Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

AKILES: An Approach to Automatic Knowledge Integration in Learning Expert Systems

Alvaro de la Ossa

SEKI Report SR-91-08 (SFB)

AKILES : An Approach to Automatic Knowledge Integration in Learning Expert Systems*

Alvaro de la Ossa
Research Group on Artificial Intelligence (Expert Systems)
Dept. of Computer Science
University of Kaiserslautern
PO Box 3049
D-6750 Kaiserslautern
Federal Republic of Germany
e-mail: delaossa@informatik.uni-kl.de

ABSTRACT

Knowledge integration is defined here as a machine learning task from a practical point of view—by identifying the requirements that a real-world complex application domain poses on the expert system in relation to a changing world. We present our current approach to knowledge integration in an expert system, required when the *structure* of the physical system, the *world* on which the expert system operates *changes*. Our exemplar domain task is *technical diagnosis*. We test our approach on the particular architecture of MOLTKE/3, our workbench for technical diagnosis[1], which integrates second-generation expert system techniques in a unique framework.

Knowledge integration is seen as the task of *elaborating* and *accomodating* new information (due to structural changes) in the expert system's knowledge, maintaining *consistency* in the knowledge base. The main focus is towards improving the *adaptability* of the expert system to the structural changes. The approach is based on three principles from the adaptation process: incrementality, extensive and intensive use of domain knowledge, and focus on strategic knowledge. We discuss how AKILES' knowledge integration task can be used to complete the modeling cycle, i.e., covering the model-evaluation step in the layout-elaboration-evaluation cycle, as defined in [13].

Topics: Learning and Knowledge Acquisition
Knowledge Representation

---

* The present work is a partial description of the dissertation research being done by the author under the guidance of Prof. Dr. M. M. Richter at Kaiserslautern. The author is a fellow student of the German Service for Academical Exchange (DAAD) and of the University of Costa Rica.

1 MOLTKE/3, for "MOdels, Learning and Temporal Knowledge in Expert systems for technical diagnosis", was developed at our research group in Kaiserslautern

1. INTRODUCTION

In this paper we describe our current approach to *knowledge integration* for *expert system adaptation* in the domain of technical diagnosis. The description is given in terms of a real-world problem: the change of structure in a machine. We consider knowledge integration necessary for expert system adaptability.

Before the structure of the machine is changed, diagnostic experience has been acquired and the knowledge has been refined. Practical restrictions imposed by the real-world application may make it impracticable or at least very expensive to generate a totally new knowledge base for the new machine every time a part is replaced with a different one. Besides, that would cause the loss of valuable refinements in knowledge, unless we are able to identify and extract those refinements before generating the new knowledge base and then apply them consistently. Our approach is identifying refinement knowledge and then adapting it to the new machine's structure.

Integrating new knowledge includes revision and evaluation of the expert system's beliefs with relation to the modified world. Basic assumptions may have to be changed and, moreover, the *representational framework* may need of adaptation too. We attempt to close the modeling cycle defined in [13] by providing the means for evaluation of the domain theory.

Section 2 of the article offers a short description of the MOLTKE/3 workbench, the computational testbed for AKILES. Section 3 discusses the requirements posed on the knowledge integration system. Several approaches to knowledge extension, refinement, and integration illustrate the discussion, and some features of our knowledge integration approach are highlighted. Section 4 first presents a functional view of knowledge integration. Then, deepening on previous discussion, it describes how our model satisfies the requirements presented before. Finally, section 5 discusses the innovations of AKILES, its state of realization, and the future research trends.

2. THE COMPUTATIONAL TESTBED FOR AKILES: THE WORKBENCH MOLTKE/3

This section describes the MOLTKE/3 workbench [2]. Due to space limitations, only a brief description of the tools and subsystems conforming the workbench is given. Details about its conceptual approach and methodological development can be found in the references mentioned through the description.

AKILES is designed as a framework for knowledge integration in expert systems on the architecture of the MOLTKE/3 workbench for technical diagnosis. Several other tools have been developed by the Artificial Intelligence Research Group on Expert Systems at Kaiserslautern, which are already integrated within MOLTKE. The shell has been tested with several application domains, such as CNC machine centers[2], CNC 3D-measurement devices and heterogeneous computer networks. The results have been encouraging and have fed back with fruitful experience to revise and enhance MOLTKE's design.

---

2 CNC - for Computerized Numerical Control

In MOLTKE, diagnostic knowledge is structured vertically—an heterarchy of *contexts*, each corresponding to particular diagnostic situations (knowledge about failures: *classification*), and horizontally—knowledge about the diagnostic process (*test selection*) is separated in ordering, shortcut, and diagnostic knowledge, as described later. The knowledge base also contains a causal model of the technical system and a case base of diagnostic cases. The knowledge is processed with several interpreters, allowing for combination of multiple strategies.

2.1. Basic Terminology. The Diagnostic Process in MOLTKE

A *symptom class* relates a name with a list of possible values (e.g., the name `Valve` associated to the set of values `(open, closed)`). A *symptom instance* describes a machine part's state (e.g., `(ValveX1 closed)`). The current value may be unknown or one of the possible values for the class. A *situation* is the set of all symptom instances (e.g., `((ValveX1 closed) (ValveY2 open) (PressureP0 high))`); it (partially) describes the actual state of the technical system. A *formula* stores the current binding of a symptom instance, which is a variable in the predicate calculus. A three-valued logic (`true, false, unknown`) is used to evaluate the formulas in a *formula language*.

Tests determine the current value of one or more symptom instances. *Ordering rules* are used to determine which test to execute next; they have a formula on the left side (test's precondition) and a symptom instance (to be tested) on the right side (e.g., `((is ValveX1 closed) -> DuctD4 test)`). The order of this rules determines the (potential) execution order of tests. *Shortcut rules* represent a relation between symptom values, and shorten the diagnostic process by deriving one or more (unknown) symptom values from one or more (known) symptom values, thus eliminating the need to gather their values through tests.

A *context* : describes a rough, intermediate, or final diagnosis; it has a *precondition* (conjunction –or disjunction of conjunctions– of symptom values) which, if satisfied under the current situation, causes the associated *diagnosis* to become *proven*, in which case the corresponding *correction* is executed. To every context a set of ordering rules and a set of shortcut rules are associated, as well as a *context interpreter*. The locality of ordering rules and the presence of a (local) interpreter allow for especialized strategy for test selection. A default context interpreter uses the ordering rules for that matter.

The space of diagnostic situations represents the search space for diagnosis. It is structured in the *context graph*, whose nodes represent diagnoses, and the arcs *refinement* links between contexts. A *global interpreter* processes the knowledge in a cycle of gathering symptom values and refining to other context when its precondition is satisfied, until a leaf node in the graph is reached.

As a result of the separation of factual diagnostic knowledge (context precondition) and strategy knowledge (ordering rules), besides the use of the domain expert's terminology, the knowledge acquisiton becomes simplified. Two powerful graphical user-interfaces (based on the Browser interface of Smalltalk-80, the implementation language of MOLTKE) are used for editing the structural and behavioral model of the machine, as well as the context graph.

2.2. Model-Based Diagnosis in MOLTKE

In MOLTKE, diagnostic knowledge extracted automatically and directly from the machine's design plans guides diagnosis. The automatic model and diagnostic knowledge compiler MAKE[3] [15] builds up a static model of the machine, including a complete description of the machine's structure and behavior. The structure is represented in a hierarchy of parts. For each primitive part, its behaviors are described as input-output rules relating its ports. Complex parts' behavior is elaborated on the basis of their component parts. The model is checked for consistency and the context heterarchy is build through a simulation process. For each context, its precondition and shortcut rules are generated.

MAKE's output is a basic expert system, which conforms the model-based component of the expert system. It might be later refined by the expert, for instance, by changing the order of ordering rules or modifying shortcut rules.

2.3. Learning in MOLTKE

Experience gained by the expert system is acquired, organized, and refined by the learning component. Experiential knowledge is modeled under a case-based approach. Two subsystems cooperate to enhance the diagnostic task. PATDEX/2 [1,3], a case-based reasoning subsystem, acts interactively, on-line, to enhance the diagnostic process. It retrieves the most similar diagnostic situation (case) to the current one, described by an ordered set of symptom values and a diagnostic hypothesis. Similarity is defined in terms of the symptom values present in the current case. The solution of the retrieved case is mapped to the current case.

The other subsystem, GENRULE [4], acts off-line by compiling heuristic generalization rules from the diagnostic cases. The heuristic rules, which describe partial shortcut rules, are given an statistically-determined certainty factor. During diagnosis, the user determines how many shortcut rules may fire, thus either avoiding uncertain conclusions (i.e., no shortcut rules may fire) or allowing for evaluation of the application of generated rules.

3. REQUIREMENTS ON KNOWLEDGE INTEGRATION

Several approaches can be found in the bibliography for knowledge revision, extension, refinement, and integration. These terms are repeatedly used in different ways, although the most important goal of knowledge integration, and motivation of most approaches, is mainly one: giving the learning system a means of reacting to a changing environment (i.e., its *world*).

In our technical diagnosis scenario, new information means evidence of a structural change in the machine. The expert system should be able of integrating new information and adapting the affected knowledge to cope with the new reality. The following discussion on the requirements imposed on the knowledge integration system highlights

---

3 for Model-based Automatic Knowledge Extractor

features for the system and leaves open some questions that we try to adequately answer in section 4, where we present the functional structure of AKILES.

3.1. Use of Domain Knowledge for Knowledge Integration

Most approaches to knowledge integration emphasize the need for exploiting domain knowledge. *Extensive and intensive use of domain knowledge* are necessary for knowledge integration. Reasoning about change and its impact cannot be guided using only a bunch of heuristic rules that attempt to describe all possible change situations in the world, or a prefixed set of dependency associations between knowledge base objects or object types. Those kind of abstractions are necessary, but we rather think of a system capable of extracting and abstracting from the change situations relevant information that is useful for further identification and elaboration of change situations. Relations between objects, properties, etc. *in the domain* itself should determine dependencies to direct knowledge refinement, revision, or integration, rather than mere syntactic or pseudo-semantic associations of objects or their properties.

Integrating involves the process of elaborating new incoming information. The integration process is defined in an incremental, cooperative manner as a process in which *the expert directs elaboration*, while *the system elaborates and proposes* (moreover, *supervises*) *knowledge integration tasks*.

3.1.1. Extensive Use of Domain Knowledge

With the term *extensive* we mean: (1) *terminological extension* —the extent to which the system defines its reasoning tasks in terms of domain knowledge objects rather than on syntactical properties; (2) *feedback extension* —the extent to which the learning system's output is fed to and used by the domain theory, *and* the extent to which the expert's advice is used for further learning; and (3) *revision extension* —the extent to which identifying knowledge for revision is guided by domain knowledge and spread in the theory.

BLIP, developed under the Sloppy Modeling approach [13] is terminologically extensive. The reasoning task, modeling, is defined in terms of objects in the theory (model), and the terminology (predicates defined in the model) is refined by establishing the user the semantics of predicates through the inclusion of meta-facts. This becomes a refinement cycle in which the system reasons about the adequacy of the representation in terms of the predicates and their semantics.

DISCIPLE [8,9,10], which uses EBL to explain a user's solution, is not feedback-extensive; feedback is only to the learning element —the explanation; the performance element gains only in the verification of classification. BLIP is feedback-extensive: it allows in one direction for model revision and in the other for theory extension; it uses the expert's judgement to extend or reorganize predicates and their semantics, which can be in turn revised again.

Murray's KI [14] is revision-extensive: the search for consequences of new incoming information is guided by inference rules local to a concept's role and by the derived dependencies between domain knowledge objects; thus, possible consequences are identified and presented to the domain expert with plausible resolution hypotheses.

We think of a knowledge integration system as being domain-knowledge extensive. Then, it has to *define its reasoning tasks in terms of domain knowledge*. That includes explaining the expert's advice in terms of domain knowledge objects and their relations. Second, it has to *allow for flexible feedback* from the user to the learning component and from this to the domain theory. This can enable theory revision and includes exploting the expert's advice to expand the search for consequences of new information and to extend the theory (e.g., identification of new concepts, relations, etc.). Finally, it must be able of effectively *identifying knowledge for revision*, and of appropriately *proposing knowledge revision* hypotheses. That involves checking for local consistency and on higher levels of abstraction of the knowledge, and searching for consequences on the representation formalism level.

3.1.2. Intensive Use of Domain Knowledge

Intensity in the use of domain knowledge is *goal-oriented*. In this sense, the term *intensive* means the extent to which the system is itself capable of guiding (i.e., controling) and expanding the search for consequences of new information to concepts, properties, etc. not explicitly contained in the new information. Intensive use of domain knowledge is a requisite for *autonomy* of the knowledge acquisiton task, and therefore of the knowledge integration task.

An expert system is *partially* domain knowledge-intensive if it guides search with only a manually prefixed set of general heuristics. As said before, that kind of abstractions (e.g., principles for reasoning about evidence and the associated repair strategies in CASEY [11]) or dependency associations (e.g., the dependency network for knowledge base maintenance and consistency checking in MOLTKE/3 [12]) are necessary to search for consequences of change. Indeed, such abstractions might have been elicitated from the domain expert [12].

But we are looking towards exploting domain knowledge for forming and refining search heuristics. For dependency associations, for instance, there still exists the need of *traducing dependencies* between knowledge structures or between knowledge objects *into causal relationships* which can allow interpreting knowledge interactions on more abstract levels [7].

Under the above view of intensity, Murray's KI is knowledge-intensive. The inference rules used to search for consequences of new information are domain-dependent and local to the concept's perspective (role) chosen by the expert. PROTOS [5], which introduces pieces of domain knowledge into the training, is knowledge-intensive too. It allows for integration of the training in the knowledge base at various levels of abstraction, as the training may now include new meta-knowledge for the domain theory. Another knowledge-intensive approach is that of Bradzil and Torgo [6]. They propose integrating independent knowledge bases. The selection of rules for the final theory is guided by a characterization of the competing theories, under an experimental approach: rules are tested against (training) cases generated from available domain data.

3.2. Support for Construction and Refinement of the Theory

A knowledge integration system must aid the knowledge acquisition system in the task of supporting construction and refinement of the domain theory. The knowledge acquisition bottleneck cannot be solved without satisfying this requirement. The knowledge integration task is the operationalization of a kind of learning tasks characterized by two features: elaboration (for knowledge expansion), and refinement. Also, the third step in the modeling cycle—evaluation of the model, must be covered by the knowledge integration task, and this step cannot be characterized *per se* under those two features[4].

The support for domain theory evaluation in a unique framework is indispensible to ensure the proper feedback of the evaluation for revising, especially, the representation; this means inferring on the terminological level possible deficiencies of the expert system's performance due to inappropriate knowledge structures or relations between knowledge structures with relation to a certain reasoning task.. This is essential for expert system adaptation, and is only possible if we have enough understanding of the purpose that knowledge structures serve for the given task.

The knowledge integrator should be able of predicting the theory's behavior in order to evaluate it. It must be able of testing the theory, and moreover, its own ability for integrating knowledge. That means for us generating experiments whose results can serve for predicting possible failures of the system's performance element due, among other reasons, to inadequate representations, inadequate terminology, or incomplete/incorrect knowledge integration.

The tasks of domain theory construction and refinement should be defined in an interactive, incremental manner. Incrementality is natural to refinement. More attention should be paid on the domain knowledge in order to allow for revision and reversability at each step of the refinement process. We approach this by extending knowledge base maintenance to domain-dependent dependencies between objects in the domain theory.

A final and most important property to discuss is reversability. The scope of this paper does not permit going depper in discussing this point. However, it is important to notice that reversability must be identified on two levels: on the domain knowledge (de-/composition of beliefs) and on the representation level (for representation revision).

In summary, the system must support construction and refinement of the theory, by (1) enabling *evaluation of the theory* on both the knowledge and on the representation level, essential for expert system adaptability; (2) focusing *refinement* with *revision as the goal*; (3) extending knowledge base maintenance with *domain-dependent dependencies* between objects; and (4) allowing for *reversability of knowledge integration* and of representation revision.

---

4 rather, evaluation is seen as a separate task with feedback to the modeling task itself, as in KEW [16]

3.3. The Knowledge Integrator as an Assistant to the Domain Expert in the Knowledge Integration Task

Another requirement reaches the confines of the knowledge acquisition system, and is necessary for knowledge integration to succeed. Knowledge integration is limited to the expert's advice if it does not *understand* the world, down from the domain's first principles. The knowledge integrator should act as an assistant of the domain expert in the theory refining process. The knowledge acquisition system allows the expert system to acquire and learn from the expert's expertise, but it is necessary to introduce domain knowledge necessary to explain that expertise as it is acquired. Without this, the assistant cannot go beyond following the domain expert's advice and serving as an interface between the expert and the theory.

To some extent, acquisition of expertise and knowledge necessary to understand the domain is already present in MOLTKE. The generation of shortcut rules from diagnostic cases represents expertise acquisition, in the sense that the domain expert decides whether shortcut rules may or may not be applied, and this information is recorded. It also represents acquisition of knowledge necessary to understand the domain: the semantics of shortcut rules may be interpreted as relevance knowledge about symptoms in particular diagnostic situations.

Knowledge integration should be justified on the domain (or on an elaboration of the expert's advice, which is endly justified on her expertise in the domain). The acquisition and elaboration of expertise and of knowledge necessary for understanding the domain, must be extended to allow for discovering (first-) principles in such a way that they can be used to justify knowledge integration.

4. KNOWLEDGE INTEGRATION

Knowledge integration means integrating new information in such a way that its consequences for the current knowledge are resolved to maintain the integrity of the knowledge base. In this paper, we describe our general knowledge integration framework within MOLTKE's approach to technical diagnosis. Figure 1 shows our functional view of knowledge integration for Moltke. This view is fully discussed in the next subsections.

4,1. The Knowledge Integration Task

Knowledge integration subsumes extension, which, as defined in approaches like DISCIPLE, means deriving new knowledge that either was implicit in the knowledge base, or is implicitly contained in user-presented examples. It implies revision, and thus, knowledge integration must also subsume revision.

Some of the approaches used to illustrate the discussion in section 3 have already argued, within their own goals and terminology, the validity of knowledge integration as a learning task. For Murray [14], new knowledge is integrated paying attention to the consequences of the training for consistency of the domain theory. Brazdil & Torgo [6] focus the problem on integrating diverse knowledge bases into one base, paying attention on the competition arised between knowledge bases. Morik [13] presents an approach that

allows for model construction, and the focus is on allowing for refinement and revision.

We agree with Brazdil and Torgo's view: integrating knowledge involves adjudicating among competing sources. Murray claims in this sense, that elaborating the training involves solving this problem. And while the literature on BLIP might not discuss the issue, it is possible to set BLIP's environment around competing theories and letting the modeling task adjudicate among them. The differences between those approaches are in perspective or presentation.

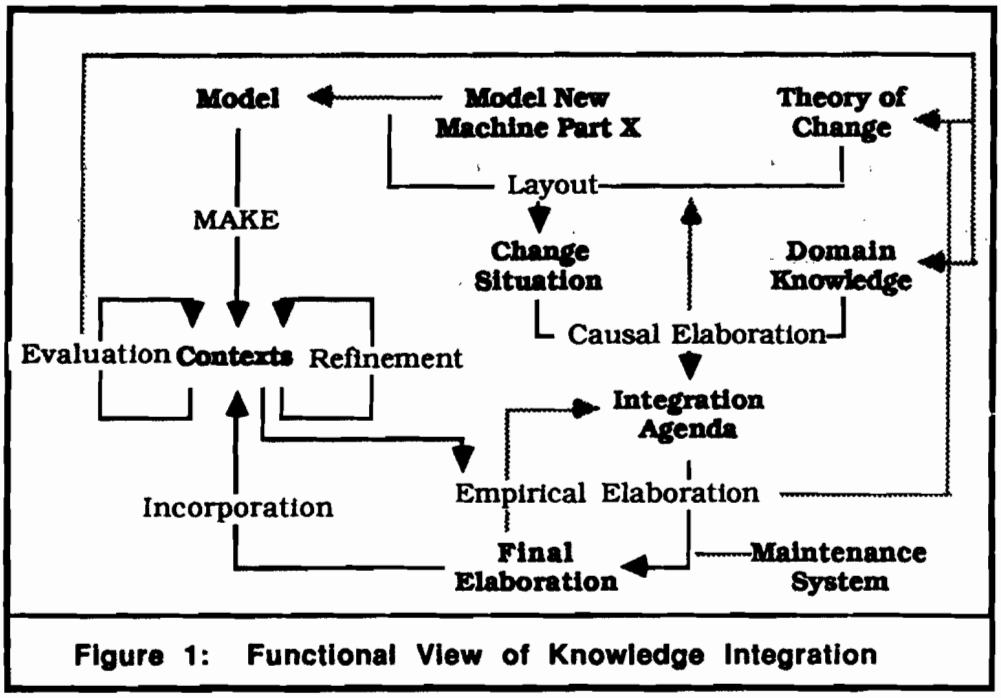

Figure 1: Functional View of Knowledge Integration

4.1.1. Identification and the Theory of Change

In order to integrate knowledge, we need to know which knowledge in the knowledge base will be affected, and how, by the insertion of new knowledge. The first step in knowledge integration is that of *identification*. For this, we create a *theory of change*, which serves as a terminological basis for explanation of the cause-effect relationships between incoming and existing information. We describe this in terms of our technical diagnostic expert system.

Changing the structure of the technical system generates inconsistencies. For instance, diagnostic knowledge about machine components which become physically connected to the new part may not be properly related to the diagnostic knowledge generated for the new part. Replacing a machine's part implies introducing in the domain theory the model of the new part and the associated diagnostic knowledge about that part and its subparts. After this, the diagnostic knowledge will only be related to other knowledge in the knowledge base by diagnostic-context refinement links associated directly to the structure of the machine (i.e., a refinement link will only exist if the "refined-to" machine part is a subpart of the owner in the relation).

The knowledge base maintenance component of MOLTKE is able of identifying and

correcting syntactical and semantical *domain-independent* inconsistencies in a dependency network. But domain-dependent inconsistencies can only be identified and solved on the basis of domain-dependent dependencies between objects in the knowledge base. Thus, for knowledge integration we extend the consistency maintenance component of MOLTKE to include *justification* links between instances of object classes in the knowledge base. Our *incremental revision-oriented refinement* approach focuses, on the deepest level of revision, those justification links, to test for (domain-dependent) consistency.

A justification is semantically seen as one object being the (domain-dependent) *reason* for the existence of other objects in the dependency network. Justification links are typed. The type of a link is domain-dependently determined by features in the structure and behavior descriptions of the related machine parts and by first principles in the domain. This view allows for unrestricted extension of the dependency network and implies the need of learning about justification link types.

The theory of change describes a *change situation*: the set of objects in the theory with more than one justification link to any other object, being those justification links equivalent. That set of objects is expanded through the justification links of one of its members until no more justification links exist. The result is a subgraph of the dependency network. At this point, we still do not address complexity and efficiency concerns in processing a change situation. For now, we will assume a way of indexing the nodes in the graph (the objects in the domain theory) which have two or more equivalent justification links to other nodes.

When a machine part is replaced, some objects in the dependency network obtain new justification links. The first objects to be identified for revision are those having justification links corresponding to different machine's structures. That is, two justification links of *equivalent* types offer two different justifications for the existence of an object in the domain theory. The similarities and differences arising from those justifications (relevance knowledge) are the source for the elaboration of the knowledge for integration. The definition of equivalence of two justification link types is derived from the next discussion.

4.1.2. Step 2 : Causal Elaboration — Proposing the Integration

Given a change situation (the *layout* of a change in the machine's structure), we proceed to elaboration. Causal elaboration's goal is the *comprehension* of the structural change in the machine. It is not before we understand in what consists a change that we can proceed to incorporate it in the domain theory.

Understanding a change includes comparing the models of the replaced and the replacing machine parts, and comparing their relations to other parts. The comparison focuses the identification of relevant changes in behavior that may cause, for instance, violations to first principles or a significant shift of focus to a different set of principles. The replaced and replacing parts are assumed of *similar* functionality, and thus the violations or shift to different domain principles are used to learn about analogical classes of machine parts. Inducing new *functionality* classes is used to guide knowledge integration for future change situations under an analogy-based approach.

The output of causal elaboration is a proposal for knowledge integration as an agenda of integration tasks identifying nodes in the dependency network and the operations that must be carried out on them to incorporate them consistently in the domain theory. The agenda supervises a cooperative, incremental process of integration, in which the domain expert directs elaboration, that is, her advice is used for deciding which task should be executed next, and the knowledge integrator presents to the expert, elaborates, and processes the integration tasks. Strategic knowledge is needed to construct integration tasks from the results of the comparison of machine parts. This knowledge is acquired from two sources: directly from the domain expert, or by induction from the domain principles. Our dialogue with the experts has shown that this kind of knowledge is so deeply intricate in their expertise, that elicitating it with reasonable explanation of the advice is almost impossible. This task requires of long periods of training in which the domain expert herself extracts the relevant information to construct a model of knowledge integration. That is impracticable in our real-world application setting. Therefore, we cope with this task by defining a primitive set of *adaptation strategies* that associate patterns of similarity or difference in structure and behavior with transformation operations. This set is later extended using the domain expert's advice.

4.1.3. Step 3 : Empirical Elaboration through Experimentation

Empirical elaboration evaluates the results of causal elaboration. We approach this through *cooperative experimentation*, by testing pieces of elaboration using dynamically generated experiments to be conducted in cooperation with the expert. An experiment begins with the proposition of a diagnostic situation and the simulation of the corresponding diagnostic task with the elaborated (now hypothetical) diagnostic knowledge. *Cooperative* means that the generation of the experiment, as well as the interpretation of results and consequent revision of the hypothetical diagnostic knowledge, are carried out by both the system and the expert. The system proposes experiments, presents evidence, and realizes the diagnostic simulation, and the domain expert gives advice to whether the proposed experiments need of modification, asks for evidence (thus potentially biasing the search implicit in the elaboration of the experiment), and directs the revision task derived from the experiment.

In our diagnostic expert system, integration includes, among others, modifying the machine's model, changing the diagnostic contexts accordingly, revising shortcut and ordering rules, and, most important, predicting the consequences of change for experiential knowledge. The elaboration of a change for contexts is of a different nature as that for the experiential knowledge. And both elaborations should lead to evidence that can support or justify each other.

Causal elaboration is defined as the task of predicting the consequences of structural changes for behavior and for diagnosis (the main problem is maintaining the context heterarchy's integrity). *Empirical elaboration* is the task of incrementally identifying experiential knowledge for revision and elaborating the consequences of its revision and of causal elaboration for experiential knowledge again. Thus, causal elaboration and empirical elaboration are complementary, while the integration steps of layout and the whole elaboration form a feedback, incremental cycle, in which identification leads to elaboration and this, in turn, to further identification.

4.1.4. Step 4 : Incorporation and Final Evaluation

The final step is the incorporation of (conflict-free) elaborated knowledge into the domain theory. Incorporating means storing or reorganizing. In the best case, the elaborated information is simply stored without affecting consistency (at least local, for instance, to a diagnostic context). But it is reasonable to expect that in most of the cases the elaborated knowledge can be only tentatively stored and waiting for later test. The integrator acts incrementally, allowing for accommodation of pieces of an elaboration. A piece of elaboration may only be accommodated if at least local consistency is guaranteed. Still then, the knowledge is tentatively stored until global consistency can be reached.

Incorporation is an iterative process in which the results of elaboration are evaluated before proceeding to accommodation. The evaluation leads to a degree of confidence of the elaboration. For this, experiments can be generated that include qualitative simulation of related machine parts' behavior, propagation of consistency constraints through the context heterarchy, among others.

We view incorporation split in four different tasks: *experimentation*, *prediction*, *conflict resolution*, and *accomodation*. Experimentation has been described. Prediction involves interpreting the results of experimentation. An interpretation is used to evaluate elaboration. Conflict resolution means solving contradictions that arise from elaboration or from experimentation and prediction. Solving a contradiction means giving credit or blame to alternative explanations of the contradiction. Explanations are rated and presented as conflict resolution hypotheses to the user.

4.2. Expanding the View to Expert System Adaptation

The functional model for knowledge adaptation described so far has been discussed on a conceptual level of knowledge. A more abstract view is needed to describe expert system adaptation. We view adaptation as the task of changing or *evolving* the representation in the presence of evidence that the expert system's domain task fails because of shortcomings of the current knowledge structures, which limit the expressive power of the domain theory.

A means of evaluating the representation is needed in order to propose modifications to enable enhancements in the system's performance. The knowledge integration task defined here is an evaluation instrument suitable for evaluation of the representation.

In this sense, a change situation represents a proposed change in the representation; identification is the task of determining which knowledge structures are affected by a change for some kind of knowledge; causal elaboration extends to finding conflicts on the domain task level (system's performance), and to propose transformation remedials; and empirical elaboration is a means of testing the remedials.

5. DISCUSSION

5.1. Innovations

Our approach establishes many important and diverse requirements that the knowledge integration task must cope with in order to allow for expert system adaptability to structural changes in its world.

Our approach exploits domain knowledge extensively by elaborating on domain-dependently defined reasoning tasks. Feedback is ensured from the learning tasks to the domain theory and from the expert's advice to further learning. And revision is enabled at all levels and on each step on the integration process. The approach exploits domain knowledge intensively by extending knowledge base maintenance with domain-dependent dependencies between domain objects in the knowledge base. Maybe the most important innovation of AKILES is offering a means of elavuating the results of knowledge integration on two levels: on the knowledge level, i.e., the integrated knowledge, and on the representation level, i.e., the terminology and basic assumptions about the domain knowledge.

5.2. State of Realization and Future Work

The MOLTKE/3 workbench is fully implemented. Its implementation language is Smalltalk-80, and it runs on SUN 3/60 and 3/80, HP 9000, Apollo, Mac II, and other workstations. The basic maintenance system is being implemented.

Currently, the subsystem to compare machine parts models to extract relevance knowledge for the identification step is being developed. Also, an incremental version of the model compiler MAKE is being designed. Scenarios for knowledge integration from various domains have been gathered and from there test knowledge bases for AKILES are being prepared. The causal elaboration subsystem is being developed. Today only small prototype programs have been designed to test fragments of causal elaboration. Empirical elaboration is still on the conceptual phase. We plan to have a fully implemented version of the elaborator by the end of May of 1991. The rest of AKILES, that is, the evaluator and integrator, are planed for implementation during the summer of 1991.

ACKNOWLEDGEMENTS

I want to thank the guidance and comments of my dissertation guide, Prof. Dr. M. M. Richter, without which my work wouldn't have benn possible. The multiple, fruitful "brainstorming" sessions with my colleagues Klaus Althoff, Frank Maurer, Ralf Trapphöner, Stephan Weß, Hans Lamberti, and others have helped me maintaining my hope of a good work. And I don't want to leave unmentioned my wife, Yailly, without whose encouraging help I wouldn't have standed long nights of reading, just thinking, and "modeling" my thoughts.

BIBLIOGRAPHY

[1] Althoff, K-D.; De la Ossa, A.; Maurer, F.; Stadler, M.; Weß, S.: Case-Based Reasoning for Real-World Applications. In: Proceedings of the FAW Workshop on Adaptive Learning and Neural Networks 1989, Ulm, Federal Republic of Germany, 1989.

[2] Althoff, K-D.; Maurer, F.; Rehbold, R.: Multiple Knowledge Acquisition Strategies in MOLTKE. In: Proceedings of the 1990 European Knowledge Acquisiton Workshop, EKAW-90.

[3] Althoff, K-D.; Maurer, F.; Weß, S.: Case-Based Reasoning and Adaptive Learning in the MOLTKE/3 Workbench for Technical Diagnosis. Submitted to IJCAI-91.

[4] Althoff, K-D.; Trapphöner, R.: GENRULE: Learning of Shortcut-Oriented Diagnostic Problem-Solving in the MOLTKE/3 Workbench. Technical Report, University of Kaiserslautern, Federal Republic of Germany, October 1990.

[5] Bareiss, E.R.; Porter, B.W.; Wier, C.C.: PROTOS: An Examplar-Based Learning Apprentice. In: Proc. of Fourth International Workshop on Machine Learning, Univ. of California, Irvine, June 1987, pp.12-23.

[6] Bradzil, P.B.; Torgo, L.: Knowledge Acquisition via Knowledge Integration. In: Wielinga, Boose, Gaines, Schreiber, van Someren (Eds.): Current Trends in Knowledge Acquisition, IOS Press, Amsterdam, 1990 (Proc. of European Knowledge Acquisition Workshop 1990), pp.90-104.

[7] Geiger, D.; Paz, A.; Pearl, J.: Learning Causal Trees from Dependence Information. In: Proc. of AAAI-90, pp.770-776.

[8] Kodratoff, Y.; Tecuci, G.: DISCIPLE: An Interactive Approach to Learning Apprentice Systems. Research Report 293, Université Paris-Sud, Laboratoire de Recherche en Informatique. Orsay, 91405, France.

[9] Kodratoff, Y.; Tecuci, G.: What is an Explanation in DISCIPLE? In: Proc. of Fourth International Workshop on Machine Learning. Univ. of California, Irvine, June 1987, pp.160-166.

[10] Kodratoff, Y.; Tecuci, G.: DISCIPLE-1: Interactive Apprentice System in Weak Theory Fields. In: Proc. of IJCAI-87, pp.271-273.

[11] Koton, P.: Reasoning About Evidence in Causal Explanations. In: Proc. of AAAI-88, pp.256-261.

[12] Maurer, F.: Knowledge Base Maintenance and Consistency Checking in MOLTKE. Submitted to IJCAI-91.

[13] Morik, K.: Sloppy Modeling. In: Morik, Katharina (Ed.): Knowledge Representation and Organization in Machine Learning. Leacture Notes in Artificial Intelligence 347, Springer-Verlag, Berlin, 1989, pp.107-134

[14] Murray, K.S.: KI: An Experiment in Automating Knowledge Integration. Proposal for Ph.D. Dissertation Research. Artificial Intelligence Laboratory Report AI88-90, Univ. of Texas at Austin, October 1988.

[15] Rehbold, R.: Die Integration von modellbasiertem Wissen in technische Diagnose-expertensystemen (in english: The Integration of Model-Based Knowledge in Technical Diagnosis Expert Systems). Dissertation. University of Kaiserslautern, Federal Republic of Germany, 1991 (to appear).

[16] Shadbolt, N.; Wielinga, B.: Knowledge-Based Knowledge Acquisition: The Next Generation of Support Tools. In: Wielinga, Boose, Gaines, Schreiber, van Someren (Eds.): Current Trends in Knowledge Acquisition, IOS Press, Amsterdam, 1990 (Proc. of European Knowledge Acquisition Workshop 1990), pp.313-338.