Chapter 1

Introduction

The main goal of this thesis is the design and implementation of a representation language for developing knowledge systems that integrate problem solving and learning.

The result has been the design and implementation of Noos, a reflective object-centered representation language for integrating inference and learning components in a uniform representation.

Our approach builds upon a variety of preceding work: the knowledge-level analysis of knowledge systems, the knowledge modeling frameworks developed for the design and construction of knowledge systems, and the research on integrated architectures for problem solving and multistrategy learning.

1.1 Motivation

One of the key issues in the current development of knowledge systems is the degree to which different components can be described, reused, and combined in a seamless way. Moreover, the current development of knowledge systems increases the necessity of incorporating learning capabilities to knowledge systems. Currently, the integration of learning components is considered as an essential topic for future design, building, and maintenance of knowledge systems.

The difficulties arisen in the development of complex knowledge systems are broadly denominated the *knowledge acquisition problem.* Adapting the approach given in [Aamodt, 1991], our view is that knowledge acquisition is a cyclic process required at the development phase and also at the problem solving phase of a knowledge system (see Figure 1.1).

The goal of the development phase is to perform a knowledge modeling analysis of the knowledge required for solving the problem and to design an application using a specific computer language. First, knowledge modeling methodologies are used to acquire knowledge models from human experts and represent these knowledge models in an implementation independent representation formalism. Specifically, the knowledge-level analysis of expert systems and the knowledge

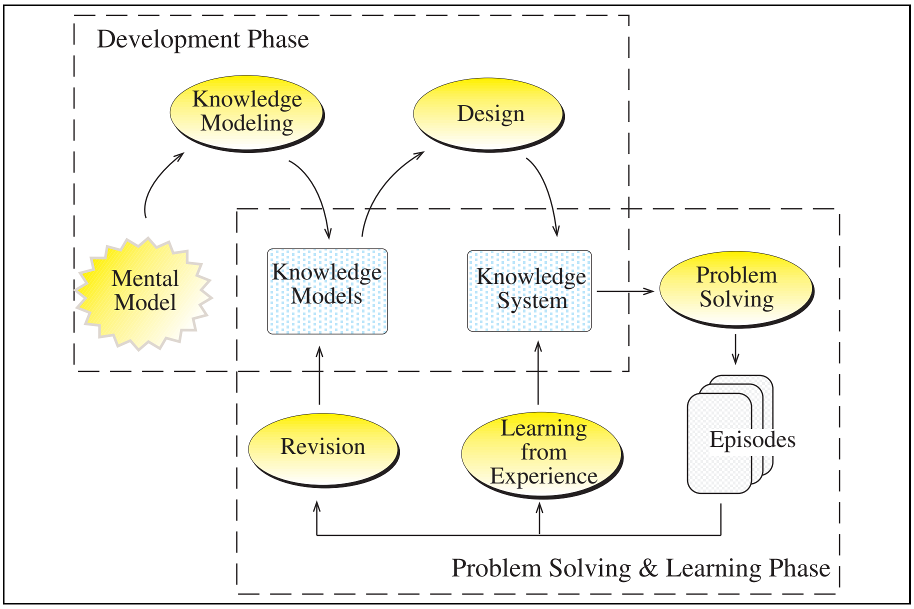

Figure 1.1. The design and maintenance cycle of knowledge systems (adapted from [Aamodt, 1991]).

modeling frameworks are methodologies developed for describing and reusing knowledge systems components. These knowledge modeling frameworks, like KADS [Akkermans et al., 1993], CommonKADS [Wielinga et al., 1993] or Components of Expertise [Steels, 1990], are based on the task/method decomposition principle and the analysis of knowledge requirements for problem solving methods (PSM).

After the knowledge modeling process (see Figure 1.1), a design process has to be performed for translating the knowledge models to a specific language constructing an executable knowledge system. A first issue arises here, since some knowledge models built by the KM methodologies are not directly operationalizable. Different languages—that partially implement the knowledge models built by KM frameworks—have been developed. Nevertheless, an active research activity is focused in developing fully operationalizable representation languages with highly expressiveness capabilities.

Machine learning (ML) techniques have been used by Knowledge Modeling methodologies as a way to acquire certain models in the knowledge acquisition (KA) process conducive to building a knowledge system. Nevertheless, learning is only used in the development phase. That is to say, the knowledge system designed is not to be capable of learning. The reason of this limited use of learning is that the working hypothesis of knowledge modeling methodologies is that the complete knowledge model of the problem can be acquired and modeled statically in the development phase.

Our proposal is that certain knowledge acquisition tasks can be *delayed* from the knowledge system development phase to the phase in which the knowledge system is actually used in the task environment. That is to say, the knowledge system can be designed with learning capabilities. Since knowledge modeling methodologies view KA as a process that basically build models, our approach means that some models are not built in the development phase—or, in general, a preliminary model is build but needs to be improved—and their construction is delayed to the problem solving phase where appropriate ML methods are appointed to generate those models.

The delay of KA tasks has as implication that the knowledge modeling of the implemented knowledge system has to include modeling of KA goals. Moreover, machine learning techniques have to be modeled inside the KM framework and the knowledge requirements of ML have to be addressed. The integration of learning requires the capability of accessing (introspection) to solved problems (that we call *episodes*) and of modifying the knowledge of the system (self-modification).

Moreover, we want to provide a framework for developing—and integrating with problem solving—several symbolic learning methods from the knowledge modeling analysis of application domains.

The contributions of machine learning have grown in the last years and have arisen many different learning techniques. Some examples of learning methods are: empirical learning or inductive learning, analytic learning or explanation based learning, memory based learning, case-based learning, learning by analogy, connectionist learning, genetic algorithms, learning by discovery, etc [Carbonell, 1989, Moreno et al., 1994].

Each different learning method is useful for specific tasks satisfying a set of requirements. For instance, analytical learning methods require a copious domain theory and few examples, while case-based learning methods require many solved cases and may also use domain knowledge. A difficulty with learning techniques is that different learning systems use different representation languages and different knowledge model about the architecture (the self-model).

There are two main approaches adopted to integrate learning with problem solving: (i) combining different learning methods in hybrid systems in order to be applied in a set of broader tasks, like CASEY [Koton, 1989] or BOLERO [López and Plaza, 1993], or (ii) developing a computational framework capable of integrating different learning methods in a uniform way. Examples of this second approach are architectures like PRODIGY [Carbonell et al., 1991] and THEO [Mitchell et al., 1991].

Integrated architectures propose different frameworks for integrating problem solving and learning. A difficulty with integrated architectures is that their representation languages are far from knowledge modeling frameworks (for instance, SOAR representation language is based on productions). Moreover, all these architectures integrate diverse learning mechanisms but present some limitations in their integration.

Our approach in developing Noos was influenced for the research on integrated architectures. Nevertheless, our approach of integration of learning techniques is restricted to symbolic learning techniques: we are focused on the integration of inductive learning techniques, case-based reasoning techniques, and analytical learning techniques. Another issue not covered by this thesis is the revision process (see Figure 1.1) for re-designing knowledge models from the experience acquired in solving problems.

One of the most research activity in machine learning, at the theoretical and practical level, is focused in the integration of several disparate learning techniques [Langley, 1989]. This research is focussed on the development of systems called *multistrategy learning systems* (MSL systems) [Michalski, 1993]. MSL systems use several learning techniques, that are combined using a specific strategy, for learning from a greater variety of inputs and acquire more flexible knowledge. Consequently, MSL systems potentially can be applied to a wide range of problems. Following this direction, our approach in Noos is to provide a representation language for facilitating the integration of several learning techniques in a knowledge system. Moreover, a domain-specific analysis for each Noos application determines different strategies for combining these several learning techniques.

1.2 Goals and contributions of the thesis

The aim of this thesis is the design and implementation of a representation language for developing knowledge systems that integrate problem solving and learning. Our proposal is that this goal can be achieved with a representation language with representation constructs close to knowledge modeling frameworks and with *episodic memory* and *reflective* capabilities.

The second proposal is that learning methods are methods (in the sense of knowledge modeling PSM) with introspection capabilities that can be also analyzed in the same task/method decomposition way. Thus, learning methods can be uniformly represented as methods and integrated into our framework.

The third proposal is that whenever some knowledge is required by a problem solving method, and that knowledge is not directly available there is an opportunity for learning. We call those opportunities *impasses*, following SOAR terminology [Newell, 1990], and the integration of learning is realized by learning methods that are capable of solving these impasses.

We have developed the Noos representation language, a language close to knowledge modeling frameworks, for developing knowledge systems that allows the integration of problem solving and learning.

Furthermore, Noos is formalizated using *feature terms*, a formal approach to object-centered representations, and providing a uniform formalism for integrating different learning techniques.

Noos is a language with capabilities of :

Representation : Noos is based on task/method decomposition principle and the analysis of knowledge requirements for methods. This capability allows

Noos to take advantage of the KA methodologies and libraries developed in KM frameworks. There are several specification languages developed in KM frameworks that can be used for knowledge modeling of applications. A knowledge modeling analysis realized in some of these specification languages can be then implemented in Noos. Nevertheless, this operationalization step is not structure-preserving with respect to knowledge modeling frameworks.

An object-centered representation language : Noos provides a structured representation of knowledge allowing the description of complex knowledge. Moreover, the Noos approach to knowledge representation provides a natural way to describe partial knowledge amenable to extension.

Introspection : Noos inference behavior is partially represented in the language itself. This reified behavior can be accessed by problem solving methods written in Noos. Introspective capabilities form the basis that allows Noos programs to reason about the system behavior. Learning methods are methods that use introspection.

Episodic memory : problems solved in Noos are automatically memorized (stored and indexed) and amenable to be accessed and reused in solving new problems. The problem solving behavior in these solved problems is also represented in Noos and, using introspection, is accessible and inspectable. Episodic memory can be inspected by means of three different access mechanisms: *access by path*, that provides a way to access to specific portions of the episodic memory; *retrieval methods*, that provide a mechanism for content-based access to the episodic memory; and *perspectives*, a mechanism to describe declarative biases for case retrieval in the structured representation of cases. The memorization of solved problems is a basic building block—together with introspection—for integrating learning into our KM framework.

Integrated Problem solving and learning : the integration of learning in Noos is based on two proposals:

- Learning methods are methods (in the sense of knowledge modeling PSM) that can be also analyzed in the same task/method decomposition way. Introspective requirements of learning methods can be fulfilled using Noos reflective operations for accessing the episodic memory.
- *Impasse driven learning*: whenever some knowledge is required by a problem solving method, and that knowledge is not directly available, there is an opportunity for learning. We call those opportunities *impasses*, following SOAR terminology, and the integration of learning is realized by learning methods that are capable of solving these impasses.

Machine learning methods : Noos provides a collection of basic mechanisms allowing the development of different learning methods such as inductive learning methods, CBR, and analytical learning methods.

- *Inductive learning methods* are developed in Noos as search methods (that follow certain biases) over the space of feature terms. Domain specific knowledge is used for constructing inductive learning methods that follow different searching biases. Inductive learning methods are based on the feature term subsumption and antiunification operations of Noos.
- *Case-based reasoning methods* are developed in Noos as problem solving methods with lazy learning capabilities that search for previously similar solved problems in the Noos episodic memory. CBR methods are based on the retrieval and subsumption operations of Noos.
- *Analytical learning methods* are developed in Noos as methods that given a training example whose problem task has been solved by a problem solving method M and given an operationality criterion, construct a new problem solving method $\mathtt{M}_{op}$ for solving that task and obeying the operationality criterion. Analytical learning methods are based on the Noos introspective capabilities for inspecting the episodic model built in Noos while solving the training example.

A multistrategy learning approach : different learning methods can be integrated in Noos using a common scheme based on three main subtasks: *Introspection*, *Construction*, and *Revision*. Using this common scheme, we will present how different learning methods can be designed and integrated in a problem solving system.

Noos has been implemented using Common Lisp and currently is running in several platforms. A window-based graphical interface has been also develpped in Macintosh Common Lisp [Digitool, 1996] for the MacOS version of Noos. Although the implementation details are not the main focus of this thesis, we provide a brief description of the Noos development environment in Appendix A.

Moreover, Noos has been used by several persons to develop several applications that integrate different problem solving methods and different learning methods. The research done in one of them, called *Saxex* and developed for myself, has been awarded with the "Swets & Zeitlinger Distinguished Paper Award" at the 1997 International Computer Music Conference.

1.3 Structure of the thesis

This thesis is organized in seven chapters, including this introduction chapter, and four Appendices:

Chapter 2 reviews the main research relevant to our thesis and discuss their contributions and limitations. We present the research on knowledge modeling methodologies for analyzing and developing knowledge systems; the research on integrated architectures for exploring the relationships among problem solving, learning, and knowledge representation; the research on reflective representation languages for providing introspection capabilities on knowledge systems; and finally, discuss the the role of reflection on learning.

Chapter 3 presents the Noos representation language. The language is introduced incrementally. First, The Noos modeling framework is presented. Next, the basic elements of the language such as descriptions, refinement, references, methods, and the basic inference are described. Then, the reflective capabilities of Noos are described introducing elements such as metalevels, tasks, reflective operations, reification, and reinstantiation. Next, a declarative mechanism for decision making about sets of alternatives, called *preferences*, is presented. Finally, the complete Noos inference engine is described.

Chapter 4 presents the Noos capabilities for reasoning about experience and the integration of learning and problem solving. First, the notion of *episodic memory* is introduced. Then, introspective mechanisms such as *retrieval* and *perspectives* are presented. Next, three different families of learning techniques such as case-based reasoning, inductive learning, and analytical learning, are presented with examples of how have been integrated in Noos.

Chapter 5 presents feature terms, a formalism for describing the Noos language. Feature terms are introduced using a syntax notation based on the λN calculus. Then, using the work on feature structures a semantics based on the notion of partial descriptions is presented. The results obtained by the research on feature structures are also adapted for providing several equivalent representations of feature terms.

Chapter 5 introduces a formalism for describing preferences based on the notion of pre-orders. Two kinds of basic operations are defined over preferences: *preference operations*, that take a set of source elements and an ordering criterion and build a preference (a partially ordered set), and *preference combination operations*, that take two preferences and a combination criterion and build a new preference.

Next, using feature terms *perspectives* are defined. Perspectives are formalized as second order feature terms that denote sets of terms.

Finally, we describe formally the inference in Noos using Descriptive Dynamic Logic, a propositional dynamic logic for describing and comparing reflective knowledge systems.

Chapter 6 provides a set of examples of how diverse applications have been developed using Noos by several persons at the IIIA. Specifically, the chapter present six applications developed using Noos: CHROMA, SPIN, SHAM, GYMEL, *Saxex*, and NoosWeb.

Chapter 7 summarizes the main contributions of the thesis and discuss further directions of research.

Appendix A presents the Noos development environment including development facilities such as browsing, tracing, and the extension of the built-in methods.

Appendix B provides a glossary of the main concepts introduced in this thesis.

Appendix C presents the complete syntax of the Noos language, using BNF notation, and the collection of compact descriptions for built-in methods.

Appendix D presents the complete list of Noos built-in methods describing their required features and their functionality.