as the common interface between modules is maintained. Essentially, we have implemented the system as five components:

1. The **Control** module, which is initially activated by the user and provides a link between the several modules.
2. The **Problem-solving** module is given a task and three kinds of knowledge, namely:

   - Extension procedures to either create or enhance a design:
   - Constraints that note when designated variables fall outside expert-defined ranges:
   - Fixes that suggest changes to be made (to one or more variables) when the value of a particular variable is not within limits.

   This module attempts to construct a solution to the design task. If the values of several variables are violated, then S-SALT asks the user which variable he would like to fix first and then the system applies all the expert-defined fixes for his variable, in increasing order of cost.
3. The **Knowledge Acquisition** module asks the expert for the three types of knowledge (extension procedures, constraints, and fixes), which the problem-solver uses. Essentially, the system has a template for *each* kind of knowledge to be acquired (in this respect, S-SALT is similar to SALT).

4. The **Knowledge Checker** module is able to detect:
   - Variables that have not been defined (i.e., have *no* expression associated with them, etc.).
   - Cyclic dependencies between variables (which require the expert to provide a further way of specifying one of the variables).

5. The **Knowledge Refinement** module detects semantic errors, that is, errors due to incorrect/inappropriate definition of certain pieces of knowledge. Here, the Knowledge Base (KB) is trained on a set of positive examples, which could be either complete or partial (Craw, 1990). The system reports the differences between the answer supplied and the calculated value for a parameter; further, the system suggests which constraints or procedures might be faulty.

3. IMPLEMENTATION OF S-SALT

The system has been implemented totally in LUCID COMMON LISP 4.0 and Lispview 1.1 (note that the original SALT system was implemented in OPS5). Note too that this system runs in two modes: a textual mode, which operates on a standard terminal and a graphical window version for Sun Sparcstation, which runs under operating system SunOS 4.X and Open Window version 2.0 (or above).

The system has, in fact, solved the VT task and the full Sisyphus-II VT task; for details of the system's performance on these tasks and details of the implementation, see Leo et al. (1994) and Leo (forthcoming).

4. FURTHER WORK

We are well aware that much has to be done before this system could be used on a demanding task by either an Expert Designer or a Student learning design. We are considering many possible extensions including:

1. Moving to a further domain, in an attempt to determine whether a designer, unaided by a knowledge engineer, could use the system to design (redesign) a new artefact.
2. What environment would S-SALT need to be embedded within to satisfy a practicing designer?

   Engineers frequently use sketches when they are in the earlier stages of design, and more detailed drawings at later stages. CAD and even Intelligent CAD systems are now frequently used by professional designers. Thus, to have a *significant* impact it would seem likely that S-SALT would have either a graphical capability or be interfaced to a CAD package. This would be a demanding system task.
3. An aide to teach Design to Students: Students typically find it hard to *begin* to design an artefact[1]; as unlike the expert designer, they may *not* have designed something analogous before. For a profes-

[1] Students beginning to program also frequently reported "problems with getting started."

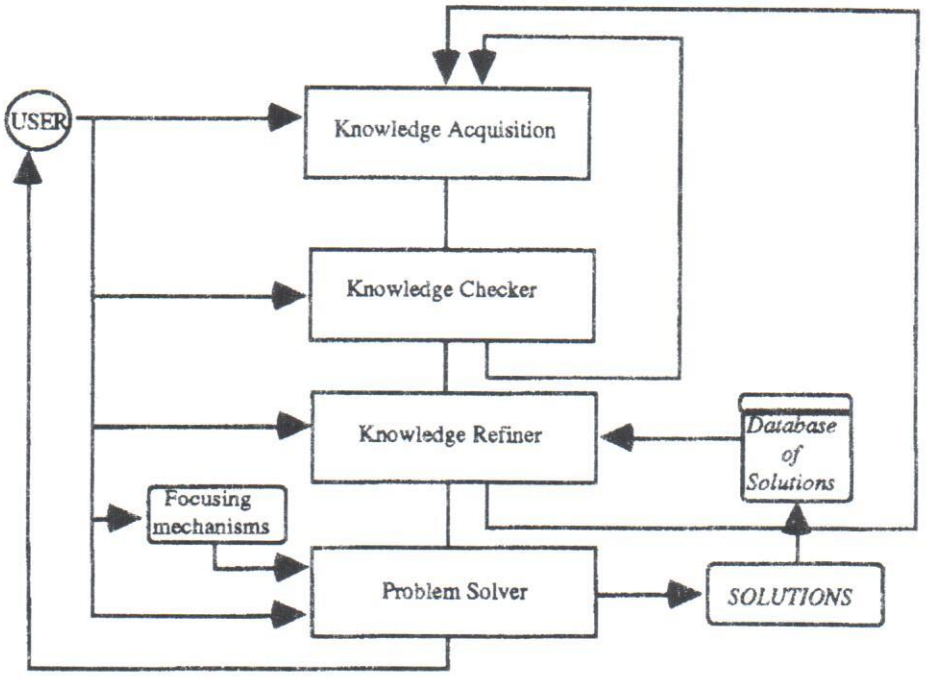

Fig. 1. The S-SALT structure.

sional designer, most new jobs do *not* involve design but really *redesign*. A student, on the other hand, may not even have a *possible* strategy. Although, we are sure there are many approaches, the one developed by SALT and S-SALT of articulating three distinct data structures is a possible "starting point," (namely, enhancement rules, constraints, and fixes). However, even given that perspective, some students might find it hard to start a design. In which case we would advocate they simply list, in no particular order, all the parameters that they think will be needed to design the artefact, like a lift, and then label these in various ways:

- which can be provided as *input* by the designer;
- between which sets of variables there are relationships, for example, TOTAL-MAX-SUSPENDED-LOAD = CAR-WEIGHT + PASSENGERS-WEIGHT;
- between which variables there are constraints;

As you can see, once the above have been articulated, the student would be well under way to formulating the three knowledge structures that SALT and S-SALT require.

ACKNOWLEDGMENT

The authors thank Pat Fothergill and Keith Dodd for helpful discussions on the system as it evolved. Pietro Leo is supported by a studentship from the Bari Polytechnic, and Augoustos Tsinakos was supported by a European Community Social Fund award. The authors thank Davide Roverso for helping with the final stages of preparation of this manuscript.

REFERENCES

Craw, S. & Sleeman, D. (1990). Automating the refinement of knowledge-based systems. *Proc. ECAI 90*, 167–172.

Leo, P., Sleeman, D., & Tsinakos, A. (1994). S-SALT: A problem solver plus knowledge acquisition tool which can additionally refine its knowledge base. *EKAW-94 Conf. Proc.*

Leo, P. (199x). S-SALT: A problem solver plus knowledge acquisition tool which additionally can refine its knowledge base. M.Sc. Thesis, University of Aberdeen, UK.

Marcus, S. (1988). A knowledge acquisition tool for propose-and-revise systems. In *Automating Knowledge Acquisition for Expert Systems*, (Marcus, S., Eds.), pp. 81–123. Kluwer Academic, Boston.

Marcus, S., Stout, J., & McDermott, J. (1988). VT: An expert elevator configurer that uses knowledge-based backtracking. *AI Magazine 9(1)*, 95–112.