Toward a General Methodology for Specifying Expert Systems

Jonathan Lee

Department of Computer Science
Texas A&M University
College Station, TX 77843
Yjlee@CS.TAMU.EDU

John Yen

Department of Computer Science
Texas A&M University
College Station, TX 77843
Yen@CS.TAMU.EDU

Abstract

There has been an increasing interest in improving the quality and reliability of AI systems. As a result, several methodologies for specifying expert systems have been proposed in recent years. However, these methodologies have been demonstrated either for a particular problem solving architecture or for a specific type of expert systems. It is unclear how one can apply these methodologies to other types of expert systems or other problem solving architectures.

To alleviate the above problems, we propose a general methodology for specifying both the model and the process knowledge of an expert system at different abstraction levels. Specification are acquired and organized around the general notion of task. The model specification of a task describes a partial model of the application domain and a partial model of the problem solving states relevant to the task. The process specification of a task describes states before, during, and after the task as well as task state expression to describe the behavior. A piece of abstract specification can be refined to a more detailed specification. Specification at different abstraction levels can thus be verified for their consistency and completeness. The potential benefits of our methodology are illustrated using a well known computer configuration expert system - R1.

1 Introduction

Of particular importance for any system is to produce an adequate specification. A specification for a software system can help in organizing the expertise, representing the knowledge, validating against the design and implementation, and maintaining the system, thus eliminating possible design failures and reducing product costs[1].

The research on specification for conventional software systems has been extensively used for practical computer systems and applications, such as filing system[5], and real-time kernel [7]. The application of these specification techniques to expert systems, however, are very limited because (1) the problems are well-defined for conventional software but ill-defined for expert systems, and (2) the solutions for conventional software are algorithmic, but the solutions for expert systems are non-algorithmic.

There has been an increasing interest in improving the quality and reliability of AI systems. As a result, several research work on expert systems specifications have been proposed in recent years[8, 3, 10]. However, these methodologies have been demonstrated either for a particular problem solving architecture (e.g., the SOAR architecture) or for a specific type of expert systems (e.g., heuristic classification systems). It is unclear how one can apply these methodologies to other types of expert systems or other problem solving architectures.

As a continuation of our previous work in improving the reliability of expert systems[9], we aim to develop an approach to knowledge specification that (1) is independent of problem solving architectures, (2) can be applied to various types of expert system applications (e.g., classification, synthesis, monitoring and control), (3) concerns only with what the knowledge is, not with how the knowledge is used, (4) introduces the major concepts involved gradually, and (5) supports verification and validation. Toward this objective, we have outlined a task-based knowledge specification methodology[4]: specifying both the model and the process knowledge of an expert system at different abstraction levels. Specification are acquired, and organized around the general notion of task. The model specification of a task describes a partial model of the application domain and a partial model of the problem solving states relevant to the task. A piece of abstract specification can be refined to a more detailed specification. Specification at different abstraction levels can thus be verified for their consistency and completeness.

In this paper, we further extend our methodology to use the notion of state transition to explicitly describe what the functionality of a task is and use task state expression to describe the behavior of a task.

Previous work on expert systems specifications are first described in the next section as a background of our approach to knowledge specification. An overview of our task-based methodology is then presented in section 3. Finally, we summarize the benefits of our approach and outline our future research plan.

2 Related Work: Expert Systems Specification

Zualkernan and Tsai developed a knowledge-model specification methodology for expert systems. The
methodology describes what knowledge structures and the
goal structures that are required to solve the prob-
lem [8]. This methodology provides a top-down ap-
proach and has been demonstrated for heuristic clas-
sification systems. Verification and validation are per-
formed through the interaction with experts. How-
ever, the methodology does not offer any automatic
verification checking capabilities.

James Stagle and his students have demonstrated
a conceptual structures methodology for specifying
a heuristic classification expert system [3]. This
methodology begins with type definitions, identifies
the high-level lines of reasoning focused on a few con-
cepts, construct implications, facts, type lattice and
canonical graphs. The methodology is suitable for
heuristic classification type systems, whose knowledge
can be specified by constructing an inference network
that relates pieces of evidence to hypotheses. Verifi-
cation of the specification is limited to checking canoni-
cal graphs.

A problem space approach to expert system spec-
ification has been proposed by Gregg Host and Allen
Newell [10]. The approach is designed specifically for
the SOAR architecture. It is difficult to see how the
methodology can be applied to other AI architectures.
No verification of the specification is provided by the
methodology.

3 A General Methodology for Specifying
Expert Systems

3.1 Components of the Specification

An expert system specification in our methodol-
ogy has two major components: a model specifi-
cation and a process specification. The former charac-
terizes static properties of the system, while the later
describes dynamic properties of the system. Both the
model and the process specifications are captured and
organized around the notion of task.

A model of a task has two components: a model
about the domain objects, and a model about the
problem solving state objects. The model about the
domain objects describe terms, relations, and con-
straints that are relevant to the task. A term is a
user-defined type that could represent abstract con-
cepts (e.g. configured-backplane) or concrete concepts
(e.g. cable length). A relation describes a property
of a term (e.g. has-component) or a relationship be-
tween terms (e.g. has-connection). Constraints can
be classified into two categories: weak constraints and
strong constraints. A weak constraint can be a pref-
erence constraint, which provides advices for finding
a better solution. Weak constraints are not rigid con-
straints, therefore we do not need to specify a repair
method for fixing a violation of these constraints. A
strong constraint, which is a condition that the sys-
tem has to satisfy (called a must constraint) or has to
avoid (called a cannot constraint), provides an order
to be obeyed. For example, the following is a strong
constraint in RI.

If the previous backplane is in the same box
then use a cable of length 10.

A strong constraint can be associated with one or
several fixes that describe various ways to repair a vi-
olation of the constraint.

The second component of the model describes ob-
jects (which we call state objects) that are used to
characterize problem solving states of an expert sys-
tem (e.g. configured-backplane). These objects are
often defined in terms of domain objects defined in
the first part of the model. For example, the descrip-
tion of the state object configured-backplane has to
refer to the concept of backplanes, modules, and slots.

The process specification of a task is two-fold: (1)
functional specification to describe 'what a task is' us-
ging state transition such as preconditions, protections,
and postconditions, which will be elaborated in the
next section, and (2) behavioral specification to de-
scribe the relationship among tasks and interactions
between tasks by means of task state expression.

3.2 Functional Specification as State-
Transition

In conventional software specifications (e.g. [9]), op-
erations/events are often viewed as means to change
states and to characterize preconditions and postcon-
ditions. However, in most of the expert systems spec-
fications methodologies, the notion of characterizing
operations as state changes is missing. This is be-
cause the operations in expert system applications are
much more complicated than those in conventional
softwares. Thus a more expressive notation for char-
terizing states is needed. In our methodology, states
are captured in the model specification, as discussed in
the previous section.

In order to explicitly specify what a task is in our
methodology, we treat a task as a state transition and
specify the states before, during, and after the oper-
ating using precondition, invariant and postcondition.
The precondition of a task describes the situation un-
der which the task can be invoked. The postcondition
of a task describes desirable state changes that should
be achieved by the task. The invariant of a task de-
scribes state properties that remain unchanged during
the period in which the task is performed. Precon-
ditions, invariants, and postconditions are described us-
ing state objects in the state model specification. The
following example describes precondition and postcon-
dition of the task: configure a backplane (Figure 1).

Planning in AI has used protection to limit the in-
teraction between subgoals. Adopting this idea to the
specification of tasks not only insures that undesir-
able interactions between subtasks are avoided, but
also enables the methodology to verify that functional
specifications of tasks, methods, and subtasks are con-
sistent.

3.3 Behavioral Specification as Task State
Expression

The technique of treating tasks as state-transition
allow a knowledge engineer to structure the function-
ality of a system into the specification. The knowledge
engineer must still specify the expected interactions of
these functional units (i.e. tasks) before anyone can
verify that the behavior of the knowledge base meets
the design intention.
Task: configure a backplane

- Model Specification:
  - State Model:
    * A configured backplane is the backplane that is moduled backplane, boxed backplane, and cabled backplane.
  - Process Specification:
    - TSE: +[unassigned backplane]
    - Precondition:
      1. There exists a backplane module for current module.
      2. There exists unconfigured boxes and modules.
    - Protection: Boards from different modules should not be interleaved.
    - Postcondition:
      * The chosen backplane is a configured backplane.

Figure 1: An Example of Task Specification

One method we are investigating for managing the relationships between tasks is to define and verify a task state expression (TSE) of each task. The notion of task state expression is an extension of operator and data state expressions used in conventional software verification. A task state expression, in our methodology, is an expression that defines (1) the desirable sequencing/intention of tasks that are expected to have processed before the given task, (2) an undesirable sequencing/intention of tasks that are not expected to proceed the given task, and (3) the interaction of tasks at different layers. The syntax for TSE is as follows: + stands for the task itself, ! for iteration, and square bracket for optional.

This example (Figure 1) illustrates usage of TSE expressions in documenting the following expectations. If the task configure a backplane fails to configure modules into the current backplane, then we should backtrack to unassigned backplane. The following are important points concerning TSE's:

- The intent of TSE's is not to define control but instead to document expectations about control.
- A TSE defines an explicit task grouping (sequence module) and an explicit temporal relationship on the task grouping that can be verified.

3.4 Model and Process Refinement

In order to help the knowledge engineer to focus on constructing a part of the specification at an appropriate abstraction level, and to avoid overloading him/her with too much detailed information, the task-based methodology supports the refinement of both the model and the process specification. Both the model and the process specifications can be first described in their high-level abstract forms, which can be further refined into more detail specifications in the next level. During the refinement process, some of the specifications may be an elaboration of model, constraint or process in the previous layer, others may be completely new constructs not mentioned in the previous layer. Some of the details could be treated as common sense knowledge, while others as exceptions.

In model refinement, both abstract and concrete concepts are refined together with their relationships. For example, the term 'connectivity' at the top level can be refined at the next level to 'configuring into', which is further refined at an even lower level to 'putting modules into slots on the backplane'.

There are two phases in the process refinement: (1) Refine a task into a set of problem solving methods that can accomplish the task. For example, the task 'select backplane' can be refined to one of the several problem solving strategies: propose-revise, least commitment, or search. (2) Refine a problem solving method by specifying subtasks involved and temporal relationship between subtasks using TSE.

To illustrate these ideas, let us consider the task of configuring box in R1. The propose and revise method for the task 'configure a box' can be refined to describe ways to revise a solution if the modules in the backplane fail to configure. The production of an adequate specification is of particular importance. Errors committed in specifications create faults which are very difficult to detect and costly to remove. Verification is, therefore, given special attention in our methodology.

Verification in our methodology is divided into two parts: (1) verification of model specification regarding constraints, and (2) verification of process specification regarding preconditions, invariants and postconditions. Furthermore, verification can be performed for a single specification in a single layer or among multiple layers. Verification in a single layer performs
self-consistency checking using domain constraint and sequence constraints. Verification between layers is two-fold: consistency and redundancy checking. A constraint defines a set of possible solutions that satisfy the constraint. Redundancy checking is to check for the subsumption relations among constraints in between layers. Consistency checking is to check for the conflict among related constraints in between layers. Two constraints are related, if they have common predicates in their situations and in their bodies. However, not all potential inconsistent constraints need to be corrected. More specifically, a constraint may describe exceptions of a more general constraint. Therefore, potential inconsistencies due to 'exceptions' will be distinguished from others in our methodology.

Potential missing/incomplete knowledge is pointed out at each layer during the refinement process. A more thorough checking for incompleteness can be performed when the knowledge engineer has completed refining the specifications. Examples of incompleteness testing include checking for 'unrefined' complex task, potentially unresolved preferences, and lack of 'fixes' for the violation of certain constraints.

### 3.6 Construction Procedure

The task-based methodology specifies an expert system by first constructing a partial task hierarchy which is followed by model and process refinement. The procedure for constructing the specification is outlined below:

1. Construct a partial task hierarchy for an overall view of the solution. This information is either derived from knowledge acquired from expert or from documentation/references.
2. Pick up a task and construct abstract specification of the model, constraints, states, and state changes relevant to the task. As the knowledge engineer moves to lower level tasks, more detailed specification about the model and process will be constructed by refining their abstract counterparts in the following ways:
   - identify terms and relations between terms to form a term hierarchy in the model;
   - identify different types of constraints in the system so that system verification can be performed based on the types of constraints;
   - identify preconditions, postconditions, and invariants of methods and subtasks;
   - identify the sequence and the interaction of subtasks;
3. Verify for consistency, redundancy, and completeness in a single layer and between layers of the generated specification;
4. Step 2 and 3 are repeated until all the subtasks identified have been specified.

At the end, an overall specification of an expert system is formed for both the model knowledge and the process knowledge. Furthermore, missing and incomplete knowledge is identified.

### 4 Conclusion

In this paper, we have outlined a task-based methodology for knowledge specification in which the specification is driven by the task structure of problem solving knowledge, pieces of specification can be refined/decomposed iteratively and verification is performed for a single layer or between layers. The methodology is illustrated using the problem domain of R1.

Our future work will consist of several tasks. First, we will further construct a complete specification language using conceptual graphs. Second, we plan to apply the specification methodology to the problem domain of R1/SOAR to illustrate the feasibility and the benefits of our approach. Finally, we intend to address the issue of testing in the context of specification.

### References


