Intelligent Computer-Aided Design in the XRL Object-Oriented Knowledge Processing Environment

Stefan Trausan-Matu and Mihai Barbuceanu

Expert Systems Laboratory
Research Institute for Informatics
8-10 Averescu Avenue,
71316 Bucharest
ROMANIA

Abstract: This paper presents some applications written in the XRL object-oriented knowledge processing environment. All of them are expert systems for design. Several considerations concerning such systems are introduced. The architecture, knowledge representation and processing in the XRL design applications are discussed.

Stefan Trausan-Matu received the engineer degree in computers from the Polytechnical University of Bucharest in 1983. Between 1983 and 1985 he worked in the CAD of integrated circuits at "Microelectronica" Company in Bucharest. Since 1985 he has worked in the domain of Knowledge Representation and Processing at the Research Institute for Informatics in Bucharest. Now he is the head of Expert Systems Laboratory in this institute. He has taught several courses on Artificial Intelligence, Data Structures and Algorithms, and Advanced Programming Languages at the "Politehnica" University of Bucharest.
His reasearch interests are knowledge representation, constraint processing, object-oriented systems, expert systems, and artificial intelligence applications in software engineering (which is also the subject of a Ph.D. thesis which will be completed this year).

Mihai Barbuceanu got the Ph.D. degree from the "Politehnica" University of Bucharest in 1993. After graduating in 1979, he joined the Research Institute for Informatics as a research fellow, a senior researcher, and, finally, as head of laboratory. For several years he has worked together with Stefan Trausan-Matu on the development of the XRL environment. The original research results have been presented at various scientific events and published in international journals.
His reasearch interests include knowledge representation, object- oriented systems, expert systems, knowledge acquisition, term subsumption languages.

1. Introduction

Expert systems are programs developed in the idea of having a similar behaviour to that of human experts. What does it mean a "Similar behaviour"? It means that they can store an amount of knowledge, use their knowledge to infer some facts which are directed towards the attainment of a goal, explain their reasoning, and, eventually, acquire new knowledge or restructure the existing knowledge. An expert system can be considered as a thesaurus of human knowledge in its domain of competence. From a software engineering perspective, the rapid prototyping and evolutionary life cycle of expert systems [3] enables the iterative transfer of knowledge from the expert to the system.

The expert system approach can extend the Computer-Aided Design (CAD) paradigm by automating some complex parts of the design process. Design can be considered a knowledge intensive domain and, therefore, expert systems can be a real help to designers by extending the traditional CAD systems with an intelligent dimension. In this way, expert systems for design might be considered as intelligent assistants that enable the human designer to concentrate on the creative aspects of design.

XRL is an object-oriented programming environment written in LISP, developed along several years at ICI for knowledge processing applications [1,2,4]. It has a layered architecture based on an object-oriented substrate and including layers for rule-based knowledge representation and processing, concurrent refinement of structured objects, constraint processing, logic programming. Several applications in XRL were written, mainly expert systems for design. In this paper there are described two representatives of classes of expert systems developed in XRL.

2. Expert Systems for Configuration Design

Two expert systems for configuration design were the first applications of XRL. The first one configurated a programmable logic controller and the second the microPDP-11 computer. Both of them used only the object-oriented substrate and the concurrent refinement system. Both systems have a similar architecture. Each of them employed a blackboard with two spaces, one for specification acquisition and the other for building the configuration. Thus, these spaces correspond to the specification and configuration subtasks of the design task. For both systems specification and configuration were carried out by concurrent refinement of the generic objects describing specifications and configurations.

For the specification subtask the system had a knowledge base consisting of a taxonomy of objects representing available modules and top level specification objects which were generic descriptions of whole specifications. The refinement of a specification object consisted in an interactive selection of the modules to be included in the configuration. During refinement, some constraints were taken into account. For example, for the microPDP-11 computer, there may be at most two disk units and one CPU. As the constraint language was not available at the time, constraints were enforced by the structure of the specification acquisition object and by the attached methods performing refinement.

The second subtask was the configuration design. This was accomplished by refining in a special space a collection of objects starting with a top-level configuration design object. The refinement of this object means filling its slots with modules from the specification acquisition object net built in the first phase. This operation is driven by a set of constraints regarding the current and power consumption, the placement of the modules in some required order and the verification that the placed modules correspond to the characteristics of the backplane slots where they have been inserted.

To illustrate this style of building systems with complex frames and refinement processes, we show below a sketch of the PDP11 configuration system. This is a simplified version which uses only one space on the blackboard doing both specification and configuration in the same instance network. First, the top level frame to be refined is the following:

(unit MPDP11Configuration

self (an ExpandUnit)

module-groups (cpu fpp very-high-p high-p medium-p low-p tbd-p)

standard-confs (an AskableSelection

alternatives (mpdp11-11a32-r-f mpdp11-11c23-r-f mpdp11-sxra500-f-a))

cpu (an AskableModule type cpu from mpdp11Specs)

fpp (an AskableModule typr fpp from mpdp11Specs)

very-high-p (an AskableGroup type vh from mpdp11Specs)

high-p (an AskableGroup type h from mpdp11Specs)

medium-p (an AskableGroup type m from mpdp11Specs)

low-p (an AskableGroup type l from mpdp11Specs) tbd-p (an AskableGroup type tbd from mpdp11Specs)

selected-modules (selected-of cpu fpp very-high-p high-p medium-p low-p tbd-p)

backplane (a mpdp11Backplane modules (my selected-modules))

resource-usage (a resourceUsage modules (my selected-modules))

(a SlotMeta when-out-done ShowResource Table))

This unit organizes the task of configuration design by dividing it into several information acquisition tasks (the Askables-s), one backplane design task and one resource consumption calculation task. The information acquisition task consists in refining several slots: standard-confs, cpu, fpp, etc. Their refinement is driven by units (AskableSelection, AskableGroup, Askable Module) with method protocols for accessing a database of modules indexed according to their type and for carrying out an interactive dialogue for selecting from among them. For example, the

AskableGroup unit specifies the selection of modules from a group of modules with the same bus priority:

```
(unit AskableGroup
 self (an ExpandUnit)
 group (msg from 'get-type type)
 selected (ask-menu group)).
```

Here, the group slot is filled by sending the "get-type" message to the object in the "from" slot (assumed to hold a database of modules) which is supposed to return all modules of the given type. Then, with the ask-menu function some will be selected.

When selections are over, in the selected-modules slot the list of all selected modules is built. This list is then used to configure the backplane and to compute resource consumption. The task of backplane configuration is achieved by the refinement of the following unit:

```
(unit MPDP11-Backplane
self (an ExpandUnit)
slots (slot1 slot2 slot3 slot4 slot5 slot6 slot7 slot8)
slot1 (a cpu-slot)
slot2 (an one-module-slot)
slot3 (an one-module-slot)
slot4 (an one-or-two-module-slot)
slot5 (an one-or-two-module-slot)
slot6 (an one-or-two-module-slot)
slot7 (an one-or-two-module-slot)
slot8 (an one-or-two-module-slot))
```

According to the semantics of concurrent refinement, to refine the backplane unit refinement tasks are created for each of its slots. Then the unit in each slot is delegated to refine itself. In the above case each of these units refines itself by placing modules from the selected-modules in its slot. The first slot is restricted to the cpu, the next two accept only one module and the rest accepts one or two modules depending on their dimension. Each of these units has its own refinement protocol that defines its refinement.

This example shows how the hierarchical refinement model, consisting of collections of actors which collaborate to build a structured object network, can be used to build knowledge based expert systems. This model has been applied in several other tasks, the most complex of which being discussed next.

3. DEXTY - An Expert System for the Design of Industrial Halls

Several expert systems were developed for the field of civil engineering in collaboration with a company specialized in civil engineering design. These systems made it possible to integrate the classic CAD approach based on graphics and numeric computation with knowledge based problem solving as supported by XRL. In this presentation we mention only the DEXTY system, probably the most complex expert system delivered to industry in Romania to date.

DEXTY [5] is a fully automatic system for designing industrial halls. It provides the following major functions:

a. Acquisition of the specification of the hall to be designed. In this phase, the user is asked to furnish some basic characteristics of the hall, for example the geometry of the hall (the number of bays and openings), the characteristics of the geographic region (earthquake degree, snowing characteristics), the destination of the hall, etc.

b. Automatic design of the structure of the hall. Here the system builds a network of generic objects for the geometric elements of the hall. For example, objects will be generated for each row of bays. Each row of bays will contain the generic objects which constitute its typical components (chessons, beams, pylons, foundations).

c. Architectural design for the placement of sky-light on the roof. This phase implies interactive graphics and the use (and possibly extension) of a collection of standard configurations.

d. Refinement of the generic elements from the above generated structure according to a collection of standard elements retrieved from design catalogues. The refinement of each element (chessons, beams, etc) is followed by stress verification using specialized programs developed

in FORTRAN. In case of a negative answer to the verification, another element is chosen.

DEXTY uses most of the knowledge representation techniques available in XRL. Structured objects are used both in the style of frames (as taxonomies of typical situations or expectations) and in the style of objects (as reusable entities with state, inheritance, message passing protocols, etc.). For example, the expectations about the characteristics of the hall are described as objects used for specification acquisition. The generic and instantiated components, like chessons and beams, are structured objects in the OOP sense. Problem-solving mechanisms, such as generic decision-making processes about let's say choosing a typified element, are also encoded as structured objects.

Rule-based representations are also used in DEXTY. Rules are mainly used for capturing expertise regarding design decisions in the following classes of problems:

a. Validation of user supplied data in the first phase. Invalid data configurations are detected by specialized rule systems.

b. Major design decisions such as deciding the space orientation of the chessons in the hall. The cross or the longitudinal orientation is chosen depending on the geometric input data, the available standardized elements and the expertise embodied in rules.

c. Refinement of generic beams into particular ones. The refinement is done by selecting an appropriate beam for the characteristics of the current problem. The rule systems embody the knowledge usually employed by the human in solving a similar design decision problem.

The problem-solving structure of the system is more complex. DEXTY is composed of several smaller cooperating expert systems, each implemented as a separate refinement system having its own blackboard spaces and being allowed to access other's spaces as well. The first refinement system, which must always be invoked first, deals with requirements acquisition, consistency checking of requirements and preliminary geometrical design for the whole building. Then, according to the user's desire, one or several refinement systems designing various parts of the building can be invoked. These refinement systems perform (1) roof design (2) selection of the columns and beams supporting the building (3) walls design, and (4) foundation design. They must be invoked in this order. Design undoing and redoing is also supported based on the replay and event handling facilities of the tool.

This organization is interesting for several reasons. First, refinement systems have proven to be a useful vehicle for modularizing systems and for modelling the global problem-solving strategy. Second, the system integrated both constructive and selective problem-solving. The latter activity led to the evolution of specialized annotation protocols, i.e. the usual way of extending and customizing concurrent refinement. Third, DEXTY was extensively modified during development. If the first prototype was built in about two months, during the next two months it underwent about ten substantial revisions. Our belief is that it was so easy to modify due to the declarative and understandable programming style figured up with annotated structured concepts and refinement architectures. This is also confirmed by the experience with modifying the earlier systems years after their development.

4. Conclusions

Our experience with building expert systems for design enables us to conclude that they can both reduce the design process time and enhance the quality of the results. Expert systems for design are also means thereby the expertise of skilled designers can be used by novices. On the other way, as a feedback result in building expert systems, skilled designers can better realize what they do know and which are the directions where they must do some research.

Other important conclusion from our work is that, even on ordinary computers, there can be developed complex knowledge processing systems. Nevertheless, the usage of a powerful structured-object, multi-paradigm environment like XRL was decisive in coping with the permanent change of the specifications. We consider that "classical" languages (e.g. C, FORTRAN) or even

other knowledge representation paradigms alone (rules or logic programming) could not handle the problems encountered.

REFERENCES

1. BARBUCEANU, M., TRAUSAN-MATU, S. and MOLNAR, B., **Integrating Declarative Knowledge Programming Styles and Tools in a Structured Object AI Environment**, Proceedings Tenth International Joint Conference on Artificial Intelligence, Milan, Italy, 1987, pp. 563-568.
2. BARBUCEANU, M., TRAUSAN-MATU, S. and MOLNAR, B., **The XRL2 Manual**, Research Institute for Computer Technique and Informatics, Bucharest, Romania, 1988.
3. DOYLE, J., **Expert Systems and the "Myth" of Symbolic Reasoning**, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, SE-11, No. 11, November 1985.
4. TRAUSAN-MATU, S., **MicroXRL- An Object-Oriented Programming Language for Microcomputers**, Institute for Technical Cybernetics of Slovak Academy of Sciences, Bratislava, Slovakia, 1989.
5. TRAUSAN-MATU, S. and BARBUCEANU, M., **Generic Knowledge Processing Architectures for CAD in Civil Engineering**, Proceedings of INFOTEC'88, Bucharest, Romania, September 1988 (in Romanian).