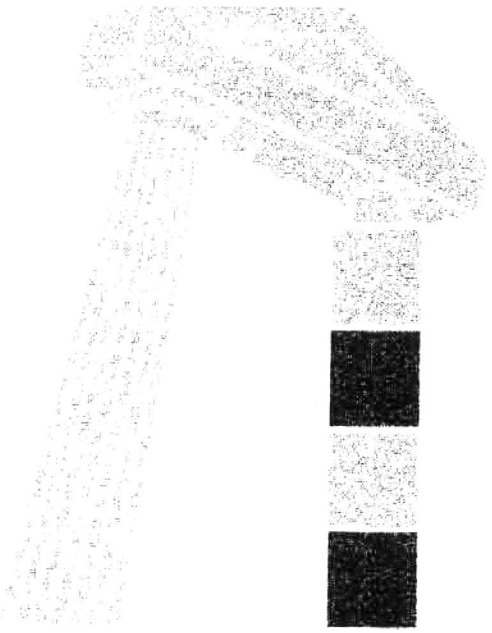
EWCBR-93

First European Workshop on Case-Based Reasoning



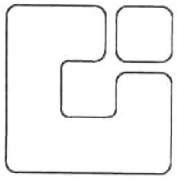
Posters and Presentations

- Volume II -

M. M. Richter, S. Wess,
K.-D. Althoff, F. Maurer (Eds.)

1 - 5 November 1993

University of Kaiserslautern (Germany)

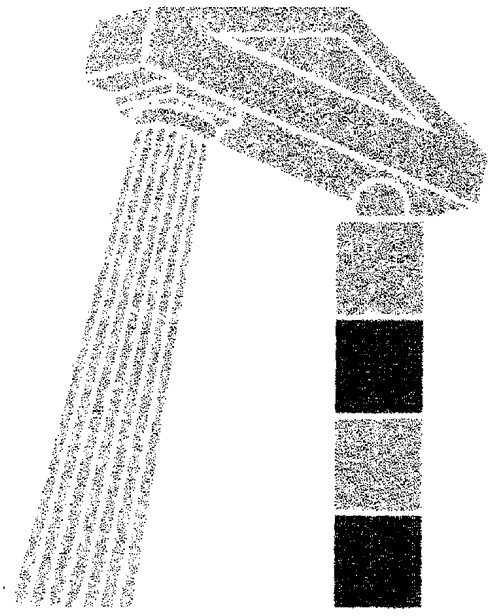


ECCAI

SFB 314

EWCBR-93

First European Workshop on
Case-Based Reasoning



Posters and Presentations

- Volume II -

M. M. Richter, S. Wess,
K.-D. Althoff, F. Maurer (Eds.)

1 - 5 November 1993

University of Kaiserslautern (Germany)

ECCAI

german acm chapter

SFB 314

First European Workshop on Case-Based Reasoning (EWCBR) Presentations and Posters Volume II

Michael M. Richter, Stefan Wess,
Klaus-Dieter Althoff, Frank Maurer (Eds.)

SEKI Report SR-93-12 (SFB 314)
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-67653 Kaiserslautern
Germany

Program Chair

Prof. Dr. Michael M. Richter
University of Kaiserslautern
Department of Computer Science
P.O. Box 3049
D-67653 Kaiserslautern, Germany

Program Committee

Agnar Aamodt	(Trondheim, Norway)
Jaime G. Carbonell	(Pittsburgh, U.S.A.)
Thomas Christaller	(Sankt Augustin, Germany)
Boi V. Faltings	(Lausanne, Switzerland)
Klaus P. Jantke	(Leipzig, Germany)
Mark T. Keane	(Dublin, Ireland)
Janet L. Kolodner	(Atlanta, U.S.A.)
Michel Manago	(Paris, France)
Ramon Lopez de Mantaras	(Blanes, Spain)
Bernd Neumann	(Hamburg, Germany)
Bruce W. Porter	(Austin, U.S.A.)
Frank Puppe	(Würzburg, Germany)
Lorenza Saitta	(Torino, Italy)
Derek Sleeman	(Aberdeen, UK)
Gerhard Strube	(Freiburg, Germany)
Walter Van de Velde	(Brussels, Belgium)

Additional Reviewers

Klaus-Dieter Althoff	(Kaiserslautern, Germany)
Kerstin Dautenhahn	(Sankt Augustin, Germany)
Pete Edwards	(Aberdeen, UK)
Wolfgang Gräther	(Sankt Augustin, Germany)
Thomas Hemmann	(Sankt Augustin, Germany)
Beatriz Lopez	(Blanes, Spain)
Frank Maurer	(Kaiserslautern, Germany)
Rüdiger Oehlmann	(Aberdeen, UK)
Enric Plaza	(Blanes, Spain)
Luigi Portinale	(Torino, Italy)
Barbara Schmidt-Belz	(Sankt Augustin, Germany)
Jürgen Walter	(Sankt Augustin, Germany)
Stefan Wess	(Kaiserslautern, Germany)

Organizing Committee

Stefan Wess, Klaus-Dieter Althoff, Frank Maurer
University of Kaiserslautern
Department of Computer Science
P.O. Box 3049
D-67653 Kaiserslautern, Germany
Tel.: +49 631 205 3360 (3362,3356,3363)
Fax: +49 631 205 3357
email: ewcbr@informatik.uni-kl.de

Organization

EWCBR-93 is organized by
the Expert System Section of the German Society for Computer Science (GI),
the Special Interest Group on Case-Based Reasoning (AK-CBR),

in cooperation with
the European Coordinating Committee for AI (ECCAI),
the German Chapter of the Association for Computing Machinery (ACM),
the Computer Science Department of the University of Kaiserslautern,
the German Special Research Investigation on Artificial Intelligence
and Knowledge-Based Systems (SFB 314) of the DFG,
the German Research Center on Artificial Intelligence (DFKI), Kaiserslautern

Contents

Chapter 5

Case-Based Design / Case-Based Planning

Combining CBR and Constraint Reasoning in Planning Forest Fire Figthing[1]

P. Avesani, A. Perini and F. Ricci
Istituto per la Ricerca Scientifica e Tecnologica
38050 Povo (TN)
Italy
e.mail: {avesani,perini,ricci}@irst.it

Introduction

In this poster we shall illustrate a work in progress aimed at developing an integrated system for planning first attack to forest fires. It is based on two major techniques: case based reasoning and constraint reasoning. The architecture we propose is part of a more extended system that is aimed at supporting the user in the whole process of forest fires management. The novelty of the proposed system is mainly due to the use of advanced techniques for the development of the man/machine interface based on the representation of the user tasks' structure, the integration of more traditional techniques for data analysis with up to date techniques for data classification developed in the AI community, the extensive application of the Case Based paradigm to the planning of the first attack and the integration of the case based reasoner with a contraint solver, mainly in charge with temporal reasoning.

Many computerised systems have been proposed and developed to help the responsible organisations in dealing with some of the phases of forest fire management: prevention, suppression, control. A part from many systems based on traditional techniques (GIS, Spreading fire models, resources management with data base) very few AI rooted applications have been developed. We cannot avoid quoting the Phoenix project [CGHH89], which is real-time adaptive planner that manages forest fires in a simulated environment, and the system developed by P. Kourtz that addresses the problem of dispatching waterbombers, helicopters and crews for fire control in Quebec [Kourtz87]. None of them use CBR techniques, the first one is a research on the design of intelligent agents and the second is a classification system based on rules implemented in PROLOG.

In this extended abstract we shall illustrate the operational context in wich the system will be deployed and the intervention planning approach. At the moment we are terminating the design phase and we shall start the develoment in the next September.

The operational context

In this subsection we shall address the operational context in which the system will be deployed. This description will make the reader aquainted with the typical tasks in charge to the user. The user of the system is the controller based in a provincial centre. His tools are: a workstation, a dedicated line to acquire data from infrared sensors and meteo sensors, a radio, a fax, a telephone and a printer. The system running on the workstation comprises a Geographic information System, a graphical simulator of the fire evolution, tools for territorial, meteo and resource assessment and a module for supporting the intervention planning and control.

When a new fire is reported, the alarm is promptly validated and the situation assessed by the user possibly running the propagation module. On the screen the operator can look at the output of the propagation module and access, through a graphical interaction, information on the graphical symbols showed by the map. At the end of this phase the operator has acquired enough information for drawing on the map a number of line sectors that subdivide the original fire front. The system runs a set of functions that compute the relevant data for each sector. The system presents these data to the operator and the operator may confirm this segmentation or revise his choice.

Once the sector has been identified on the map the operator is now looking for a plan to fight the fire in each sector and that achieves some objectives. The plan may use air forces and/or ground means, and they have to adopt a specific scheme of work. Searching in a data base of past sector plans, the system retrieves a set of plans that achieves these objectives. Follows a modification phase to fit these plans to the current situation. The plans are showed to the operator by means of a predefined form. After this phase of sector plan evaluation the operator may choose to repair a plan editing some subpart. Otherwise he may propose a new one, that seems applicable, based on his experience. The system will provide to check the numeric consistency of the repaired plan: that is verify the constraint on temporal relation, water quantity and resource availability.

The operator now has a set of alternative sector plans for each sector. The system may suggest some combinations and may also verify the combination chosen by the operator. The operator composes on the screen the sector plans using the system as a constraint checker. Furthermore we may also require the system to propose complete plans and to compare different approaches. The system does not select a unique plan, but list a number of plans that differ on the cost, on the expected territory burned by the fire, on the use of specific resources such as water bomber or helicopters.

[1]The research was supported by the Esprit project CHARADE n. 6095.

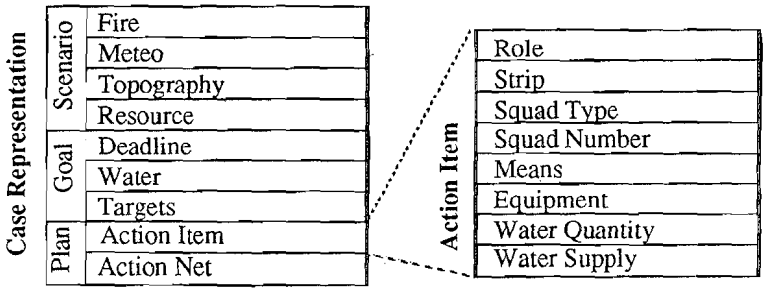
At this poin the operator has to assign specific actions to specific means. He selects the squads, choosing the base and the leader, he must assign to each squad the necessary means. He sets up a contact with the external organisations that are needed for help. He requests the cooperation of the Regional Centre asking for state owned air resources and further resources from other Provinces or Compartments. He requests the cooperation of the Prefecture for the intervention of the Army. He assures himself of their readiness to help. The operator finally takes a decision: selects a plan and sends the appropriate orders to the bases closing the planning of the first intervention.

The Intervention planning approach

The intervention planning system rests on a CBR module integrated with a Constraint Reasoner module.It works on sector plans whose representation consists of two main parts. The first one allows to efficiently associate the current situation to an old, similar one among those recorded in the historical database of interventions. The second one contains a description of the structure of the plan in terms of action and their temporal relations. Constraint propagation techniques are applied to this part of the plan representation in order to support adaptation and repairing of a sector plan. Moreover constraint reasoning techniques support composition of sector plans into a global plan and resource allocation providing a plan instance specific to the current situation.
In the following we shall discuss provisional representational choices for plan representation and describe how the system will support the main decisional steps of the planning process above stated.

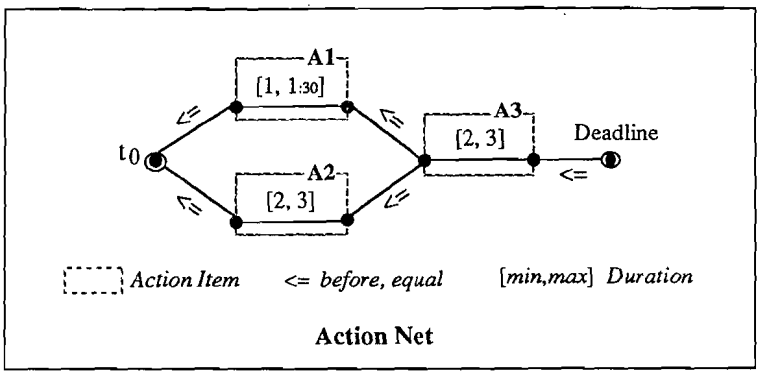
Sector plans

A sector plan is defined by a set of features describing the physical environment and by the structured set of actions that have to be performed, as depicted in Figure 1a.
The sector plan features include a description of the scenario and the goal. The scenario is defined by a set of fire parameters, the most critical to evaluate the fire danger, the meteo conditions, the topography of fire location and the available resources that are located in the bases close to the fire. The goal includes three basic objectives of the global plan: the intervention deadline, the water equivalent of the fire fight, the targets of the fire. The intervention deadline is the maximum time in which the plan has to be completed. The water equivalent of the fire fight is an estimate done by the fire expert to dimensionate the global intervention. The targets of the fire are a set of valuable things to protect, for instance a building or a well.



Figures 1a, 1b

The scenario corresponds to the initial condition and the goal to the final condition in classical planning formulation. Sector plan actions identify the fire fighting tasks that should be performed by a set of squads with appropriate equipment or means, see picture 1b. Fire suppression by ground attack using noozles or air attack with helicopters are typical examples of fire fighting actions. The structured set of actions describes the temporal dimension of the plan. It contains information on action durations, possible time constraints respect to the starting and ending times of the plan, and temporal relation between actions. We represent it by a graph whose nodes correspond to starting and ending times of actions and whose arcs are labelled with the temporal relation between the connected nodes following the approaches presented in [DMP91], [vanBeek92]. We shall call this structure action net (see Figure 1c for an example). For instance the arc connecting t_0 and the starting time of the action A_1 is labelled with $\{<, =\}$ representing the information that A_1 must start at or after t_0. The arc connecting the starting time of A_1 with its ending time is labelled by the interval [1 1:30] meaning that the duration of A_1 will take a value belonging to that interval, i.e. $1 \leq t_{end} - t_{start} \leq 1{:}30$ hours.



Figures 1c

The intervention planning steps

In the following we shall describe how the system supports the main decisional steps of the planning process, as illustrated in Figure 2 and described in a previous section. The first four steps are a direct derivation of the reference schema of CBR planning architecture developed by Hammond in [Hammond89]. The Figure 2 also highlights the fact that constraint reasoning techniques are continuosly exploited in all the reasoning steps a part from the retrieval.

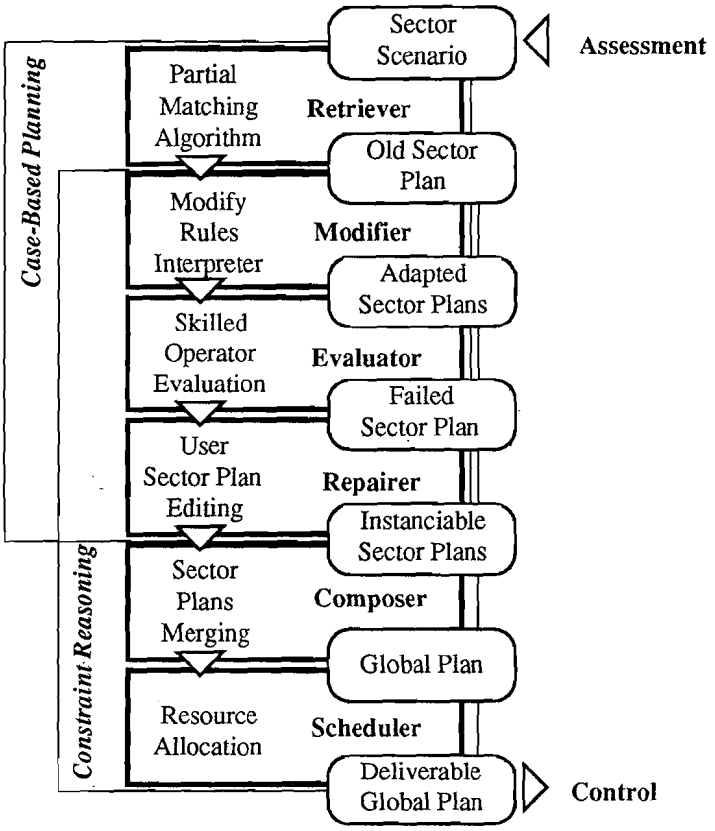


Figure 2

The Retrieval and the Adaptation of an old plan.

The fire front segmentation performed by the operator as described above produces the description of one or more sectors in terms of a scenario and the related goal. This description drives the search, into the plan memory, of similar cases where the same kind of goal in a similar scenario has been dealt with.

The search process is performed by using a partial matching algorithm which computes and compares similarities among features of the plan description. It is only possible a partial match because of the typical incompleteness of the information gathered during the alarm verification.

In fact this is a common problem in applying CBR techniques. This problem is taken into account in some CBR systems, among these we are analysing the COBWEB system [Fisher87] which seems the most appropriate to our application. COBWEB is based on an category utility function that identifies a trade-off between intra-class similarity and inter-class dissimilarity of cases. Intra-class similarity is the probability that two cases in the same category share their values and inter-class dissimilarity is the the probability that two cases in different category share their values.

But, since some features are more relevant then others, this kind of match does not guarantee that the plan associated to the most similar case is the most appropriate to the current situation. This calls for an enhancement of this method introducing a supplementary technique in order to take into account also the partial order relation among the features that describe the scenario.

The retrieval process on each sector produces a set of old sector plans. A further selection phase refines this set yielding a subset of plans that implement structurally different fire fighting strategies. Two strategies are structurally different if they don't use the same actions or if they order the actions differently.

According to the reference case-based planning architecture a modification phase follows the retrieval. A specific set of constraint, based on the domain knowledge, are associated to the features describing the old plan actions and the new current scenario. These constraints represents for instance how the number of squads, the type of action (role) they perform and the sector length define a constraint on the action duration. Adapting an old sector plan means to change some action parameter values according to the features of the current scenario and then to recompute the constraints involving them. At this level constraints are mainly used for deducing consistent values for constrained parameters.
Constraint propagation and consistency check guarantee the respect of the goal statements (for example deadline and water quantity).

Usually in the full automated case-based planner the plan validation is performed by simulation. It is out of the scope of our work to address the complex problem of simulating actions and environment evolution in the forest fire domain (see [CGHH89] for a significant work in this direction). The expert, on the base of his experience and the data on the current situation, is generally able to repair the plan. He can perform changes into the plan structure using a sector plan editor. For example substituting an action with a different one or changing the temporal relation between two actions. Repairer is supported by the system running constraint propagation algorithms on the modified action net. These two last steps, evaluator and repairer, can be repeated till a satisfiable sector plan is obtained.

The composition of a global plan and its resource scheduling.

Sector plans are merged into a global plan during the composition phase. At this level the operator can decide to consider only the highest risk sectors among the originally planned sectors, in order to avoid resource allocation failure. This merging process corresponds to the composition of the single sector action nets into a global, time consistent, net.

The scheduling phase deals with the selection and allocation of the resources to the resulting global plan. This process starts by considering the action time net of the global plan from the point of view of the resource requests associated to each action. In other words the action time net is mapped to the corresponding resource time net where the variables are the resources required by each action. The temporal constraints representing action durations in the action net are mapped to constraints on the duration of the allocation status of the associated resources. Additional constraints representing resource characteristics (such as shifts or maximum operating periods) will enrich the resource time net. Appropriate resource schedules for the global plan are solutions of this constraint network which will be obtained by running Constraint Satisfaction Problem solution algorithms driven by heuristic criteria on resource selection (see for instance [Fox87] [VanHentenryck92]). Possible failures are taken into account and could require to consider a different global plan (i.e. redo the composition step) or require the user to make extra resource available for instance releasing them from different fires or requesting them to the Regional Centre.[2]
The resulting resource schedule will be represented by a resource time map where the allocation period of each resource instance is recorded as well as their relative dependencies (for instance between squad n.1 and tanktruck n.3 which will support its activity). The resource time map will support the plan control activities which could require to consult and update it.
At this stage also the CBR process closes storing all the adaped sector plans which have been composed into the succesfully scheduled global plan.

[2] A precise definition of the failure that will be dealt is still object of discussion.

Bibliography

[Bareiss-King89] Ray Bareiss, JamesA. King. *Similarity Assessment in Case-Based Reasoning.*, in Proceedings of Case-Based Reasoning Workshop. Florida, 1989.

[CGHH89] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe. *Trial by fire: Understanding the design requirements for agents in complex environments.*, in *The AI Magazine*, 10(3):32--48, 1989.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. *Temporal Constraint Networks.*, in *Artificial Intelligence J.*.49, 1991.

[Fisher87] Douglas H. Fisher. *Knowledge Acquisition Via Incremental Conceptual Clustering.*, in Machine Learning 2, 1987.

[Fox87] M.S. Fox, *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Morgan Kaufmann Publishers, Inc., 1987.

[Hammond 89] Kristian J. Hammond. *Case-Based Planning: viewing planning as a memory task.* Academic Press, 1989.

[Kolodner89] Janet L. Kolodner. *Judging Which is the "Best" Case for a Case-Based Reasoner.* , in Proceedings of Case-Based Reasoning Workshop. Florida, 1989.

[Kourtz87] Kourtz, Peter, *Expert System Dispatch of Forest Fire Control Resources*, in *AI Applications*, vol 19, n. 1, 1-8, 1987.

[vanBeek92] Peter van Beek. *Reasoning about qualitative temporal information.* in *Artificial Intelligence J.*. 58, 297-326, 1992.

[VanHentenryck92] P. Van Hentenryck, *Constraint satisfaction using constraint logic programming*, in *Artificial Intelligence J.* 58: 113-159, 1992.

Our Perspective on Using CBR in Design Problem Solving *

Shirin Bakhtari, Brigitte Bartsch-Spörl

BSR Consulting GmbH
Wirtstraße 38
81539 München
Germany

1 Introduction

Due to the open-ended nature of all synthesis problems, a design problem can appear in different forms. There is a general acceptance that design problems can be classified as being *routine design, innovative design* or *creative design*. These predicates, routine, innovative and creative can be applied either to the process of problem solving or to the final product, or to both.

In [Brown, Chandrasekaran 89], the proposed classification criteria are oriented towards the design process and the primary focus is set on the distinction between routine design processes and non-routine design processes. In [Rosenman, Gero 93], on the other side, design problems are categorized according to the kind of solution obtained and the authors' main interest is to distinguish between creative and non-creative design by an examination of the result of a design process.

Routine design is generally characterized by the presumption that all design functions (goals and requirements) as well as all available structures of the artefact are fully specified and given beforehand. The process required for problem solving is one of mapping functions to structures. In an innovative design process, one may get confronted with functions or structures not known a priori. Innovative design is therefore accepted as being a process of going beyond the known structures and functionality and leading to a product which extends the set of known solutions.

Creative design, as defined in [Rosenman, Gero 93], "*incorporates innovative design but involves the creation of products that have little obvious relationships to existing products*". Creative designs can be achieved, as proposed in [Rosenman, Gero 93], by applying one of the following methods: *combination* of existing designs to a new one, *mutation* of an existing design, drawing an *analogy* and designing from *first principles* using building blocks.

If we accept the above characterization of creative design, then the use of CBR in design problem solving, as in e.g. [Domeshek 92] [Domeshek, Kolodner 92] [Hinrichs 92], would often lead to creative design as a product because CBR uses two methods classified as leading to creative designs: the combination of existing designs to a new one and the adaptation (mutation) of an old solution to get a proper solution for the problem at hand. We will return to this topic later.

Our objectives for the discussion of CBR in design problem solving in this paper are as follows. First we will describe the use of CBR for the development of a building-design support system. We discuss our point of view of categorization of design tasks as being innovative or creative and circumscribe the role of CBR in the context of this classification of design tasks. Finally we will show with an example from our application domain the necessity to use both routine and innovative CBR together with other problem solving methods in an environment which is open for user interactions that can bring in the creative solutions we cannot produce by software.

* This work was partially supported by the German Ministry for Research and Technology (BMFT) within the joint project FABEL under contract no. 01 IW 104. Project partners in FABEL are German National Research Center of Computer Science (GMD) Sankt Augustin, BSR Consulting GmbH München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

2 Characterization of the Problem Type

In the project FABEL [FABEL 92], we are concerned with a building design problem which primarily focusses on the support of mechanical constructions using a method called ARMILLA [Haller 92]. ARMILLA includes design guidelines as well as template models for designing mechanical subsystems. Building-design integrates architectural as well as mechanical and technical design aspects, e.g. air conditioning, lighting, etc.*

The design process is strongly affected by different types of knowledge and building-design problem solving is characterized by a thriving interaction between these types of knowledge, e.g. functional knowledge, topological knowledge and geometrical knowledge. Some knowledge occurs in the form of former layouts, other knowledge in the form of design patterns for designing mechanical subsystems. In addition, there are guidelines, standards, heuristics etc.

The question of how to find an adequate knowledge representation has to deal with good comprehensibility, an appropriate formal representation and the efficiency of the problem solving methods needed. In order to represent the knowledge adequate, we kept close to the *natural manner* of the types of knowledge and represent them in the form of cases, template models, heuristics, constraints, etc. Representing some knowledge in the form of cases seems to be an appriopriate and efficient way of knowledge representation, although it is not the only one.

In fact, human experts use variations of former layouts, but mostly as a source of inspiration and they adapt former layouts rather partially than completely. To design a new building does certainly not require new discoveries about statics or geometrics or to circumvent standards and constraints being established. But the building-design problem solving process continues to confront the architects and the technical engineers with problems that never occured before in exactly the same constellation and therefore need creative ideas to solve the particular problem within the given context.

As we got more involved in the development of the building-design support system, the following question raised up: Do we need a problem solving method using CBR to support routine design, innovative design or even creative design? Using CBR is actually a conservative way of reasoning, because it mainly deals with problems solved at least once before. But the adaptation needed to make a former solution applicable can lead to an innovative solution and therefore right into innovative design. In contrast to this, in [Hua et al. 92] adaptation is seen as follows: "*Adaptation of cases are mostly used to generate routine design*".

In our opinion, we should make a difference between *adjusting* and *adapting* an old solution to the problem at hand. Adjustment certainly leads to a routine design as solution, because it deals with marginal modifications and is mainly restricted to the change of the coordinates of former solutions. One important aspect of supporting building-design problem solving deals with the routine type of design problems, due to standards and template models and some established constraints. Standards and established constraints must be taken into account while designing the mechanical subsystems and the template models should also be considered.

Routine design can thus be obtained by applying standard AI-methods like rule-based reasoning or constraint satisfaction or - as an alternative - by applying case-based reasoning with adjustment, but without major adaptation. Adaptation, as we understand it, modifies an old solution in an essential way. This has to be done on the basis of non-trivial domain knowledge and adaptation procedures. One important aspect of adaptation is that it always leaves behind a certain risk that the adapted solution might not be acceptable with respect to the context of the building or the architect's assessment.

For example, applying ARMILLA-guidelines leads to a significant adaptation of the former design and produces a new design. The obtained solutions can be regarded as a mutation of former layouts due to the available knowledge. According to the above characterization we obtain a novel, innovative design.

Another method of achieving innovative design is, as mentioned above, the combination of cases [Faltings 91]. There is a distinction drawn in [Hua et al. 92] between crossing cases and composing cases: "*In crossing, we combine properties of cases, while in composition, we assemble pieces of structures represented in cases*".

* A CAD system for ARMILLA which is used in the project FABEL as a first prototype is called A4 [Hovestadt 93].

If we want to cross or compose several cases, we need compatibility knowledge, which may be acquired and represented in some kind of knowledge representation schemes. Applying this kind of knowledge in order to get a correct new design can be seen as obtaining an innovative design.

As we are concerned with developing a building-design support system, the flexibility of human expert behaviour has to be supported by the adaptability of the problem solving process. This means the ability to change between different kinds of knowledge and problem solving approaches in order to solve the problem at hand. Designing buildings is therefore an integrated cooperative activity between architects and engineers aimed at creating a new design.

3 The Application of Case-based Reasoning and other Problem Solvers

We have outlined above that our application can be seen as a conglomeration of different subproblems, e.g. building construction, duct layout, lightening cabel layout, etc., which range from routine to innovative design. Designing buildings is a holistic process and control is managed interactively. Therefore, all solutions of subproblems, e.g. duct layout, are taken preliminary as long as there are other unsolved (sub-)problems whose solvability or whose solutions may affect not only the local neighbourhood but also principal design decisions made in the early stages of the planning process. Therefore all (sub-)problem solving is preliminary and subject to change if new requirements arise.*

For the subproblems involved which can be classified as routine design problems, we have some freedom to decide which method of problem solving we want to use. Former prototypes of ARMILLA-based design problem solvers [Drach 93] used different rule-oriented AI-approaches. These approaches led to some progress concerning small subproblems but became much too complex for larger contexts. From these experiments we know that we can use rule packages or constraint problem solvers to obtain solutions for small and completely specified subproblems, namely routine designs.

Another choice for this kind of subproblem solving is case-based reasoning. In the FABEL project we have done some experiments with different similarity functions to test the case-based problem solving approach for cases of different complexity [Coulon et. al. 93]. The results show that case-based reasoning is an applicable alternative to the established AI methods. The speed of problem solving and the quality of the results heavily depend on the contents of the case library. Therefore the two types of problem solving approaches are difficult to compare.

What we learn from this mixed approach is that 'routine'now has two meanings:

- being a known type of problem which can be completely modeled, de- and recomposed and not being too large to be solved by a strong AI method or
- having predecessor problems which have lead to entries in the case library and which are retrievable through a similarity function and can be adjusted to the problem at hand.

For those subproblems which can be classified as innovative design problems, our classical AI methods are not so easily applicable. Here the CBR approach is at the moment our best chance to solve these problems by software. As innovative solutions range over a wider spectrum than routine solutions, it is less probable that the case library already contains a solution that fits exactly and adaptation becomes more and more important.

An essential source of adaptation knowledge are design-guidelines which help both to structure and to reduce the amount of possible transformations. In the next section we give an example of how a design process can proceed in rather little steps using different kinds of knowledge and specialized little problem solvers which have the form of completion and transformation functions for architectural designs.

* The problem addressed here is that of the recomposition of subproblems to a final solution. In our opinion we cannot avoid this problem by using CBR in hardly decomposible application domains such as ours, as it is stated in [Kolodner 92]. This problem is currently one of our ongoing research subjects.

We classify all our transformation steps as being routine or innovative steps for the following reasons:

- We do not agree with [Rosenman, Gero 93] that the usage of combination, mutation or anology is sufficient to lead to a creative design. Instead we classify our design results from combination and mutation as being innovative. We do not use analogy so far but even if we would draw analogies between different aspects it would remain innovative from our point of view.

- What we consider to be creative design is beyond all our software problem solving abilities. To bring in creative solutions is a role which remains with the architect. This distribution of problem solving capabilities is highly acceptable for the architects we work with. They would have had serious problems with giving away the 'inventory phases' to a software system.

The more adaptation is needed the higher rises the risk that the produced solution causes conflicts with the surrounding context or other preliminary solutions already achieved. Therefore we have decided to develop so called *quality-assessment functions* [Silverman 92] which check the coherence of the embedding of new solutions into the designed environment. These assessment functions are a good example of how the degree of automated problem solving decreases when the need of innovation or even creativity to solve the problem at hand increases. This form of openness has two main advantages:

- the system does not prevent the architect from being really creative and

- the system is able to catch the creative solution found by the architect, to store it in the case library and to use it as an innovative solution the next time it is applicable.

This gives both the architect and the software system a chance to improve their performance.

4 An Example

The following example illustrates a design process where a rather abstract sketch is subsequently refined and completed by a cooperative interaction between different problem solving paradigms. It shows how a cooperative interaction between the agents involved in the design process can lead to a more informative and useful stage of a design arrangement at hand. The design process can proceed, even if not all agents are able to deal with the design arrangement and if no accurate problem description is given.

The architect can for example draw three columns and call the system for assistance. Please note that the arrangement given is no proper problem description and there is no concrete goal articulated.

1-2: Applying a design guideline leads to the following: Every column in a building has to be connected by beams. According to the distance given in figure 1, the columns get connected as shown in figure 2.

2-3: As there are no standard beams of the length given (figure 2), the beams have to get adjusted. The length of the beams becomes standardized as shown in figure 3.

3-4: At this stage the design process can be continued by CBR. The most similar former layout found in the case library is shown in figure 4.
Our similarity function works with abstract graph representations of the layouts. Similarity between the new case and an old case means that the old case is the smallest graph found in the case library which subsumes the new one completely. The graph matching is done on the basis of an application specific normal form which is obtained by abstracting away deviations from a MIDI-specific grid, absolute coordinates and effects from rotation or reflection.

4-5: Now the design guideline from 1-2 is applied again and the stand alone column is connected properly (figure 5).

5-6: The standards applied change the plan to figure 6. The reason for this is that the kind of connecting columns in figure 5 would not stand the weight on it. There have to be added two more columns, otherwise the whole construction built on this arrangement would be in danger to collapse.

6-7: Proceeding the design process with applying CBR a second time leads to a new arrangement, shown in figure 7, which is again symmetric.

7-8: One of the design guidelines gives the advice: if ever possible, make your design regular.

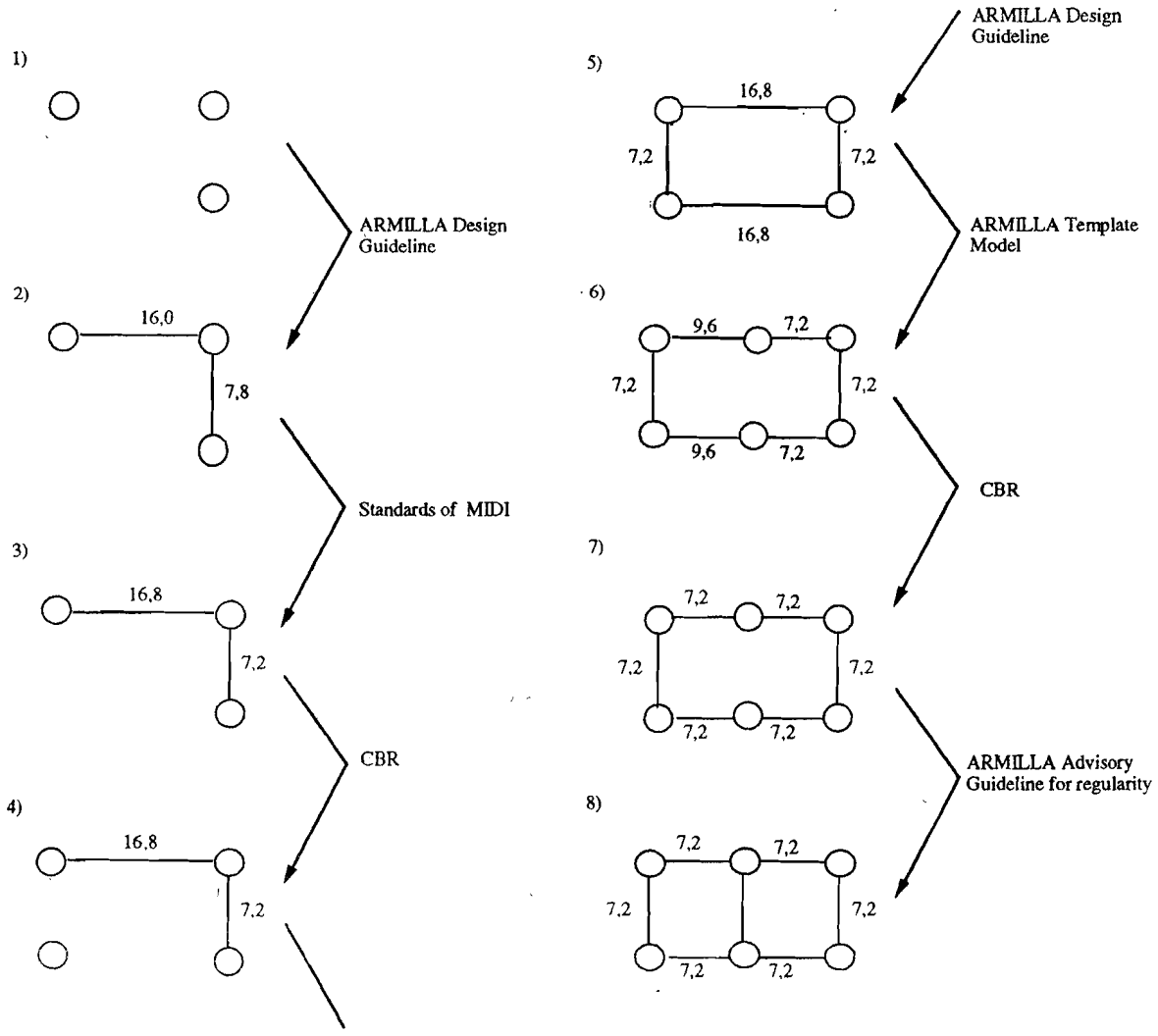


Fig. 1. Example for a design process

5 Outlook

In the above example, we focussed on a single-stream design task, although building-design is an integrative discipline covering several design processes concurrently. This has the consequence that in the future we will have to deal with interactions and mutual influences between concurrent design processes.

Acknowledgement

We thank Ludger Hovestadt for his cooperation and his contribution according to the example.

References

Brown, D.C.; Chandrasekaran, B.: *Design Problem Solving: Knowledge Structures and Control Strategies.* Pitman, London 1989.

Coulon, C.H. (ed.).: *Ähnlichkeitsansätze*. FABEL-Report No. 13. To appear. GMD St. Augustin 1993.

Domeshek, E.A.: *Using Cases for Design Aiding*. International Conference on Artificial Intelligence in Design. Carnegie Mellon University, Pittsburg, USA 1992.

Domeshek, E.A.; Kolodner, J.L.: *Toward a Case-Based Aid for Conceptual Design.* International Journal of Expert Systems.

Drach, A.: *Flexible Werkzeuge für die Integrierte Gebäudeplanung.* Dissertation. Institut für industrielle Bauproduktion. Universität Karlsruhe 1993.

FABEL-Consortium: *Survey of FABEL.* FABEL-Report No.2. GMD St. Augustin 1992.

Faltings, B.: *Case-based Representation of Architectural Design Knowledge.* DARPA Case-based Reasoning Workshop 1991.

Haller, F.: *Integrale Planung - Seminarbericht.* Fakultät für Architektur, Universität Karlsruhe 1992.

Hovestadt, L.: *A4 - Digitales Bauen*. Dissertation. Fakultät für Architektur, Universität Karlsruhe 1993.

Hinrichs, T.R.: *Using Case-Based Reasoning for Plausible Design Tasks.* Second International Conference on Artificial Intelligence in Design. Carnegie Mellon University, Pittsburg, USA 1992.

Hua, K.; Schmitt, G.; Faltings, B.: *What Can Case-based Design do?* Second International Conference on Artificial Intelligence in Design. Carnegie Mellon University, Pittsburg, USA 1992.

Kolodner, J.: *CBR for Design.* Second International Conference on Artificial Intelligence in Design. Carnegie Mellon University, Pittsburg, USA 1992.

Rosenman, M.A.; Gero, J.S.: *Creativity in Design Using a Design Prototype Approach.* In: Gero, J.S.; Maher, M.L.(eds.): Modelling Creativity and Knowledge-Based Creative Design. Lawrence Erlbaum, Hillsdale, New Jersey 1993.

Silverman, B.G.: *Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers*. Communications of the ACM. Vol. 35, No. 4 1992.

Integrated Case-Based Building Design

Kefeng Hua, Ian Smith and Boi Faltings

Artificial Intelligence Laboratory
Swiss Federal Institute of Technology
1015 Lausanne, Switzerland

Summary

A building design problem can be viewed according to many different abstractions. For example, an architect views a building as a collection of spaces with particular properties, while a civil engineer might consider it as a structure made up of load-bearing elements. For design, it is important to combine these different viewpoints into a single coherent object. Difficulties associated with combining viewpoints lead to what is termed the *integration problem.* Case-based design (CBD) is a recently developed technique for knowledge-based design systems. This paper shows how the technique can help solve the integration problem. Cases of previous design solutions already provide solutions for the integration of several abstractions into a single object. When novel designs are created by *adapting* cases, integrity can be maintained through careful formulation of the adaptation procedures. We describe a prototype design system, CADRE, which applies CBD to several examples of building design.

1 Introduction

Integration is one of the central issues for large scale engineering design problems. For example, in building design many design abstractions are considered in order to define an artifact that satisfies multiple functions. The integration problem in the domain of building design has been traditionally solved with a blackboard control structure. IBDE [18] and ICADS[14] are examples of such systems. Blackboard control helps reduce the search space in these systems. However, it might result in loops, or even diverge. Blackboard architectures are still used in large scale engineering systems because engineering design is too complex to be processed in a single model. Recently developed case-based reasoning(CBR) technology[17, 12] provides a natural solution to the integration problem, since cases themselves are integrated solutions to particular design contexts. Creative use of design cases through adaptation and combination may provide solutions for new design specifications. We have developed a prototype design assistant called *CADRE.* In this system, the processes and representations are divided into two levels: *dimensional* and *topological.* In this paper, adaptation based on *dimensional models* of design is studied for integration of building design including structure, spaces and circulation pattern.

Adaptation of a design case was addressed in Julia[9], a system that interactively designs the presentation and menu of a meal. The system implicitly achieves integration by satisfying multiple, interacting constraints. In our research, we are studying a more complex domain: building design. A building design involves thousands of parameters, even for a simple house. Building design is an ill-structured problem where domain theories can not be formalized. Building design requires integration of different abstractions which are in entirely different domains. In building design, shape and geometry are important. Building design poses a challenge to the CBR technique, conceptually and computationally.

Our contribution lies in using case-based reasoning technology to solve large scale engineering design problems where geometry and dimensional information are important. This paper describes our research on integration of building design through case-based reasoning. We achieve integration most effectively at the dimensional level of designs.

2 Integration

Any physical artifact can be viewed according to many different abstractions. For example, a building can be:

- an ingenious civil engineering structure of beams and columns.
- a magnificent way of creating architectural spaces.
- a practical arrangement of functions for its occupants.

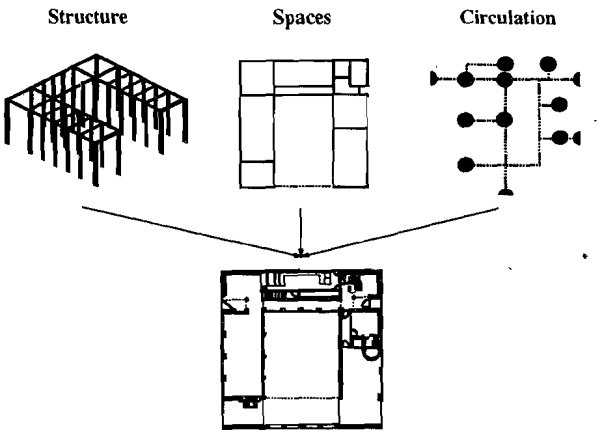


Fig. 1. *A building represents an integration of many different abstractions, including structure, spaces and circulation pattern.*

Designing a building is difficult because it has to *integrate* satisfactory solutions in each abstraction: the structure designed by the civil engineer, the spaces laid out by the architect, and the circulation pattern desired by the user are part of one single structure (Fig. 1).

Disagreements and misunderstandings between architects and civil engineers are recognized as sources of many problems in construction[1]. Producing and documenting designs on a CAD system, preferably an intelligent CAD system, help detect problems during the design phase by checking consistency between the designs produced by different people. Research efforts such as IBDE [18] have already proposed computer tools for integrating designs generated in different abstractions.

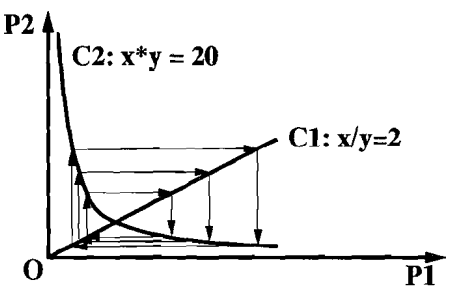


Fig. 2. *P1/C1 and P2/C2 represent parameters and constraints in structural and architectural abstractions. When discrepancies in each abstraction are corrected in isolation, the process may go into cycles as indicated by the arrows.*

In IBDE, seven different modules correspond to different abstractions and communicate via a common data representation called a blackboard. Inconsistencies are detected by critics and cause reactivation of certain modules in order to eliminate the problem. Since corrections are constructed locally, this process may well cycle or even diverge. For example, in Figure 2, constraint C1 is a constraint in structure abstraction which specifies that the length of beam x in a rectangular space is two times of the length of beam y in the space. The two beams span directions of length and width in the space. C2 is an architectural constraint specifying that the area of the space is 20. P1 and P2 are two parameters corresponding to the length and width of this space. Supposing that the starting point is $(20, 10)$ which is on line $x = 2y$, but not on curve $x * y = 20$, the module for the architectural abstraction may revise values to $(20, 1)$ by adjusting parameter P2 to satisfy constraint C2. Since $(20, 1)$ does not satisfy constraint C1, blackboard control shifts to the structural abstraction and adjusts parameter P1 to satisfy constraint C1 by moving to point $(2, 1)$. This process cycles. In general, correcting the discrepancies by locally adjusting either P1 to satisfy C1 or P2 to fall onto C2 leads to a cycle which may not converge to the solution. Only through simultaneous consideration of all abstractions can such problems be avoided.

Achieving integration in a classical knowledge-based system framework is in principle possible, but extremely difficult because there are few general principles which hold over all abstractions. Attempts to

[1] Disagreements involving occupants are probably even more frequent, but rarely communicated to the designers.

formulate knowledge in an integrated way exist. For example, Alexander [1] has produced a handbook which defines principles of good design that consider several abstractions simultaneously. A striking fact about his work is that the rules he defines are actually prescriptions for particular buildings in particular environments, with little generality. The lesson from this observation is:

> *Integrating design knowledge from many abstractions amounts to formulating particular cases of good design.*

This observation leads to the formulation of design knowledge as *prototypes* [5] which are generalized versions of particular structures. However, since prototypes still require tedious formulations of the generalizations that apply, design by reusing previous *cases* is of interest. In this paper, we show how this paradigm of *case-based reasoning* can be applied to solve the integration problem in building design.

3 Adaptation

Case-based reasoning originates from psychological models of human memory structure [17, 12]. A case-based problem solver consists of mainly two processes: *indexing* to find a suitable precedent, and *adaptation* to use it in the new problem context. For case-based *design*, adaptation is essential; no two design problems are ever identical. Since indexing can be carried out by user interaction and since indexing schemes may depend upon characteristics of adaptation strategies, we have focused our research on the adaptation of cases to new problems.

Design cases Design requires knowledge in order to synthesize structures. For building design problems of realistic size, formulating such synthesis knowledge is very tedious, since conflicting goals lead to many tradeoffs. This knowledge is more easily accessible in the form of cases of existing buildings, and each case incorporates a large amount of synthesis knowledge. A case defines a set of "good" ways of achieving functions in different abstractions, and a way to integrate them into a single building.

A case-based design system can be characterized by its dependence on cases as the main knowledge source. Following discussions in the literature [8, 2, 15] regarding the distinction between "surface" and "deep" features and their relative merits in indexing cases, we distinguish two kinds of cases: *shallow* and *deep*. A *shallow* case is a model of an existing building without any further information about how it was obtained. In contrast, a *deep* case is augmented by a trace of the process which devised the design. Since such design trace can not be easily acquired in engineering design, we attempt to limit our research to cases which are as shallow as possible in order to test how far this approach is applicable.

A shallow case defines an actual artifact, represented for example as a CAD model of the actual building. In our implementation, we use AutoCAD as a tool for representing and rendering this information plus a set of basic constraints(*first principle constraints*) to translate the AutoCAD model into shallow cases.

A good building design is an example of successful integration of functions from different abstractions. These functions are modeled by a symbolic vocabulary appropriate to the corresponding abstraction, and mapped to *constraints* formulated on the common CAD model. The CAD model thus serves as a basis for integrating different abstractions.

Case adaptation Applying a case to a new problem requires changing the structure while maintaining the integration of the abstractions that has been achieved in the case. We divide the process into two layers: topological and dimensional. Dimensional adaptations are changes in geometry that do not involve the removal or addition of elements and spaces. In its simplest form, dimensional adaptation reduces to scalings. Topological adaptations are changes which involve a modification of space or element topology. Dimensional integration can be achieved in a domain independent way. Topological integration requires explicit design functions and constraints, which means extra design knowledge in addition to the case.

Integration of multiple abstractions of design can be achieved during dimensional adaptation. In case-based design, dimensional constraints from a shallow case and a specification for a new design problem at dimensional level give a set of dimensional constraints. Constraints include equalities and inequalities, and constraints can be linear or non-linear. Therefore, an integration of multiple abstractions corresponds to a simultaneous solution to these constraints. Our methodology for integration of design at dimensional level can be described as follows:

- Parameterize the case and new design specifications with first principle knowledge for each design abstraction.
- Perform dimensionality analysis with equalities.
- If the equalities are over-constrained, either resolve the conflict by *dimensionality expansion* to introduce new design dimensions or relax constraints.

- If the equalities are under-constrained, use the process of *dimensionality reduction* to define the variability of the case under new design specification with the free variables.
- Propagate the adaptation among the inequality constraints in the dimensionality-reduced space to check its validity.
- If the adaptation is not valid, go back to a new parameterization.

With these steps, integration of design at dimensional level can be achieved.

Parameterization Parameterization is achieved through the base-parameterization of the CAD model of the case plus constraints in first principles and constraints interactively posted by users.

Dimensionality analysis Dimensionality analysis is the process that decides if a given constraint system is under-constrained, over-constrained or has exactly one solution.

Dimensionality Expansion Dimensionality expansion is used to introduce new degrees of freedom. Constraints should not be dropped when they are in conflict since they represent certain design requirements. The method of dimensionality expansion was originally used in 1stPRINCIPLE, a program that does creative mechanical design through monotonic analysis of design parameters to the object function [3]. We use the method to solve conflicts by introducing new design parameters, for example, to free some of the parameters that are originally fixed as constants in base parameterization. Dimensionality expansion sometimes implies the modification of the structure of the case by introducing new elements. Each step of dimensionality expansion will give at least one more degree of freedom. Dimensionality expansion provides a link between dimensional and topological adaptations.

Dimensionality Reduction The concept of dimensionality reduction is adopted from the recoding method in the reduction of dimensionality of multivariate data in statistics [13]. This idea was developed further by Saund [16] in image recognition.

The concept of dimensionality reduction was first introduced to case-based design by Faltings [6]. In integrated case-based design, this method is used to simplify dimensional adaptation of case by finding the exact degrees of freedom that can be changed for the case in a given new situation and by defining all the other design variables in terms of a small set of adaptation parameters.

Inequalities Dimensionality reduction only applies to equalities. Among inequalities, we can distinguish two types: *critical* inequalities which are limitations exploited to the maximum and just satisfied in the case, and *non-critical* ones which are satisfied by a large margin. If the case is sufficiently close to the new solution, critical-constraint sets can be assumed to remain the same in spite of the adaptation. Thus, critical inequalities can be replaced by equalities to which dimensionality reduction applies. Non-critical inequalities are constraints on new parameter values and are handled by a constraint propagation mechanism.

Topological changes In our system, when adaptation at dimensional level fails, topological adaptation of the case starts. For topological changes, we have not yet succeeded in defining an analog to dimensionality reduction; in fact, such an analog may not exist. Thus, we cannot ensure that integration is maintained throughout a topological modification. However, case adaptation still offers advantages over generation: if the case is sufficiently close to a feasible solution, the number of topological changes that are required, and may destroy the integration, is much smaller than what would be entailed by generating the building from scratch.

Topological changes require explicit functionalities and associated constraints for all the abstractions of the case. Generalized design knowledge for each aspect of design and knowledge about tradeoffs among the design abstractions should be used to achieve integration of design at the topological level. Integration during case adaptation at the topological level can be formulated as a dynamic constraint satisfaction problem(DCSP). Topological changes are triggered by dimensionality expansion. At the topological level, relations and functional attributes are represented by constraints with conditions for each design abstraction. Integration of all the abstractions after modification at topological level requires combination of constraint networks. When topological adaptation is performed in one abstraction, integrity of the case can be destroyed. Knowledge about integration will be required to re-establish integration of all the abstractions. Such knowledge can come from user interaction during the adaptation as with the CADRE system.

Computational models for DCSP in case-based design can be realized by a combination of traditional constraint satisfaction algorithms plus truth maintenance systems. In topological adaptation, when a physical constraint should be retracted, a new design case will be considered. DCSP is not suitable for dimensional adaptations. In our approach, we use DCSP for topological adaptation (symbolic computation) and dimensionality reduction/expansion (numerical computation) for dimensional adaptation.

4 *CADRE*, a prototype design system

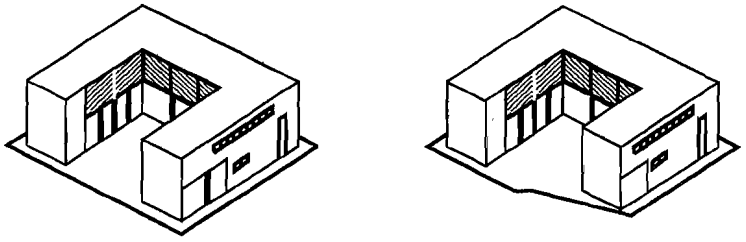


Fig. 3. *Example of case adaptation*

In order to explore the adaptation of cases in design, we have implemented a CAse-based spatial Design REasoning system(CADRE) [6, 10, 11]. One example treated by CADRE is shown in Figure 3. It is a U-shaped building (the Felder house in Lugano, Switzerland, [4]) adapted to a slightly different site. CADRE modified both the dimensions and the topology of the case in order to obtain a solution that preserves the functionalities and tradeoffs in the case.

Computationally, the processes in CADRE are divided into two layers: a symbolic layer and a numerical layer. They correspond to the topological and dimensional models of the case. CADRE focuses on case adaptation, leaving case selection to the user. The adaptation is conducted with the following procedure:

1. Evaluation of the existing case in the original and new environments in order to find discrepancies. Insertion of the case into the new design context so that a maximum coincidence is achieved, subject to constraints posted by the user. In the example of Figure 3, opening of U was placed on the ragged edge of the new lot.
2. If there are dimensional discrepancies, identify the violated constraints and the parameters which are involved in them. Complete the set of applicable parameters and constraints with all those which are related to the original ones through links in the constraint network. This defines the complete *base set* of parameters and constraints related to the discrepancies. In the example of Figure 3, these parameters and constraints are located in the right wing.
3. Apply *dimensionality reduction* to the base set of parameters and constraints to define an adaptation parameterization which is guaranteed to avoid conflicts. In the example, most parameters are represented by a few parameters that decide the change of the right wing.
4. Modify the dimensions using the parameters resulting from dimensionality reduction. Users control the process by asserting additional constraints or manually identifying suitable values.
5. Check the validity of the adaptation by verifying inequality constraints in the base set that were not critical and thus not treated by the dimensionality reduction.
6. If there is no solution at the dimensional level for the new design problem, trigger topological transformation rules which relax constraints in the related constraint set. If there is a transformation which preserves design features of the case, go back to step 1, otherwise the case is not suitable.

Tests on several real examples, along with discussions with practicing engineers and architects lead us to believe that the procedure described above is complementary to their activities.

CADRE was implemented in Common-Lisp, C and AutoCAD. It runs on Unix based workstations and could be migrated to any platforms which possess the same software environment.

5 Conclusions

We have argued that case-based reasoning offers assistance for integrating different abstractions in design. Our prototype system, CADRE, illustrates the usefulness of the approach for practically interesting designs. The paradigm of case-based design fits very well with the observation that human designers like to work by reusing cases of previous designs. The considerations we have presented in this paper may be an explanation for *why* this is the case: integration of abstractions may be the main reason why designers reuse previous cases. Adaptation of single cases is suitable for *routine* design. For innovation, we have to address the *combination* of cases; this is the topic of our current research.

Acknowledgements

This work is a result of collaborative research with CAAD(Computer-Aided Architectural Design), ETH Zürich, and ICOM(Steel Structures), EPF Lausanne. Discussions and collaboration with professor Gerhard Schmitt (CAAD) have been most valuable. We would also like to thank the collaborators Shen-Guan Shih and Simon Bailey for their work on implementation of the ideas described herein, and to whom the credit for many of the details of the work is due. We also thank the Swiss National Science Foundation for funding this research as part of the National Research Program 23 on Artificial Intelligence and Robotics.

References

1. **Alexander, C.** "Notes on the synthesis of form" Harvard University Press, Cambridge, Mass, 1964
2. **Birnbaum, L. And Collins, G.** "Remindings and engineering design themes: a case study in indexing vocabulary" Proceedings of workshop on case-based reasoning. 1989, pp47-51.
3. **Cagan, J. And Agogino, A.M.** "Dimensional variable expansion – a formal approach to innovative design" Research in engineering design, Springer-Verlag New York Inc., 1991, vol. 3, pp75-85
4. **Mario Campi - Franco Pessina** "Architects", Rizzoli International Publications, New York, 1987
5. **Balachandran, M., Gero, J.** "Role of prototypes in integrated expert systems and CAD systems" International conference on artificial intelligence in engineering, Boston, 1990
6. **Faltings, B.** "Case-based representation of architectural design knowledge" Computational Intelligence **2**, North-Holland, 1991
7. **Faltings, B. And Haroud, J. And Smith, I.** "Dynamic constraint propagation with continuous variables" Intelligent Computer Aided Design, 1991
8. **Hammond, K.J.** "On functionality motivated vocabularies: an apologia" Proceedings of workshop on case-based reasoning. 1989, pp52-56
9. **Hinrichs, T. R. and Kolodner, J. L.** "The Roles of Adaptation in Case-based Design" in: DARPA Case-based Reasoning Workshop, Butterworth, 1991, pp.121–132
10. **Hua, K., Smith, I., Faltings, B., Shi, S. And Schmitt, G.** "Adaptation of spatial design cases" in: Artificial intelligence in design'92. Kluwer Academic Publishers, 1992, pp559-575
11. **Hua, Kefeng and Faltings, Boi** "Exploring case-based design: CADRE" Artificial Intelligence for engineering design, analysis and manufacturing, **7(2)**:135-144, 1993.
12. **Kolodner, J. L.** "Retrieval And Organizational Strategies in Conceptual Memory: A Computer Model" Hillsdale, NJ: Lawrence Earlbaum Associates, 1984.
13. **Krishnaiah, P. And Kanal, L.** "Handbook on statistics" North-Holland, Amsterdam, vol. 2, 1982
14. **Myers, L., Pohl, J., and Chapman, A.** "The ICADS expert design advisor: concepts and directions" in: Artificial intelligence in design '91, J.S. Gero (ed.), Butterworth 1991. pp. 897-920
15. **Owens, C.** "Plan transformations as abstract indices." Proceedings of workshop on case-based reasoning. 1989, pp62-65.
16. **Saund, E.** "Configurations of shape primitives specified by dimensionality-reduction through energy minimization" IEEE spring symposium on physical and biological approaches to computational vision, Stanford, March 1988
17. **Schank, R.** "Reminding and memory" chapter 2 in: Dynamic memory - a theory of reminding and learning in computers and people, Cambridge University Press, 1982
18. **Schmitt, G.** "IBDE, VIKA, ARCHPLAN: architectures for design knowledge representation, acquisition and application" in H. Yoshikawa, T. Holden (Eds.): Intelligent CAD ii, North Holland, 1990

Case-Based Reasoning in Complex Design Tasks

Neil A.M. Maiden

Centre for Human-Computer Interface Design
City University
Northampton Square
London EC1V OHB
Tel: +44-71-477-8412
E-Mail: N.A.M.Maiden@city.ac.uk

Abstract. Case-based reasoning can aid complex design tasks, however a cooperative paradigm for case-based design is needed. The paper argues for computational mechanisms for retrieving design cases but cooperative assistants to aid designers to understand and adapt retrieved designs. This necessitates multi-disciplinary research in case-based reasoning.

1 Introduction

Case-based reasoning can aid complex design by retrieving and customising old designs to similar problems (e.g. [1]). It mimics the behaviour of expert designers who recall old solutions to structure and complete new designs (e.g. [2,3]). This capability to solve ill-structured problems and designs [41] distinguishes case-based reasoning from previous artificial intelligence paradigms. However, effective case-based design requires cooperation between tool and designer to maximise their reasoning capabilities and the knowledge available [4]. This paper proposes a cooperative, case-based design paradigm founded on complex analogical reasoning mechanisms for case retrieval and explanation, and cognitive models of designer's analogical reasoning to inform design of tools which aid adaptation of retrieved designs. The paper has four parts. First, arguments for cooperation during case-based design are outlined. Second, conclusions from two projects which investigated case-based design in software engineering are followed by implications for case-based design in other disciplines. Finally, future research is discussed.

2 Cooperation During Case-Based Design

Case-based reasoning has the potential to aid design practice, however many case-based reasoning tools only support diagnosis tasks such as electric circuit repair [5] and computer help desks [6,7]. Case-based design introduces two problems: (i) retrieval of complex designs from ill-defined problem statements, and (ii) adaptation of retrieved solutions to fit the new design. Design retrieval is difficult due to the ill-structured nature of design tasks, and complex designs such as office blocks or software systems cannot be retrieved by simple categorisation or faceted classification schemes (e.g. [8]). Furthermore, retrieved designs must be customised by the designer. However, empirical evidence indicates that understanding and adapting old solutions is problematic. Studies of physics [9] and mathematical [10] problem solving revealed solution copying and comprehension avoidance as a short cut to cognitively demanding tasks. Design adaptation requires good understanding, so cooperative assistance is needed to explain cases to the designer and guide the adaptation task.

This paper argues for cooperation between tool and designer throughout case-based design. Division of tasks is achieved by viewing case-based design as a complex analogical reasoning task. Analogical reasoning is defined as the transfer of knowledge from past problem solving episodes to solve new problems that share significant aspects with corresponding past experience [11], a definition which can be applied to complex case-based design tasks. Analogical reasoning has been investigated by artificial intelligence researchers [12,13] and cognitive scientists [14,15,16], therefore the strengths and weaknesses of human and tool-based analogical reasoning are known. The following task division is proposed:

- computational mechanisms for design retrieval and explanation;
- design selection and adaptation by the designer who is assisted by computational mechanisms.

Computational models of analogical reasoning exploit partial domain knowledge to retrieve complex cases and form analogical mappings with cases [17,18]. This contrasts with the complete domain knowledge needed to adapt complex designs. Computational retrieval mechanisms have several advantages over information retrieval strategies. First, computational models of analogical reasoning are robust and can retrieve analogical cases from ill-defined problems typical in complex design tasks. Second, inferred analogical mappings between designs can improve the explanation of retrieved cases. Third, analogical reasoning exploits the structure inherent in most designs to reduce dependence on domain-specific indices or keywords (e.g. [19,20]). On the other hand, designers possess greater domain knowledge and are better analogical reasoners with single cases than are design tools, although difficulties do arise. Therefore, computational mechanisms must aid case adaptation by designers rather

than adapt designs themselves. Their design must be informed by how designers understand and adapt old designs. This paper argues that effective tool design should be founded on cognitive models of analogical reasoning during design reuse. Cognitive science provides an empirical basis for these models, however the case-based reasoning community lacks evidence of difficulties which arise during these tasks. To answer some of these questions, cooperative, case-based design paradigm was introduced to aid requirement and specification of software systems.

3 Case-Based Design in Software Engineering

The SERC 'AIR' and ESPRIT 'Nature' projects have investigated retrieval and adaptation of cases, including analogical specifications and domain abstractions, to aid specification of new software systems [21]. Experienced software engineers' tend to recall mental abstractions when modelling new domains which permit them to perceive meaningful patterns in domains (e.g. [22,23]), therefore reuse of domain abstractions representing the fundamental behaviour, structure and functions of software engineering domains mimics expert design behaviour. Human [14] and computational models [24] of analogical reasoning also exploit mental and computational abstractions. This suggests the need for domain-specific models of analogical reasoning [25], in contrast to earlier domain-independent, quantitative theories. Therefore, the broad paradigm applies to all design disciplines, however the abstractions are specific to the design discipline.

Case retrieval uses a computational model of analogical reasoning to match cases which have semantically-equivalent goals, which share knowledge structures and which instantiate a domain abstraction [26], see Figure 1. Furthermore, software systems are ill-defined during specification and design, so well-articulated queries for case retrieval are unlikely. Incremental system specification is aided by retrieval of domain abstractions, similar to the approach proposed in [27]. Therefore, case-based design is in two parts. First, retrieval of domain abstractions provides feedback for problem reformulation and understanding. Second, analogical specifications are retrieved and explained to permit case adaptation and redesign.

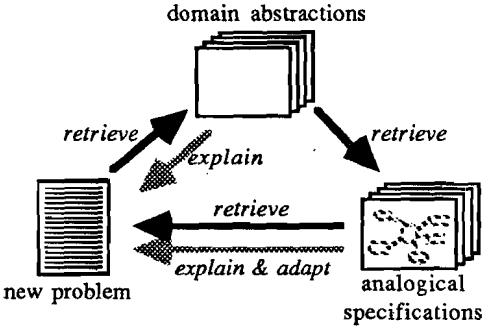


Fig 1. Overview of the case-based design paradigm for software specification

AIR, a prototype toolkit to aid system specification has been implemented and evaluated [21,28]. It consists of 6 components which is discussed in further detail to explain retrieval and customisation of analogical cases:

- the requirements capturer guides acquisition of the fundamental behaviour, structure and functional requirements of software systems as a basis for retrieving existing cases [28];
- the domain matcher is a computational model of analogical reasoning which retrieves cases and infers mappings to inform explanation [29];
- the requirements critic explains retrieved domain abstractions to encourage validation of new specifications through detection of problem situations such as incompleteness, inconsistencies, ambiguities and overspecification;
- the problem classifier reasons with retrieved cases to detect problems in new specifications;
- the specification advisor explains retrieved analogical specifications, guides their transfer to construct a new specification and promotes specification validation by cross-mapping with retrieved specifications [21];
- the dialogue controller controls interaction with the software engineer. It controls mixed-initiative dialogue permitting tool initiative to direct the design task and retrieve and explain cases when necessary.

3.1 Analogical Case Retrieval

The domain matcher [29] is a computational model of analogical reasoning which exploits partial domain knowledge to retrieve cases [17,18]. It matches a set of fundamental problem features to domain abstractions and analogical specifications. Domain abstractions are retrieved from a domain specialisation hierarchy. The domain matcher maps the new problem to a high-level domain abstraction then specialising this match to specialisations of the domain abstraction until no further specialisation is possible. Structure matching is used to establish an initial match with a high-level domain abstraction which may be specialised to lower-level domain abstractions,

therefore it is only called once during the matching process. Rule-based matching then specialises this match by mapping the new problem to lower-level domains which specialise the original domain abstraction, see Figure 2. Analogical specifications are retrieved using structure matching to the new problem and the retrieved domain abstraction. Therefore, case retrieval exploit partial domain knowledge representing fundamental domain features to retrieve domain abstractions before more complex analogical specifications.

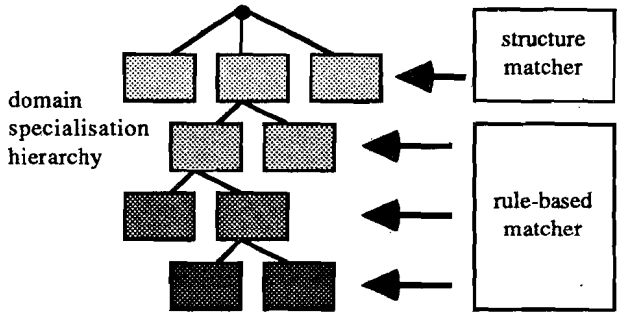


Fig 2. Analogical retrieval of domain abstractions from the domain specialisation hierarchy

Structure matching uses domain semantics and pattern matching to retrieve software specifications. First, local mappings between fundamental domain features are inferred from the theoretical model of domain abstraction. This model uses application-independent lexicons to detect semantic equivalence between domain features. Structure mapping then determines an interrelated knowledge structure possessed by both domains from these inferred local mappings. There is insufficient space in this paper to describe the matching algorithm, however an example of structure matching is shown in Figure 3. State transitions and object structures represent the fundamental behaviour and structure of software engineering domains which is used to match system specifications. Local mappings are inferred between object relations rather than attributes to allow matching between different application domains. The algorithms are defined in [29]. The rule-based matcher implements the same structure matching algorithm, however it maps a smaller number of facts in the context of an existing structure match. These isolated concepts more amenable to rule-based matching, thus improving the computational efficiency of the retrieval mechanism.

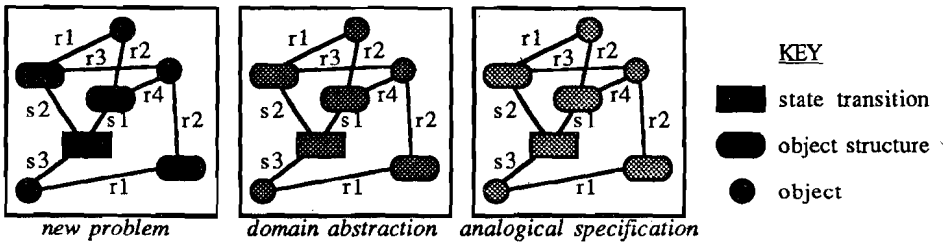


Fig 3. Graphical illustration of structured matching

The implemented architecture and algorithms of the domain matcher are described in [29]. Implementation is on a SparcStation IPX using BIMprolog as part of Nature's concept demonstrator [30].

3.2 Cooperative Case Adaptation

Cooperative tools which aid comprehension and adaptation of retrieved specifications are informed by empirical studies of specification reuse of software engineers (e.g. [31]). Analogical understanding is needed for effective adaptation of retrieved cases, and people understand analogies by forming mental abstractions representing shared features. However, understanding retrieved software specifications proved difficult, especially for less-experienced software engineers who have most to gain from case-based design:

- inexperienced software engineers exhibited mental laziness manifest as copying as a short cut to avoid understanding [31], a result also reported by [32]. Understanding complex and unfamiliar specifications proved difficult despite use of graphical notations (see Figure 4) which aid specification comprehension [21]. Indeed, these notations enabled direct understanding without understanding;
- inexperienced software engineers did not recognise mappings with domain abstractions [33]. This led to poor reuse because software engineers could not perceive similarities with cases and hence the benefits from reuse;
- problems during case understanding discouraged adaptation [33] because software engineers failed to see benefits from case-based design;
- inexperienced software engineers did not adapt retrieved domain abstractions unless structured notations were provided to enable direct transfer of the abstraction [33].

To overcome these problems, retrieved cases were explained [34] using the following strategies:

- visualisation of domain abstractions can aid understanding and hence adaptation. Gick & Holyoak [14] reported better learning of abstractions from spatial diagrams representing fundamental features during analogical problem solving. Diagrams are annotated and supported by text descriptions to aid recognition and understanding of domain abstractions [33];
- simple prototypical examples can also aid understanding of domain abstractions and more complex analogical specifications. Empirical studies revealed that abstraction induction required presentation of two or more analogical instances [14]. Indeed, people often understand new concepts using prototypical examples [35];
- domain abstractions can be animated using dynamic illustrations of domain behaviour, similar to animations of computer algorithms proposed by [36]. Studies demonstrated that animations improved learning of declarative knowledge, although it also needed text and diagrammatic explanations to consolidate learning. Domain animations are interactive and permit playback and pause facilities to aid exploration;
- tool-based guidance can aid formation of mappings between domain abstractions, analogical specifications and the new problem. Notations which guide analogical mappings include tabular definition of object mappings;
- guided exposure to cases can discourage copying and aid mental model formation, therefore these strategies are integrated during controlled explanation of cases.

Case adaptation was assisted by strategies derived from empirical studies (e.g. [21]):

- experienced software engineers used graphical notations to guide knowledge transfer and adaptation. These structured notations lead to systematic adaptation of the specification to maximise advantages and avoid omissions;
- guided exposure to cases supported this systematic transfer and adaptation by discouraging copying and drawing attention to exposed features of the specification;
- guided exposure is combined with explanation strategies to encourage incremental understanding and adaptation, following the practice of successful and experienced software engineers.

Furthermore, mixed initiative dialogue permits passive and active guidance durng case adaptation. Active guidance intervenes at the right time to explain retrieved cases and detected problems at the right level of abstraction, following the definitions in [27]. Passive guidance detects problem situations but the assistant does not intervene. Rather, detected problem situations are recorded on a problem notepad containing issues to be resolved. Problems can be browsed and acted upon by the software engineer when appropriate.

3.3 Summary

Case-based design has been shown to aid specification and high-level design of complex computer systems. Cases are retrieved using a computational model of analogical reasoning. User studies with AIR have revealed the effectiveness of retrieval of domain abstractions during problem formulation and system specification. Studies have also shown that analogical specification reuse can improve completeness of new specifications, although difficulties arise which necessitate cooperative assistance. Results have implications for case-based design in other disciplines.

4 Case-Based Design in Other Disciplines

Case-based design of children's bedrooms [37] was chosen to investigate whether these findings generalise to other design disciplines. It was chosen because of the effect of cases on design practice does not require domain experts who are difficult and expensive to acquire. This research is ongoing and its directions are two-fold.

Domain analysis of the design discipline is needed to determine fundamental domain features for case retrieval. This analysis differs from domain analysis in software engineering (e.g. [38]) because it must detect design categories (e.g. bedroom designs for different ages, budgets) and discriminating features of these categories. Knowledge acquisition from domain experts is needed to derive design categories using card sort and laddering [39] techniques. Results from this analysis identify design abstractions which can direct analogical retrieval of cases. Furthermore, structure matching between discriminating features using AIR's structure matching algorithm can enable retrieval of incomplete and inconsistent designs. Therefore, a similar computational model of analogical reasoning can retrieve bedroom designs, however the discriminating features change. This computational model is being designed.

Second, design understanding and adaptation will remain problematic due to the complexity of designs and lack of relevant domain knowledge. Comprehension avoidance and design copying is possible because designs are represented using graphical notations to define floor plans (bedroom design), architectural design (office blocks), circuit boards (electronics) and concrete stress fractures (civil engineering). An example of a bedroom floor plan is

shown in Figure 4 [37]. Complex designs which require effort to understand may not reveal the degree of similarity with the new problem can discourage design adaptation. Case-based reasoning research has paid little attention to cooperative assistance. Design tools for case-based bedroom design is assisted by explanation and adaptation strategies. These strategies were derived from reported findings during case-based software specification.

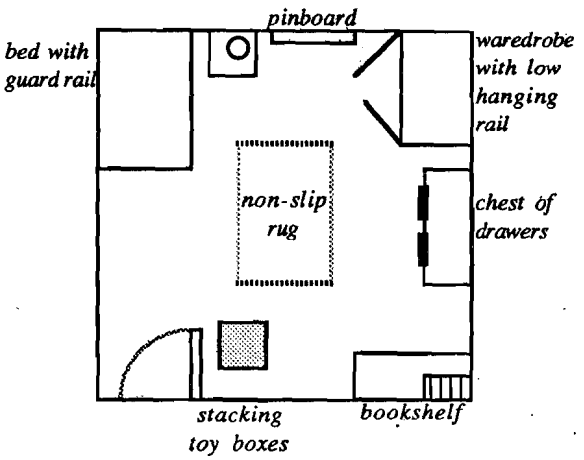


Fig 4. Floor plan of a toddler's bedroom (based on Lott 1989)

5 Future Work

Case-based design can be problematic. Therefore, computational models of analogical reasoning are needed to retrieve designs and cooperative assistance and aid designers to understand and adapt designs. Retrieval mechanisms can be informed by existing computational models of analogical reasoning [17,18]. Design of cooperative assistants for case adaptation can be informed in two ways. First, empirical studies of design understanding and adaptation can be undertaken. Studies provide domain-specific findings with implications for detailed design of assistants. Second, case understanding and adaptation can be informed by findings from other research disciplines. Cognitive studies of analogical reasoning reveal problems during similarity-based reasoning. The psychology of program comprehension and reuse has also revealed a rich seam of relevant findings [42]. The case-based reasoning community can benefit from findings in these other research disciplines.

Furthermore, cooperative assistants must be designed to support mixed initiative dialogue with designers. Design can involve complex activities including communication and negotiation. Designs are social objects in which different people find meaning. Therefore, cooperative assistants permit three-way interaction between assistant, designer and design user by using explanations to facilitate design negotiation by explaining design scenarios. Furthermore, retrieved and explained designs can provide common understanding and communication between designer and others, similar to clichés in program understanding [40]. Therefore, viewing case-based design as a cooperative process provides additional roles for design cases which warrant further investigation.

References

1. Pearce M., Goel A.K., Kolodner J.L., Zimring C., Sentosa L & Billington R.: Case-Based Design Support, IEEE Expert October, 14-20 (1992).
2. Akin O.: Psychology of Architectural Design, Pion Ltd, 1986.
3. Guindon R.: Designing the Design Process: Exploiting Opportunistic Thoughts, Human-Computer Interaction 5, 305-344 (1990)
4. Kolodner J.L.: Improving Human Decision Making through Case-based Decision Aiding, AI Magazine 12 Summer, 52-68 (1991).
5. Oehlmann R., Sleeman D. & Edwards P.: Case-Based Planning in an Exploratory Discovery System, Proceeding of Case-Based Reasoning Colloquium, IEE Digest No: 1993/036, February 1993.
6. Simoudis E.: Using Case-Based Retrieval for Customer Technical Support, IEEE Expert, October, 7-12 (1992).
7. Dearden A.M.: Interacting With a Case Memory, Proceeding of Case-Based Reasoning Colloquium, IEE Digest No: 1993/036, February 1993.
8. Prieto-Diaz R. & Freeman P.: Classifying Software for Reusability, IEEE Software, January, 6-16 (1987).
9. Chi M.T.H., Bassok M., Lewis M.W., Reimann P. & Glaser R.: Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems, Cognitive Science 13, 145-182 (1989).
10. Novick L.R.: Analogical Transfer, Problem Similarity, and Expertise, Journal of Experimental Psychology: Learning, Memory and Cognition 14(3), 510-520 (1988).

11. Carbonell J.C.: Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, Technical Report CMU-CS-85-115, Computer Science Department, Carnegie-Mellon University, March 1985.
12. Kedar-Cabelli S.: Towards a Computational Model of Purpose-directed Analogy, Analogica, Pitman (London) 1988, 89-105
13. Hall R.P.: Computational Approaches to Analogical Reasoning: A Comparative Analysis, Artificial Intelligence 39, 39-120 (1989).
14. Gick M.L. & Holyoak K.J.: Schema Induction and Analogical Transfer, Cognitive Psychology 15, 1-38 (1983).
15. Ross B.H.: This is Like That: The Use of Earlier Problems and the Separation of Similarity Effects, Journal of Experimental Psychology: Learning, Memory and Cognition 13(4), 629-639 (1987).
16. Novick L.R. & Holyoak K.J.: Mathematical Problem Solving by Analogy, Journal of Experimental Psychology: Learning, Memory, and Cognition 17(3), 398-415 (1991).
17. Falkenhainer B., Forbus K.D. & Gentner D.: The Structure-Mapping Engine: Algorithm and Examples, Artificial Intelligence 41, 1-63 (1989).
18. Holyoak K.J. & Thagard P.: Analogical Mapping by Constraint Satisfaction, Cognitive Science 13, 295-355 (1989).
19. Leake D.B.: An Indexing Vocabulary for Case-Based Explanation, Proceedings of AAAI'91, AAAI Press/MIT Press, 10-15 (1991).
20. Domeshek E.: Indexing Stories as Social Advice, Proceedings AAAI'91, AAAI Press/MIT Press, 16-21 (1991).
21. Maiden N.A.M. & Sutcliffe A.G.: Exploiting Reusable Specifications Through Analogy, Communications of the ACM. 34(5), 55-64 (1992).
22. Rich C.: Inspection Methods in Programming, Technical Report TR-604, Cambridge MA, Artificial Intelligence Laboratory, MIT, 1981.
23. Soloway E. & Enrlich K.: Empirical Studies of Programming Knowledge, IEEE Transactions on Software Engineering 10(5), 595-609 (1984).
24. Greiner R.: Learning by Understanding Analogies, Artificial Intelligence 35, 81-125 (1988).
25. Russell S.J.: The Use of Knowledge in Analogy and Induction, Pitman (London) 1989.
26. Maiden N.A.M & Sutcliffe A.G.: Analogical Matching for Specification Retrieval, Proceedings 6th Knowledge-Based Software Engineering Conference, IEEE Computer Society Press, 108-116 (1991).
27 Fischer G., Nakakoji K., Otswald J., Stahl G. & Sumner T.: Embedding Computer-Based Critics in the Contexts of Design, Proceedings of INTERCHI'93, ed. S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel & T. White, ACM Press, 157-163 (1993).
28. Maiden N.A.M. & Sutcliffe A.G.: Requirements Engineering by Example: An Empirical Study, Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 104-112 (1993).
29. Maiden N.A.M. & Sutcliffe A.G.: The Domain Matcher: Architecture and Algorithms, Nature Report CU-93-00D, Department of Business Computing, City University (1993).
30. Jarke M., Pohl K., Jacobs S., Bubenko J., Assenova P., Holm P., Wangler P., Rolland C., Plihon V., J. Schmitt, Sutcliffe A.G., Jones S., Maiden N.A.M., Till D., Vassilou Y., Constantopoulos P. & Spandoudakis G.: Requirements Engineering: An Integrated View of Representation, Proceedings 4th European Software Engineering Conference, Garmesh-Partenkirchen, September 1993.
31. Sutcliffe A.G. & Maiden N.A.M.: Software Reusability: Delivering Productivity Gains or Short Cuts, Human Computer Interaction: Proceedings of INTERACT'90, edited by D. Diaper, D. Gilmore, G. Cockton & B. Shackel, North-Holland, 895-901 (1990).
32. Lange B.M. & Moher T.G.: Some Strategies of Reuse in an Object-Oriented Programming Environment, Proceedings of CHI'89, ed. K. Bice & C. Lewis, ACM Press, 69-73 (1989).
33. Maiden N.A.M. & Tyndale D.E., Reuse of Domain Abstractions During Requirements Engineering: an Explanation is Required, submitted.
34. Maiden N.A.M. & Sutcliffe A.G., Requirements Critiquing Using Domain Abstractions, submitted.
35. Riesbeck C.K. & Schank R.C.: Inside Case-based Reasoning, Lawrence Erlbaum Associates, Hillsdale NJ (1989).
36. Stasko J., Badre A. & Lewis C.: Do Algorithms Assist Learning ? an Empirical Study and Analysis, Proceedings INTERCHI'93, ed. S. Ashlund, K. Mullet, A. Henderson , E. Hollnagel & T. White, ACM Press, 61-66 (1993).
37. Lott J.: Children's Bedrooms, Conran Octopus, 1989.
38. Prieto-Diaz R.: Domain Analysis: An Introduction, ACM SIGSOFT Software Engineering Notes 15(2), April 1990, 47-54.
39. Rugg G. & McGeorge P.: On Laddering, Technical Report LG-4-92, Department of Psychology, University of Aberdeen, 1992.
40. Rich C. & Waters R.C.: The Programmer's Apprentice: A Research Overview, IEEE Computer, November 1988, 10-25.
41. Simon H.A.: The Structure of Ill-Structured Problems, Artificial Intelligence 4, 181-201 (1973).
42. Koenemann-Belliveau J., Moher T.G. & Robertson S.P., Empirical Studies of Programmers, 4th Workshop,, Ablex, Norwood NJ, 1992.

CASE-DELIVERER: Making Cases Relevant to the Task at Hand

Kumiyo Nakakoji

Department of Computer Science and Institute of Cognitive Science
University of Colorado
Campus Box 430
Boulder, Colorado 80309-0430; and

Software Engineering Laboratory
Software Research Associates, Inc., Tokyo, Japan

E-mail: kumiyo@cs.colorado.edu

Abstract. Designers are limited in exploiting a catalog knowledge base of design cases because they may be unable to articulate what they are looking for, or be unaware that potentially useful catalog examples exist. KID (Knowing-In-Design), a domain-oriented, knowledge-based design environment for the kitchen floor plan design, integrates the use of the catalog-base with its design tools. The information given through KIDSPECIFICATION (for specifying a design requirement) and KIDCONSTRUCTION (for graphically constructing a floor plan) provides representations of the designers' task at hand, and recorded design rationale in its argumentation-base is used to infer the relevance of catalog examples to the task at hand. The CASE-DELIVERER component orders catalog examples according to the partial specification, and the CATALOGEXPLORER subsystem allows designers to explore further the catalog space in terms of the task at hand. The study and assessment of the mechanisms have revealed that delivered cases helped designers reframe both a problem and a solution, and encouraged designers to articulate a new portion of design knowledge, which addresses the knowledge acquisition problem.

1 Introduction

Domain-oriented, knowledge-based design environments are computer systems that provide design tools and knowledge repositories that designers use for understanding, reflecting on, and framing their designs [6]. The environments augment skills of designers in managing and communicating with complexity of a design space, instead of modeling the cognitive processes of designers and automating them. This paper presents research efforts in embedding the use of a catalog base as a case library in such an environment to aide designers to exploit previously constructed design cases.

Design is ill-defined [16]. Specifying a problem and constructing a solution are intertwined. Every transformation of the specification of the problem provides the direction in which a partial solution is to be transformed, and every transformation of the constructed solution determines the direction in which the partial specification is to be transformed. While coevolving the specification and construction, designers gradually gain the understanding of the correspondence between a partial specification and a partial construction.

For example, let us take the kitchen design domain as an *object-to-think-with*. Kitchen designers gain their expertise through practice. They identify new heuristics by solving specific design tasks. In our preliminary study, while designing a floor plan for two cooks, a professional kitchen designer identified a new portion of design knowledge that a dishwasher door should not interfere with the work space for a stove because one working with the stove may step over the dishwasher door while the other is installing dishes in the dishwasher.

The design environments provide two types of design knowledge: (1) an argumentation-base stores heuristics that have been accumulated via recording design rationale, and (2) a catalog-base stores previously constructed design cases. For example, using the design environment, the above knowledge can be accumulated by storing design rationale as a form of argumentation (i.e., *Where should a dishwasher be? — A dishwasher should not face a stove. — If the kitchen is for two cooks, it is dangerous because one may step over the dishwasher door while using the stove*), and a design case in a catalog-base (i.e., a constructed floor plan for two cooks, which has a dishwasher *not* facing a stove).

Thus, using the design environment, designers could access such useful case-based design knowledge that solved problems similar to their own and a way to assess their partial solutions when no algorithmic method is available for evaluation [8]. However, the designers are limited in exploiting the design knowledge because they may be unable to articulate what they are looking for, or be unaware that potentially useful catalog examples exist. With the above example, in order to access useful catalog examples, designers who want to design a kitchen for two cooks have to know which stored floor plans are designed for two cooks, and how they are useful for solving their problem.

By integrating the knowledge bases with a specification component (for specifying design requirements) and a construction component (for constructing a floor plan), the design environment supports designers to access the catalog examples relevant to their task at hand. In this paper, first I describe problems of location of *useful* cases

in general, and presents a knowledge delivery paradigm as an approach. Then, I describe the mechanisms in terms of KID (Knowing-In-Design), an integrated, domain-oriented knowledge-based design environment for a kitchen floor plan design. Finally, I briefly discuss the result of user observations and assess the approach.

2 Retrieval of Useful Case Knowledge

2.1 Problem

Traditional information retrieval techniques cannot simply be applied to support the location of *useful* design cases. Problems and challenges of locating useful cases include:

- **Interdependency between information needs and problem-solving.** Designers cannot completely specify a design problem before starting to solve it. Designers cannot understand a problem without information about its context, but designers cannot meaningfully search for information without the orientation of a partial solution [15].
- **Difficulty of defining a set of indexes that will become useful later.** Different design situations may need to view a piece of knowledge differently. It is impossible to anticipate all possible design situations a priori [17], which makes a static indexing scheme for design cases inapplicable.
- **Need for integrating information search in design activities.** Information needs arise through a design task. Designers want to access information to solve a current design task, and should be able to retain the context of their current task.
- **Unawareness of the existence of potentially useful cases.** Designers are limited in making use of information because of the large and growing discrepancy between the amount of potentially relevant information and the amount any one designer can know and remember [5]. When designers are neither aware of the existence of potentially useful information nor aware of their information needs, no attempt will be made to access the information.

2.2 Approach: Knowledge Delivery

In human-human collaborative problem solving, both participants can adapt their own behavior according to the characteristics of the partners and by gradually gaining shared understanding. The shared understanding enables the partners to improve the communication process, to accelerate the discovery of either common or conflicting goals, to optimize the efficiency of the communication, and to increase the satisfaction of the partners [11].

A knowledge delivery mechanism is an instantiation of applying this idea to the collaboration between designers and design environments. Knowledge delivery mechanisms deliver "*the right knowledge, in the context of a problem or a service, at the right moment for designers to consider*" [3]. The mechanisms infer a designer's task at hand, detect the designer's information need, then present stored knowledge for the designer, who may be unaware of the existence of such useful design knowledge in the system. This paper describes design and implementation of a delivery mechanism, which delivers catalog examples in a design environment using the shared knowledge about a design task provided by a partial specification and construction. The mechanism is illustrated in the context of the KID (Knowing-In-Design) design environment for the kitchen floor plan design [10]. The system is implemented in the CLOS programming language, and runs on Symbolics Genera 8.1.

KID consists of:

1. KIDSPECIFICATION, which enables an explicit representation of the designer's goals and intentions with respect to the current design;
2. KIDCONSTRUCTION, which provides designers with a palette of domain abstractions and supports them to construct artifacts using direct manipulation styles;
3. the argumentation-base, which stores design rationale represented in the IBIS structure [2] (i.e., a network of nodes, consisting of issues, answers and arguments); and
4. the catalog-base, which stores completed floor plans (construction) together with associated specifications.

KIDSPECIFICATION and KIDCONSTRUCTION provide the explicit representations of a problem specification and a solution construction, which allow designers to coevolve a problem and a solution. Information given through the two components increases the system's shared understanding about the designers' intentions for the current task. Using the shared understanding about the task at hand, KID can deliver task-relevant information for the designers' perusal. The relevance is dynamically computed using heuristics (called *specification-linking rules*) identified through stored design rationale in the argumentation-base.

Two subsystems, CASE-DELIVERER and CATALOGEXPLORER, support designers to locate useful catalog examples. Cases stored in the catalog-base of KID are represented in the KANDOR knowledge base [12], including a construction (a floor plan) and a specification (a set of issue-answer pairs). In addition to access mechanisms provided by CATALOGEXPLORER [6] (such as retrieval by matching specification, retrieval by matching construction, and query-based search), CASE-DELIVERER automatically orders catalog examples according to the

partial specification provided through KIDSPECIFICATION. In the next section, I briefly describe the mechanism.

3 KID: Design Environments for Kitchen Design

Various knowledge representations used in KID is linked through the design rationale stored in the argumentation-base; that is, a network of nodes consisting of issues, answers, and arguments. A pair of issue and answer represents a design decision in terms of function, structure, or behavior at various levels of abstraction. An interdependency between two design decisions (i.e., issue-answer pairs) can be captured through the associated argument. KID uses *specification-linking rules* to represent such interdependencies.

KIDSPECIFICATION. The representation for a specification is a set of issue-answer pairs, designed after analyzing questionnaires used by professional kitchen designers to elicit design requirements from clients. KIDSPECIFICATION has been built as a hypertext interface, built on top of the argumentation base. Using KIDSPECIFICATION, designers can specify their design priorities by selecting and annotating alternative design decisions documented in the argumentation-base. Figure 1 shows an example of a specification.

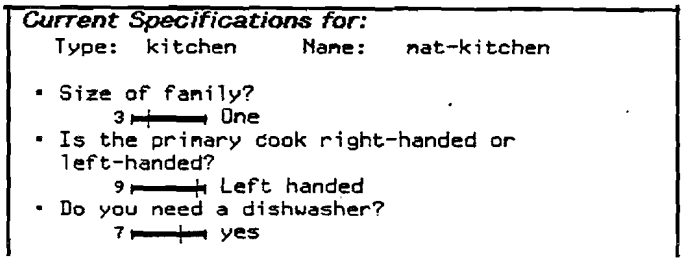


Figure 1: A Partial Specification in KIDSPECIFICATION

The summary of currently selected answers in KIDSPECIFICATION is provided in this window. Users can assign weights of relative importance to selected answers by moving associated sliders. In this figure, the user has put most importance to the left-handed requirement (i.e., 9 in the 1-10 scale) and little importance to the single-person household requirement (i.e., 3). The state of the specification component (i.e., a set of selected answers with assigned weights) is referred to as the current *partial specification*.

Although many of such issue-answer pairs have already been articulated through previous design efforts and accumulated by recording design rationale in the argumentation-base, if no prestored alternatives express their position, designers can add or modify information in the underlying argumentation-base using a property sheet. Designers can assign weights to the selected answers to articulate the relative importance of specified items.

Specification-Linking Rules. A specification-linking rule represents a computable interdependency between two issue-answer pairs; for example, *"Size-of-family=one → Type-of-sink=Single-bowl-sink"* implies that there is a relation between the size of a household and the type of a sink to be used in the kitchen design. This rule is based on the associated argument to the selection of the type of a sink, which says that a single-bowl-sink is enough for a single-person house-hold.

Specification-linking rules are derived by a mechanism (see Figure 2) that calculates the design constraints implied by a partial specification. The above rule is derived by finding issue-answer pairs that are implicated by the specification of *"Size-of-family=one."* In this case, the answer, *"Type-of-sink=Single-bowl-sink,"* to the issue, *"Which type of sink should be used,"* is implied because it is supported by an argument *"A single bowl sink is enough for a single-person household,"* which is associated with *"Size-of-family=one."* The mechanism is described further in Nakakoji [1993].

Link to KIDCONSTRUCTION. Some of the issue-answer pairs of KIDSPECIFICATION are related to construction situations, such as a need for a dishwasher, or a type of sink. In order to link the text representation of KIDSPECIFICATION to a graphic representation of KIDCONSTRUCTION, the system provides pre-defined predicates over the construction. The representation of a construction includes a list of design units used in a partially designed floor plan and their configuration information. The predicates determine whether a condition is satisfied in the partial construction, such as checking the existence of a single-bowl-sink. Using a property sheet provided by KIDSPECIFICATION, users can associate one of such pre-stored predicates with an issue-answer pair in the textual representation of KIDSPECIFICATION. Users are allowed to define a new predicate by using the MODIFIER system [7], if necessary.

Thus, when either an antecedent or a consequent of a specification-linking rule represents a construction situation, the rule provide a partial mapping between a specification requirement and a feature in the construction, forming a dependency network.

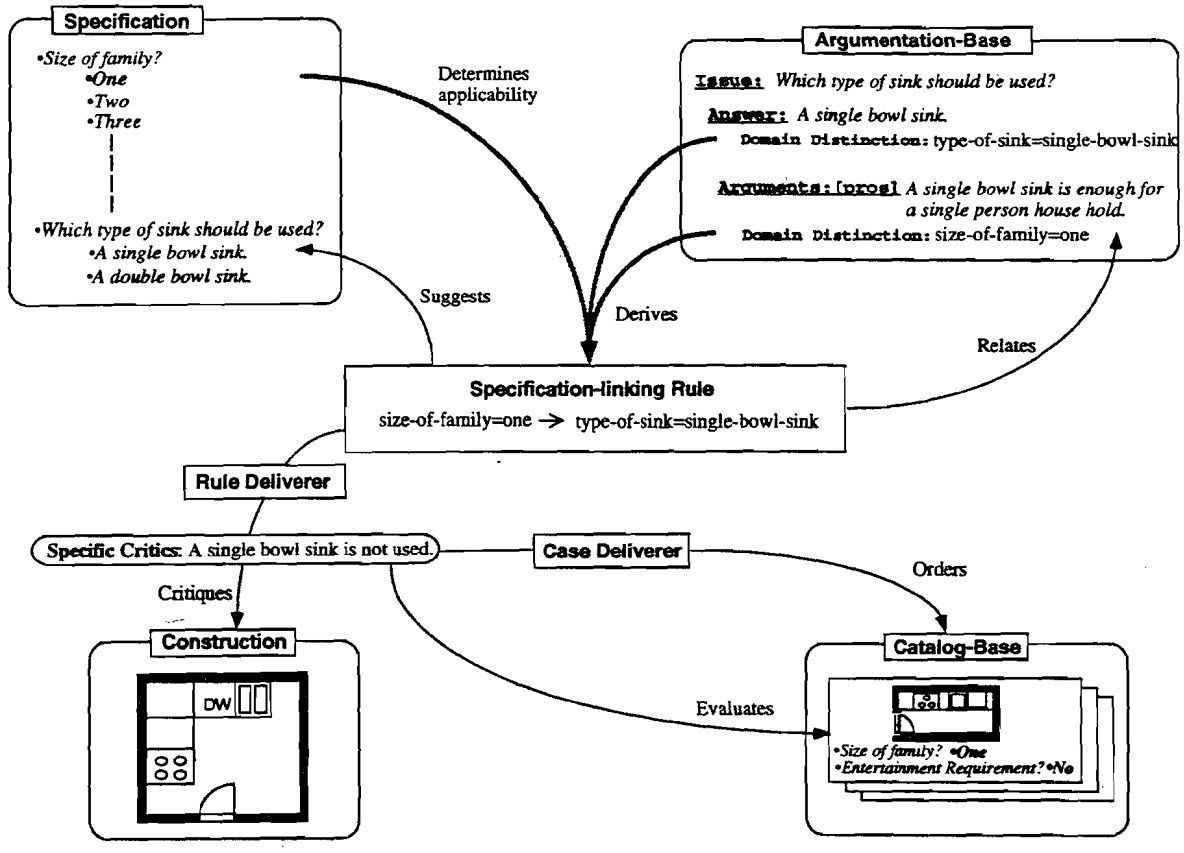


Figure 2: Integration of Components in KID

Specification-linking rules are derived from the argumentation, and the applicability is determined with a partial specification. CASE-DELIVERER uses the derived rules to order the catalog examples according to the partial specification. In addition, the specification-linking rules are used to make suggestions in KIDSPECIFICATION, to show a related argument in the argumentation base, to identify relevant critics as specific critics, and to evaluate a catalog example using the specific critics [10].

CASE-DELIVERER. A collection of consequents of rules represents required features for a construction inferred from the partial specification. When designers tentatively finish using KIDSPECIFICATION by using one of the other subsystems or explicitly request for retrieving useful catalog examples, CASE-DELIVERER uses the computed consequents to order the catalog examples in the catalog-base according to the "appropriateness" to the partial specification. The algorithm used by CASE-DELIVERER is briefly described below. The detail is described in Nakakoji [1993].

1. First, the system identifies the collective of specification-linking rules relating to the partial specification (i.e., a set of selected answers) using a forward chaining inference engine to the multiple depths of the dependency network (the depth can be changed by users). While collecting these rules, the system assigns a weight as relative importance to a consequent of each rule according to the weights assigned to the selected answers in KIDSPECIFICATION (see Figure 1) and the number of inference steps involved, in order to prioritize potentially conflicting consequents.
2. Consequents (i.e., issue-answer pairs) that are associated with predicates over the construction are identified. If the same consequent appears more than once, they are combined and the assigned importance values are summed.
3. For each floor plan (construction) of the catalog examples, CASE-DELIVERER determines whether or not each of the identified predicates is satisfied in the floor plan, and sums the assigned importance values of the satisfied predicates as an appropriateness value of the catalog example.
4. CASE-DELIVERER orders the catalog examples according to these values, and redisplay a list of catalog example names in the *Catalog* windows of KIDSPECIFICATION and KIDCONSTRUCTION.

CATALOGEXPLORER. CATALOGEXPLORER allows designers to further explore the catalog space. The system describes why and how catalog examples are ordered by CASE-DELIVERER, and allows them to retrieve examples in terms of the task at hand.

Consequents of the specification-linking rules that are used to order catalog examples can be displayed with the *Show Delivery Rationale* command in CATALOGEXPLORER; for example, *"A single-bowl sink should be used."*

Each of these messages is a mouse-sensitive link to the location of related arguments (see Figure 2). Selecting the message with a mouse accesses the related argumentation, and provides a starting point for browsing the argumentation-base. The *Evaluate Example* command allows designers to evaluate a catalog example in terms of the current specification by using critics [4].

Finally, CATALOGEXPLORER allows designers to search the catalog space with more control over search. The *Retrieval by Matching Specification* and *Retrieval by Matching Construction* commands allow designers to retrieve catalog examples that have similar features. The detail is provided in Fischer, Nakakoji [1991].

4 User Study: Knowledge Construction Facilitated by CASE-DELIVERER

KID has been studied by observing several subjects, including both domain-experts and novices in using the system. Test sessions were videotaped and the protocols were analyzed.

When presented with the ordered catalog examples, the subjects often used CATALOGEXPLORER, and either started to examine the example located at the top of the list, or asked for further explanations about why KID judged the example as the best example for their specification by accessing the underlying argument associated with the listed delivery rationale. Their response could be classified in the following three ways: (1) applied the delivered cases to reframe their partial design, (2) explored the related information space to the delivered cases, or (3) articulated new design knowledge by arguing against the underlying delivery rationale.

The reflection on their current partial construction and specification was often triggered by ordered catalog examples. Delivery of sometimes *unexpected* information was found to be an effective way to trigger the subjects to reflect on their task at hand. The subjects often discovered new features, which were breakdowns or important considerations they had not been aware of before, in catalog examples presented by CASE-DELIVERER.

Delivered catalog examples encouraged the subjects to further search the catalog-base. They often wanted to retrieve catalog examples that had the same feature discovered in one of the delivered catalog examples. There is evidence that people search longer for answers to questions when they believe they know the answer [14]. Thus, high *feelings of knowing* correlate with longer search time. When KID delivered information that was relevant to the task at hand, but not quite right, then they gained this "feeling of knowing," which made their information search longer.

The subjects often reacted to delivered knowledge and argued against the delivered knowledge in terms of their task at hand. When being given an object to think with, people start thinking about it and trace associations, which may be linked to tacit part of design knowledge [13]. Thus, it was easier for the subjects to become able to articulate new design knowledge than given no context.

5 Discussion

Having the catalog-base, KID can be viewed as a case-based design aiding system [9]. Embedding the use of the case-base within a design environment addresses several issues in the case-retrieval research. First, KID retrieves useful cases according to the explicit representation of designers' problem-solving goal provided by KIDSPECIFICATION, in addition to retrieving structurally similar cases. Second, instead of indexing cases at storage time by defining features a priori, the specification-linking rules are used to perform analogical matching to the users' task at hand. Third, the specification-linking rules are dynamically derived from the argumentation base. When designers add a new argument, the rules are immediately recomputed. Moreover, the rules are weighted according to the relative importance, or weights, that designers associate with selected answers. Thus, designers have more control over the retrieval.

In summary, CASE-DELIVERER of KID has the following characteristics.

- *Cases delivered help designers to reframe a partial problem as well as a solution.* Delivered knowledge is relevant to the task at hand, in terms of a partial specification and construction. By looking at the delivery rationale (why this knowledge is relevant to their partial problem specification), designers are often triggered to reframe not only a partial solution (which most case-based design assistant systems support) but also a partially framed problem.
- *Cases delivered facilitate learning-on-demand.* Because the specification-linking rules used to order catalog examples are derived from the argumentation base, KID can provide an explanation as to *why* some catalog examples are judged to be relevant to their task at hand. Designers have access not only to case-based information itself, but also to the underlying delivery rationale. Because the delivered knowledge is situated, it is easier for designers to understand the information.
- *Cases delivered facilitate knowledge acquisition.* Delivered knowledge encourages designers to articulate a new portion of design knowledge. Delivering knowledge to designers can be a *knowledge-attractor,* or a knowledge elicitation method [1], which encourages and helps designers to articulate and store design knowledge into the system, addressing the knowledge acquisition problem.

Since various types of design knowledge stored in KID are linked together, designers can easily explore the knowledge base relevant to their problem context. Embedded CASE-DELIVERER enables KID to be an intelligent

design assistant by having shared understanding about a designer's task at hand given through KIDSPECIFICATION and KIDCONSTRUCTION. Thus, KID increases the chance that designers will encounter useful design cases stored in the system. Such design cases can be accumulated by using the design environment, and the rules used for the case retrieval are derived from design rationale, which can also be accumulated by using KID, addressing the knowledge acquisition problem.

Acknowledgements

I thank the HCC group at the University of Colorado, who contributed to the conceptual framework and the systems discussed in this paper. I also thank Barbara Gibbons of Kitchen Connection at Boulder, Colorado, for her valuable time and commenting on the work. The research was supported by: the National Science Foundation under grants No. IRI-9015441 and MDR-9253425; the Colorado Advanced Software Institute under grants in 1990/91, 1991/92, 1992/93; US West Advanced Technologies; NYNEX Science and Technology Center, and by Software Research Associates, Inc. (Tokyo).

References

1. N. Bonnardel. Knowledge Elicitation Through Project Transfer: An Experimental Study. *The International Review: Behavior and Information Technology* (1993). (forthcoming).

2. J. Conklin, M. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *Transactions of Office Information Systems 6*, 4 (October 1988), 303-331.

3. Computer Science and Technology Board. *The National Challenge in Computer Science and Technology.* National Academy Press, Washington, D.C., 1988.

4. G. Fischer, K. Nakakoji, J. Ostwald, G. Stahl, T. Sumner. Embedding Computer-Based Critics in the Contexts of Design. *Human Factors in Computing Systems, INTERCHI'93 Conference Proceedings*, ACM, 1993, pp. 157-164.

5. G. Fischer, S. Henninger, K. Nakakoji. *DART: Integrating Information Delivery and Access Mechanisms.* Unpublished Manuscript.

6. G. Fischer, K. Nakakoji. Empowering Designers with Integrated Design Environments. In J. Gero (Ed.), *Artificial Intelligence in Design'91*, Butterworth-Heinemann Ltd, Oxford, England, 1991, pp. 191-209.

7. A. Girgensohn. *End-User Modifiability in Knowledge-Based Design Environments.* Ph.D. Thesis, Department of Computer Science, University of Colorado, Boulder, CO, 1992. Also available as TechReport CU-CS-595-92.

8. J.L. Kolodner. *What is Case-Based Reasoning?*. In AAAI'90 Tutorial on Case-Based Reasoning, pp. 1-32.

9. J.L. Kolodner. Improving Human Decision Making through Case-Based Decision Aiding. *AI Magazine 12*, 2 (Summer 1991), 52-68.

10. K. Nakakoji. *Increasing Shared Understanding of a Design Task between Designers and Design Environments: The Role of a Specification Component.* Ph.D. Thesis, Department of Computer Science, University of Colorado, Boulder, CO, 1993. Also available as TechReport CU-CS-651-93.

11. R. Oppermann. Adaptively Supported Adaptability. *Sixth European Conference on Cognitive Ergonomics, Human-Computer Interaction: Tasks and Organization (Balatonfuered, Hungary)*, September, 1992, pp. 255-270.

12. P.F. Patel-Schneider. *Small Can Be Beautiful in Knowledge Representation.* AI Technical Report 37, Schlumberger Palo Alto Research, October, 1984.

13. M. Polanyi. *The Tacit Dimension.* Doubleday, Garden City, NY, 1966.

14. L.M. Reder, F.E. Ritter. What Determines Initial Feeling of Knowing? Familiarity With Question Terms, Not With the Answer. *Journal of Experimental Psychology: Learning, Memory, and Cognition 18*, 3 (1992).

15. H.W.J. Rittel, M.M. Webber. Planning Problems are Wicked Problems. In N. Cross (Ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, 1984, pp. 135-144.

16. H.A. Simon. *The Sciences of the Artificial.* The MIT Press, Cambridge, MA, 1981.

17. L.A. Suchman. *Plans and Situated Actions.* Cambridge University Press, Cambridge, UK, 1987.

Finding Strategies in Organic Synthesis Planning with Case-based Reasoning

Amedeo Napoli and Jean Lieber
CRIN CNRS – INRIA Lorraine
B.P. 239 – 54506 Vandœuvre-lès-Nancy Cedex – France
(e-mail: napoli@loria.fr – lieber@loria.fr)

(Extended abstract)

1 Introduction

In this paper, we present an application of case-based reasoning to the research of strategies in the context of organic synthesis planning. The main objective of organic synthesis planning is to build new molecular structures using a goal-directed problem solving approach. Two main kinds of reasoning processes are employed in the system that we are developing for building new molecular structures. Classification-based reasoning is used for tactic purposes, to achieve *local goals* (or *subgoals*), e.g. selection of transformations that can be applied to modify specific parts of a molecular structure. Case-based reasoning is used for strategic purposes, to achieve *global goals*, e.g. selection of an adequate memorized synthesis plan on which can rely the building of a molecular structure. The study presented here can be also considered as an illustration of the possible integration of classification-based and case-based reasonings, in the context of organic synthesis planning (both kinds of reasoning rely on reminding).

2 An Object-Based Approach to Organic Synthesis

2.1 A Brief Introduction to Computer-Aided Synthesis Planning

One of the main object of organic synthesis is to build up molecules, called *target molecules*, from readily available starting materials [Corey *et al.*, 1985]. Once a target molecule has been chosen, the chemist searches for a *retrosynthetic plan*, which is constituted by a sequence of transformations leading from the target molecule to starting materials. A transformation is used to break down the target structure into (usually) simpler structures, called *precursors*. This problem solving approach, called the *retrosynthetic mode*, continues until the precursors are recognized as readily available starting materials. From a computing point of view, the retrosynthetic mode is similar to a goal-directed problem solving process. More precisely, the retrosynthetic mode depends on the perception of structural features of the target molecule, called *functional group* or *retrons*[1], that condition the application of transformations. A

[1] From a chemical point of view, there is a difference between retrons and functional groups, but this difference will not be taken in account in the paper.

functional group is a particular molecular substructure that determines a chemical function – the *functionality* of the molecule – and usually characterizes a family of molecules. Thus, functional groups are of primary importance for the categorization of molecular structures, according to their chemical properties.

The concepts of the retrosynthetic mode have been used as guidelines for the construction of a series of computer programs, usually called *computer-aided synthesis systems*. The main goal of these programs is to assist the chemist working on complex synthesis problems. Classically, the chemist is responsible for choosing strategies, e.g. choosing one transformation among several, and for deciding which precursors should be submitted to the system for further simplification. The system is responsible for selecting the actual transformations to be used and for displaying the precursors that result from these transformations. Thus, one of our goals is to automate the selection of strategies, using case-based reasoning.

2.2 A Classification-Based Approach to Organic Synthesis Planning

Our approach to organic synthesis planning relies on object-based formalisms, the emphasis being placed on the description of molecular structures and substructures such as functional groups [Napoli, 1990] [Napoli, 1992a]. Atoms and bonds are the primary chemical objects, and they are the components of the molecular structures, namely molecules and functional groups. Primary objects and molecular structures are implemented as frames [Masini *et al.*, 1991] and lay in an inheritance hierarchy called the *chemical taxonomy*. Frames are used to describe chemical objects, while transformations are implemented as operations attached to frames representing functional groups. All molecular structures are manipulated by classification-based reasoning as explained in the following.

Solving a synthesis problem relies on the retrieval of specific substructures, the functional groups, lying in a target molecule. Thus, the recognition of these substructures is an operation of primary importance underlying the retrosynthetic mode. To retrieve this information, the system uses a specific classification-based reasoning, according to a particular substructure/structure inclusion. Relying on the works done about subsumption in terminological logics [Nebel, 1990], we have defined a *subsumption* relation on molecular structures and their components, namely atoms and bonds [Napoli, 1992b]. This subsumption relation depends on the chemical type of atoms and bonds, and on the graph associated with the molecular structure. Briefly stated, a molecular structure *M1 subsumes* a molecular structure *M2* if there exists an isomorphism between the graph associated with *M1* and a subgraph associated with a substructure *M'2* of *M2*, and if the atoms and bonds in *M1* subsumes the corresponding atoms and bonds in *M'2* (according to their types and environments).

Thus, contrasting the chemical taxonomy, a second *tangled* hierarchy reflects the subsumption relations holding between memorized functional groups. This second hierarchy constitutes the *functional partonomy*, and can be seen as *orthogonal* to the chemical taxonomy. On the one hand, the inheritance relation determines the chemical taxonomy and is used for code factorization and property sharing between frames. On the other hand, the subsumption relation is used to organize functional substructures in the functional partonomy according to substructure/structure inclusion, and, as well, to guide a classification-based process producing synthesis plans.

More precisely, the process underlying the design of the synthesis plan of a target molecule relies on a *classification cycle* that makes explicit the dependencies holding between a new molecular structure, say TARGET, and the functional substructures lying in the functional

partonomy. The classification cycle proceeds with three main steps:

- *Instantiation*: the target molecule is represented by the new object TARGET.
- *Classification*: TARGET is classified in the functional partonomy. The system first searches for the most specific subsumers of TARGET (MSS), then, for the most general subsumees of TARGET (MGS). At last, TARGET is inserted in the functional partonomy, under its MSS and above its MGS. The subsumers of TARGET determine the functional groups lying in TARGET.
- *Operations*: the set of valid transformations that can be applied to TARGET is calculated according to a property sharing rule (this property sharing rule will not be described here but details are given in [Napoli, 1992a] and [Napoli, 1992b]). One transformation is chosen and is applied to simplify the target molecule into precursors. The precursors become new targets if they are not recognized as readily available starting materials, and the cycle continues.

The classification cycle modelizes the retrosynthetic mode and relies on a functional group-oriented approach, i.e. the application of a transformation depends on the functional groups included in the target molecule. However, more than one transformation can be selected during the second step of the classification cycle, leading to a classic "conflict set problem": what is the best transformation that must be chosen according to the the current target molecule and the actual chemical context? The choice of a transformation must be controlled by a synthesis strategy. In the following, we show how case-based reasoning is used to automate the selection of transformations.

3 Planning Syntheses with Case-based Reasoning

3.1 The Modelization of Synthesis Strategies

A *synthesis strategy* can be seen as an ordered sequence of *goals* or *objectives*, that correspond to the applications of transformations. Objectives usually are associated with transformations modifying the structure of a target molecule. However, in practice, there is usually a partial match between the target structure and the retron for a transformation. In this case, a single, or, more often, several *subgoals* related with a modification of the functionality of the target molecule[2], will rectify the mismatch and allow the transformation to be performed. Thus, the application of a transformation associated with a synthesis strategy can be conditioned by an ordered sequence of subgoals. Choosing a strategy instead of another means that an ordered sequence of actions to be performed is preferred among a set of other ordered sequences of actions. Classically, in computer-aided synthesis systems, strategy selection is usually fixed or left up to the chemist. An automated selection of strategies involve identification of substructures that suggest a specific approach, an ordering of the chosen approaches, a possible interactive verification done by the chemist, and, at last, the execution of the "best" strategy.

In our approach, strategies are represented as temporal objects, called *retrosynthetic routes* [Laurenço *et al.*, 1990]. A strategy consists in a sequence $(\{Oi\}, \{Ti\})$, where $\{Oi\}$

[2] Modifying the structure of a target molecule means breaking or building a bond, while modifying the functionality means substituting a functional group for another.

is a subsequence of objectives and $\{Ti\}$ is a subsequence of transformations associated with these objectives. Roughly, a retrosynthetic route can be seen as a tree whose root is the objective *O1* and whose nodes are the different objectives included in $\{Oi\}$ $(i \neq 1)$, obtained by application of the transformations included in $\{Ti\}$. Actually, there are a vertical and a horizontal temporal dimensions in a retrosynthetic route. The first consists in a sequence of temporal levels describing the temporal steps of the retrosynthetic route, leading from the root *O1* until the leaves of the tree; these leaves correspond to the last objectives of the considered retrosynthetic route. The second dimension is used to differentiate the nodes within a temporal level.

Given a target molecule TARGET, the choice of a retrosynthetic route relies on the similarity existing between an abstraction of TARGET and the root *O1* associated with memorized retrosynthetic routes. This similarity is determined using the subsumption relation defined on molecular structures. As more than one strategy can be selected, retrosynthetic routes must be classified according to their specific chemical characteristics. Thus, we define a subsumption relation for retrosynthetic routes that is inspired by works done on plan-based terminological reasoning [Devanbu and Litman, 1991] [Weida and Litman, 1992]. Briefly stated, a retrosynthetic route *RS1 subsumes* a retrosynthetic route *RS2* if the tree associated with *RS1* corresponds to a subtree *RS'2* of *RS2*, such that every node of *RS1* subsumes every corresponding node of *RS'2*. Note that nodes are molecular structures, and thus "subsumption" refers to molecular subsumption. Given two retrosynthetic routes, the most specific route (regarding the subsumption relation defined on routes) will be preferred.

At present, the work on the representation and handling of retrosynthetic routes is still in progress. However, we present in the following the current simplified transformation selection process, that applies to one-step routes and relies on a case-based reasoning approach.

3.2 The Handling of One-Step Retrosynthetic Routes

In the following, retrosynthetic routes are only one-step path corresponding to the application of a single transformation. Then, selecting a strategy means searching for a transformation T that can be applied to the target molecule TARGET, according to the *similarity* existing between TARGET and the functional groups guiding the application of T. The set of transformations lay in a specific base of transformations that can be seen, in this context, as the "memory of cases". The representation of a transformation T includes three main lists of characteristics, that are used to build an index associated with the transformation (for transformation retrieval) [Lieber, 1993] [Napoli and Lieber, 1993]:

- ACTIVE(T) is a list of active functional groups, that play an actual role in T, i.e. if *fg* is in ACTIVE(T), then at least one bond of *fg* is modified by the transformation T (see BONDS(T) below).
- INACTIVE(T) is a list of inactive functional groups, i.e. they do not play any role in the application of T. However, the inactive functional groups are taken in account because they can play a secondary role in a (more complete) retrosynthetic route, e.g. they can be used to reach a local subgoal for example.
- BONDS(T) is a list of bonds modified by the transformation T. This list contains quadruplets *(a1 a2 b1 b2)* describing a single bond, where *a1* and *a2* are the atoms at the extremities of the bond, *b1* and *b2* respectively denoting the type of the bond before and after the application of T.

In the following, we briefly explain how is calculated the similarity between the target molecule TARGET and the functional groups ACTIVE(T) conditioning the transformation T. This similarity depends on the functionality, described by the list FUNCTIONS(TARGET), and the structure, described by the list STRUCTURE(TARGET), of the target molecule TARGET. The list FUNCTIONS(TARGET) includes the functional groups of TARGET (in fact, they are given by the classification of TARGET in the functional partonomy). The second list STRUCTURE(TARGET) memorizes the connections of the carbon atoms of TARGET (*skeleton* of TARGET). Then, the global similarity existing between TARGET and ACTIVE(T) is calculated according to a formula that will not be detailed in this abstract (for more details, see [Lieber, 1993]). When the similarity is greater than a given threshold, the transformation T can be chosen, i.e. the retrosynthetic route materialized by T can be selected, and applied to TARGET. Relying on the matchings existing between TARGET and ACTIVE(T), the transformations of atoms and bonds in TARGET are performed according to the list of bonds changes given in BONDS(T) associated with T. Further, the matchings between atoms of TARGET and atoms of ACTIVE(T) do not need to be exact, and can depend on a degree of similarity.

Summarizing the above process in terms of a case-based reasoner [Riesbeck and Schank, 1989] [Hammond, 1990a] [Hammond, 1990b], the base of known transformations can be seen as the memory of cases. In this base, a transformation T can be reduced to the triplet (ACTIVE(T) INACTIVE(T) BONDS(T)), the index of the transformation T being ACTIVE(T). The pair (FUNCTIONS(TARGET) STRUCTURE(TARGET)) associated with the target molecule TARGET can be seen as the set of features characterizing TARGET, the index of the target molecule being FUNCTIONS(TARGET). Thus, searching for a case (a transformation T) in the memory relies on the matching existing between the index of the case, i.e. ACTIVE(T), and the index of the target molecule TARGET, i.e. FUNCTIONS(TARGET). At last, the adaptation step depends on the similarity of the atoms and bonds included in ACTIVE(T), i.e. involved in the transformation T, and the corresponding atoms and bonds in FUNCTIONS(TARGET) [Napoli and Lieber, 1993].

At present, we are extending this simplified process to more complicated retrosynthetic routes including more than one step. As one can see, much work must still be done for this real-world, complex and very interesting application.

References

[Corey *et al.*, 1985] E.J. Corey, A.K. Kong, and S.D. Rubenstein. Computer-Assisted Analysis in Organic Synthesis. *Science*, 228:408–418, 1985.

[Devanbu and Litman, 1991] P.T. Devanbu and D.J. Litman. Plan-based Terminological Reasoning. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Cambridge, Massachusetts*, pages 128–138, 1991.

[Hammond, 1990a] K.J. Hammond. Case-Based Planning: A Framework for Planning from Experience. *Cognitive Science*, 14(3):385–443, 1990.

[Hammond, 1990b] K.J. Hammond. Explaining and Repairing Plans That fail. *Artificial Intelligence*, 45(1–2):173–228, 1990.

[Laurenço *et al.*, 1990] C. Laurenço, M. Py, A. Napoli, J. Quinqueton, and B. Castro. Représentation de connaissances en synthèse organique à l'aide d'un langage à objets. *New Journal of Chemistry*, 14(12):921–931, 1990.

[Lieber, 1993] J. Lieber. Étude du raisonnement par cas. Rapport de Recherche 93-R-043, Centre de Recherche en Informatique de Nancy, 1993.

[Masini *et al.*, 1991] G. Masini, A. Napoli, D. Colnet, D. Léonard, and K. Tombre. *Object-Oriented Languages.* Academic Press, London, 1991.

[Napoli and Lieber, 1993] A. Napoli and J. Lieber. Classifying Knowledge for Reusability – An Application to Organic Synthesis Planning. In *Proceedings of the IJCAI'93 Workshop "Reuse of designs: an interdisciplinary cognitive approach", Chambéry,* 1993.

[Napoli, 1990] A. Napoli. Using Frame-Based Representation Languages to Describe Chemical Objects. *New Journal of Chemistry,* 14(12):913–919, 1990.

[Napoli, 1992a] A. Napoli. An Object-Based Approach to Computer-Aided Planning of Organic Syntheses. Rapport de Recherche 92-R-101, Centre de Recherche en Informatique de Nancy, 1992.

[Napoli, 1992b] A. Napoli. Subsumption and Classification-Based Reasoning in Object-Based Representations. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92), Vienna, Austria,* pages 425–429, 1992.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Computer Science 422. Springer-Verlag, Berlin, 1990.

[Riesbeck and Schank, 1989] C.K. Riesbeck and R.C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.

[Weida and Litman, 1992] R. Weida and D. Litman. Terminological Reasoning with Constraint Networks and an Application to Plan Recognition. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Cambridge, Massachusetts,* pages 282–293, 1992.

Case-Based Configuration in Technical Domains: Combining Case Selection and Modification[1]

Thomas Vietze

Universität Hamburg, FB Informatik
Bodenstedtstr. 16
D-22765 Hamburg
e-mail: vietze@informatik.uni-hamburg.de

Abstract. Generalization can serve as means for the selection of cases and for the adaption of selected cases to fit the current problem. We present an approach which tackles both tasks in an integrated manner. Cases are matched against the current problem specification. The case selection is based on the structure of the configured entities. If necessary, the case knowledge is generalized. It can be shown that the resulting generalized case knowledge can always be used as a basis for a configuration process which generates a fully specified solution.

1. Introduction

Human experts use case-based reasoning in a variety of different domains, e.g. in the field of law [1] or in medicine [9]. In this paper we address case-based reasoning in the field of technical configuration. To configure an artefact means to compose a configuration from a set of objects. Restrictions upon objects and attribute values have to be met. Typically, the resulting search space is huge. This is an important difference to diagnosis. By the integration of cases into the configuration process we hope to decrease the search space so that e.g. the presentation of "ad hoc" solutions becomes possible.

It is necessary to define how a case can be identified that is useful for solving the current problem. Usually this is achieved by some kind of similarity metric. If it is not proposed that selected cases do fit the current problem directly, a modification procedure has to be available. The modification of case knowledge is typical for configuration tasks.

With our approach we aim at a faster generation of possible solutions by search space reduction. If the presentation of one or more solutions in a timely fashion is possible, the usability of a configuration system is increased significantly. The user can then choose among alternatives in the form of concrete suggestions. The neccessity to decide about parameters in an early stage of the configuration process with unforseeable consequences for the whole configuration decreases. Whether this goal can be met depends highly on the employed methods for the case-based reasoning process and the structure of the domain processed.

In our configuration system all objects of the application domain and their interdependencies are described in a conceptual hierarchy and a constraint system [4]. Concepts form a taxonomic hierarchy that is used to represent classes and generalizations (superclasses) of objects and to specify their properties. The "Closed-World-Assumption" is valid, e.g., we assume that the part of the world described is complete. We may safely do so because the application is restricted to configuration in technical domains, where all objects and the possible attribute values are known in advance.

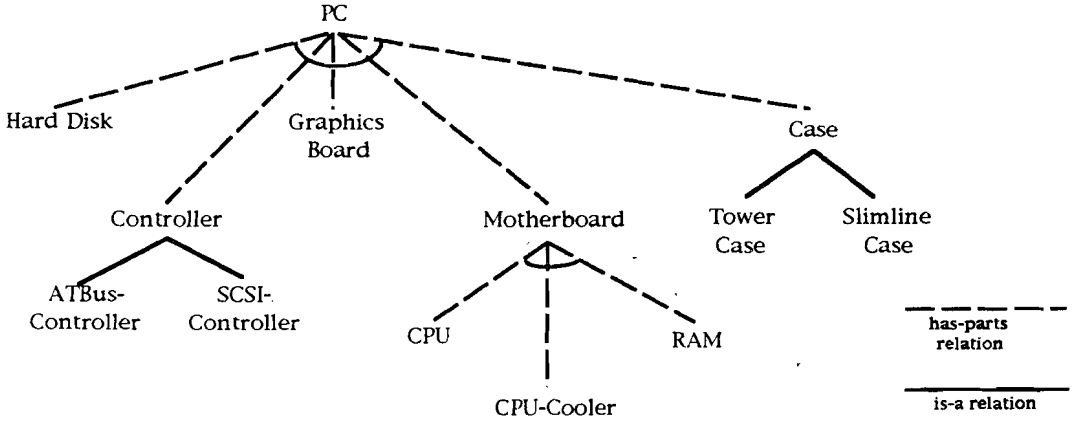


Figure 1: Partial conceptual hierarchy with is-a and has-parts relations

[1] This work has been partially supported by the BMFT (German Federal Ministry for Research and Technology) within the PROKON-project, grant no. ITW9101A6.

The part-of / has-parts relation forms a compositional hierarchy in addition to the taxonomic hierarchy. It describes the decomposition of configuration objects into components. It is thus a major guideline for the generation of a configuration. The has-parts relation is represented by set-valued attributes. The set descriptions contain references to concepts of components, optionally associated with number restrictions.

The compositional hierarchy together with the taxonomic hierarchy describes the set of all admissible configurations. Fig. 1 illustrates the taxonomical and compositional hierarchy for a simple example, the configuration of a personal computer. The solution of a configuration task is a correct instantiation of the conceptual hierarchy, e.g., a network of instances of concepts interconnected by has-parts / part-of relations with fully specified values for all attributes.

Dependencies between configuration objects are represented by constraints. These are maintained during the configuration process by means of a constraint net [13]. To configure a solution for a given problem instances are generated and successively specialized. The attribute values of these instances are specialized to terminal values,e.g. the hard disk capacity ranging from 20 to 1000 megabytes initially is specialized to 200 megabytes.

An important step in case-based configuration is the modification of selected cases to fit the current situation. This applies to planning as well [7]. This step is of minor importance for analytic problems like diagnosis but it is essential for synthesis tasks. We show how case selection can be combined with the modification of case knowledge. The exploitation of the internal structure of the cases [11] is the basis for a straightforward integration of the modified cases into the configuration process.

2. Case Selection

We give a defintion for generalization in the configuration context. First of all we introduce several necessary configuration related terms.

- The *domain model* is an explicit description of all concepts of the domain. These are arranged in a conceptual hierarchy [6] which consists of is-a and has-parts relations (e.g. Fig. 1).
- A *configuration* is defined as a set of instances ordered by a has-parts relation, e.g. a PC which consists of the parts motherboard, hard disk, contoller, keyboard, monitor etc.
- If the instances a configuration consists of are not fully specialized, we describe a set of possible fully specialized configurations, e.g. a set of hard disks with capacities ranging from 40 to 250 megabytes. We call these sets *partial configurations*.
- The current task specification is represented as an *initial partial configuration*, e.g. a PC with a 200 megabytes hard disk and a 14" color monitor.
- The *goal object* is the root of the compositional hierarchy of an initial partial configuration (e.g. a PC). It is a complete task specification already. Such a task specification can be expanded by further objects and specialized attribute values.
- A *case* is a fully specialized configuration, which has been successfully created as a solution of an old configuration problem (e.g. Fig. 3).
- A partial configuration P_1 *subsumes* another partial configuration P_2 if all fully specified configurations described by P_2 are also members of the set of fully specified configurations described by P_1.

For the selection of a case from the case-base, each case is tested as to whether it is subsumed by the initial partial configuration or not. If this is true, the case is a possible solution. Subsumption holds if the case is an element of the set of fully specialized configurations described by the initial partial configuration. This corresponds to KL-ONE. There subsumtion is defined as a subset relationship between the extensions of two concepts [2, 14].

If no case fits directly, each case is generalized [3, 8, 10] together with the initial partial configuration (Fig. 2). In contrast to the intuitive approach to use the old and the new task specifications for case selection, the old solution (the case) and the current task specification are used here [12].

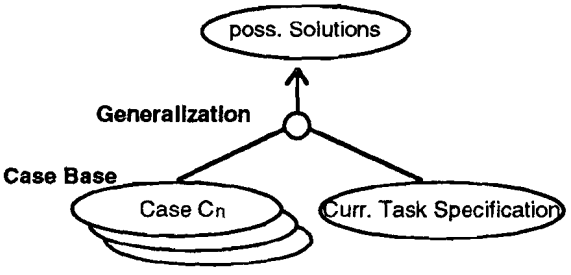


Figure 2: Generalization of single cases and the current task specification

The case selection process consists of three steps:

1. The goal object of the current task description and the goal object of the has-parts hierarchy of the case are compared. If the goal object does not subsume the root object, the case is rejected.
2. For each of the elements (instances) of the current task description, an object of the case must be found that is subsumed by it. If this is not possible for a certain object, this object is processed further in step 3. All objects of the case that have been used once are no longer available for another subsumption test.
3. Each object that could not be handled by step 2 is generalized with one of the remaining objects of the case. The generalization takes place on the basis of the domain model. A generalization that violates existing constraints is rejected.

To select one of the case's remaining instances for generalization, the set of available instances is tested for an instance that belongs to the same concept as the current instance of the initial partial configuration. If such an instance exists, generalization is realized as application of the union operator to the attribute values. If more then one instance exists, depth first search is applied.

If an instance of the same concept is not available, the most special common superconcept of the tasks instance and the case's instance is identified. From the set of most special common superconcepts which are derived from all available instances of the case, the most special one is selected. If there is no single most special concept, a concept can be chosen selectively if additional knowledge is available. The attribute values of a new instance of this concept are specialized to the union of the values of the two original instances. Because of the monotonous nature of the is-a relation it is always ensured that these values are a subset of the generalized concepts own values in the domain model. If at any point throughout the generalization process a corresponding instance of the case cannot be found, the case is rejected.

The result of the generalization process is a set of generalized cases, suitable for the solution of the current configuration problem. If the set is empty, no case passed the selection process. If more then one case was found, a case can be chosen. The selection can be based upon additional knowledge about the expected solution quality or the estimated modification costs. Another way is to leave the decision among the qualified cases to the user. In a more implementation-oriented notation the case selection process looks like this:

```
function select_case (initial_partial_configuration,case_base,domain_model)

qualified_cases <- ∅
for each element c in case_base do
   if is_a (root (c), goal_object (initial_partial_configuration))
      case_instances <- instances_of_case (c)
      instances_to_generalize <-
            instances_not_subsumed (case_instances,initial_partial_configuration)
      qualified_cases <- qualified_cases ∪
            generalize_if_necessary (c,instances_to_generalize,case_instances,domain_model)
   endif
endfor
return qualified_cases


function generalize_if_necessary (c,instances_to_generalize,case_instances,
                                  domain_model)
case_ok <- ∅
if instances_to_generalize = ∅
   case_ok <- c
else
   for each element e in instances_to_generalize do
      solved <- false
      for each element i in case_instances do
         if concept-of (e) = concept-of (i)
            replace (i,generalize_attributes (instantiate (concept-of (i)),e,i),c)
            case_instances <- case_instances \ i
            solved <- true
            exit for
         endif
      endfor
      if not solved
         most_special <- general_concept
         best_instance <- ∅
         for each element i in case_instances do
            if subsumep (most_special,most_special_superconcept (e,i,domain_model))
               most_special <- most_special_superconcept (e,i,domain_model)
```

```
                best_instance <- i
            endif
        endfor
        replace (i,generalize_attributes (instantiate (most_special),e,i),c)
    endif
  endfor
  if no_constraint_affected (c)
    case_ok <- c
  endif
endif

return case_ok
```

The functions *goal_object, root, concept_of, is_a* and *instantiate* are trivial. They will not be described any further. *instances_not_subsumed* checks for all instances, whether *replace* exchanges an instance of a structure (e.g. a case) for another instance (e.g. a generalized instance). The function *most_special_superconcept* extracts the most special common ancestor of two instances out of an is_a hierarchy (e.g. a domain model). *generalize_attributes* applies the union operator to all attribute values of the instances passed to the function. The function *no_constraint_affected* is used to distinguish between generalization with and without impact on existing constraints.

Out of the set of qualified cases only those are taken into account which can be integrated into the configuration process straightforward. This is true for those generalizations which are not connected to other parts of the whole configuration via relations except is-a and has-parts.

An example is given below. The case base consists of only two cases (Fig. 4 and 5). With respect to the initial partial configuration given in Fig. 3, both cases do not fit directly. In case 1 generalization is necessary because the SCSI-controller does not match the AT-bus-controller in the current task specification. In case 2 the tower case has to be generalized.

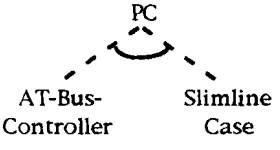


Figure 3: Initial partial configuration

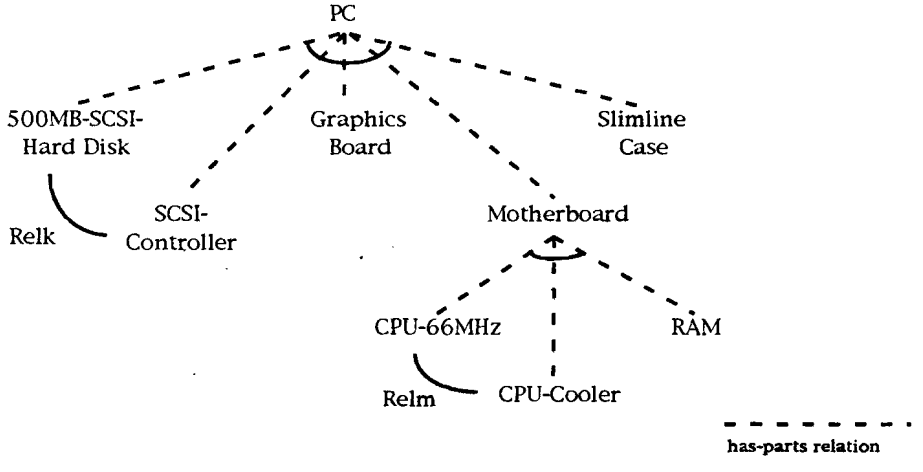


Figure 4: Case 1

Controller and hard disk have to be of the same type (here SCSI). This is modelled by the relation k (Rel_k). A generalization of the controller (or the hard disk) would affect the other part if the type SCSI (and the relation k) does not remain. The generalization is therefore rejected. A generalization of the slimline case on the other hand would not affect any other parts of the configuration. The generalization is accepted. The case knowledge that is integrated into the configuration process is the generalized case 2 (Fig. 6).

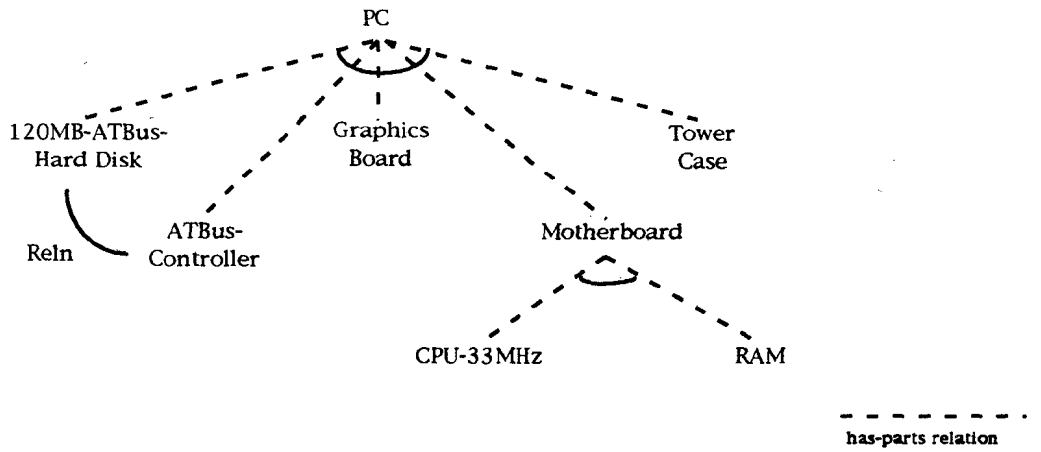


Figure 5: Case 2

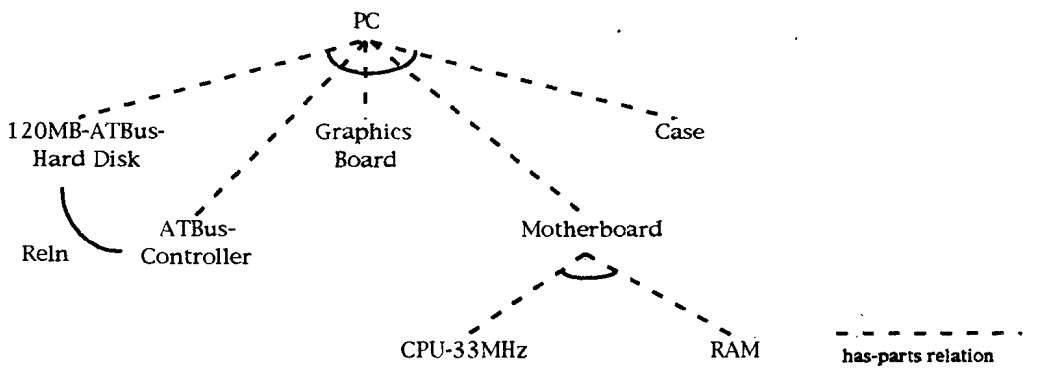


Figure 6: Generalized case 2

Generally speaking a consequence of this approach is that a case cannot be used if a generalization of a value is required that is connected with other values of the case via an arbitrary relation. On the other hand, the selection process will possibly offer a multitude of cases if the initial partial configuration is specified in a very general manner (e.g. just a PC).

3. Using Case Knowledge

We define configuration as search in the search space given by the domain model (see section 1). The search starts with the initial partial configuration. This configuration is successively refined until a fully specialized configuration is reached. If the initial partial configuration is substituted by generalized case knowledge, the search space is reduced. What is left of the original search space depends on the generalization (Fig. 7).

The search space that still remains has to be searched, e.g. depth first to reach a fully specified solution for the original task specification. This process can be interpreted as model-based configuration with an expanded task specification [5]. If a case can be used directly, no further search is necessary.

The integration of generalized case knowledge into the configuration process is always possible because a generalized case describes a set of fully specified solutions which is the same as a partial configuration. Consistency with the domain model is enforced via the constraint net.

4. Summary and outlook

We have shown how case selection can be combined with the modification of case knowledge. The output of the case selection process is a generalized case which is nothing else but a partial configuration describing a set of possible configurations. The partial configuration is then used as basis for the remaining configuration steps (Fig. 7). The reduction of the search space depends on the depth of the generalization. The search space can be reduced substantially if moderately generalized case knowledge is available (see section 3).

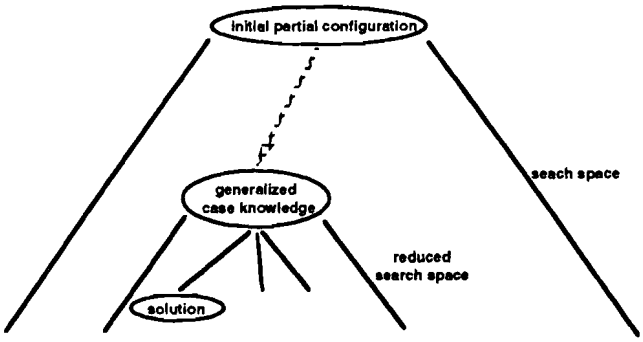


Figure 7: Integration of Generalized Case Knowledge into the Configuration Process

With this approach the use of a similarity metric can be avoided. A disadvantage in this context is that only a set of qualified cases can be identified, but not a single best case. We argue that it does not really matter which case out of the set of qualified cases is selected because the generalizations employed do not affect the configuration as a whole.

The solutions created with our case-based approach are implicitly represented in the domain model. A case not covered by the domain model will never be selected. Therefore the knowledge acquisition process is still necessary. However, the search through the space of possible configurations is guided by the selected case. In this view a case represents domain dependent control knowledge. This knowledge need not be formulated in general terms anymore, which makes knowledge acquisition easier.

A higher solution quality can be achived if the case-based approach offers better heuristics for single decisions then available in the domain model. Only if the available control knowledge as part of the domain model is incomplete, the solution quality can be improved by a case-based approach.

The restrictions upon our domain model are rather strict due to the fact that a complete model is required. Our future work aims at a more flexible approach which includes learning from cases that describe solutions outside the solution space given by the domain model.

References

1. W. Bain, Case-Based Reasoning: A Computer Model of Subjective Assessment, PhD thesis, Yale University, 1986
2. R.J. Brachmann, J.G. Schmolze, An overview of the KL-ONE knowledge representation system, Cognitive Science, 9(2) p.171-216, April 1985
3. W. Buntine, Generalized subsumption and its applications to induction and redundancy, Artificial Intelligence, 36:149-176
4. R. Cunis, Modellierung technischer Systeme in der Begriffshierarchie, in: Das PLAKON-Buch, p. 58-76, Springer-Verlag, 1991
5. A. Günter, Kai Pfitzner, Fallbasiertes Konstruieren mit Bibliothekslösungen, in: Das PLAKON-Buch, p. 131 ff., Springer-Verlag, 1991
6. A. Günter, R. Cunis, Flexible Control in Expert Systems for Construction Tasks, in: Journal of Applied Intelligence 2, p. 369-385, Kluwer Academic Publishers, 1992
7. K. Hammond, Case-Based Planning: Viewing Planning as a Memory Task, Academic Press, Boston, MA, 1989
8. Y. Kodratoff, J. Ganascia, Improving the generalization step in learning, in: R. Michalski, J. Carbonell, T. Mitchell (editors) Machine Learning - An Artificial Intelligence Approach, p. 215-244, Morgan Kaufmann, 1986
9. P. Koton, Reasoning about evidence in causal explanation, Los Altos, in: Proc. AAAI, p. 256 ff., 1988
10. T. Mitchell, Generalization as Search, Artificial Intelligence, 18(2):203-226
11. J. Paulokat, S. Weß, Fallauswahl und fallbasierte Steuerung bei der nichtlinearen, hierarchischen Planung, in: Arbeitspapiere der GMD, Nr. 723, 1993
12. K. Pfitzner, Die Auswahl von Bibliothekslösungen mittels induzierter Problemklassen, in: Beiträge zum 4. Workshop Planen und Konfigurieren, Ulm, 1990
13. I. Syska, A. Günter, R. Cunis, H, Peters, H. Bode, Solving construction tasks with a cooperating constraint system, in: Proc. Expert Systems 88, p. 199-209, Brighton, 1988
14. K. von Luck, KL-ONE: Eine Einführung, Wissenschaftliches Zentrum der IBM, Institut für Wissensbasierte Systeme, Report 106, Februar 1990

Chapter 6

Integrated Problem Solving and Learning Architectures

Explanation-Driven Retrieval, Reuse and Learning of Cases

Agnar Aamodt

University of Trondheim, Department of Informatics
N-7055 Dragvoll, Norway
agnar@ifi.unit.no

Abstract. A method for integrated case-based and generalization-based reasoning and learning is described. The primary role of general domain knowledge is to provide explanatory support for the case-based processes. A general explanation engine - the ACTIVATE-EXPLAIN-FOCUS cycle - utilizes a presumably rich, multirelational knowledge model in producing context-dependent explanations.

1. Introduction

Case-based reasoning covers a wide variety of methods. While some methods emphasize problem solving and learning by use of specific cases *instead of* general domain knowledge, others use general knowledge[1] *combined with* cases. Among the latter, various approaches to what types of general knowledge to incorporate, as well as to how general knowledge is used, are taken. General knowledge may be used for an additional problem solving method, e.g. a method that is applied if the case-based method fails, and/or it may be used within the case-based method itself. The general knowledge may be of a shallow, associational type (e.g. a set of heuristic rules), or deeper, more principled knowledge (e.g. a model combining causal, functional and componential knowledge).

This paper addresses the use of general domain knowledge *within* a case-based method. The focus is on the integrated utilization of case-specific and general knowledge. Our work aims at improved AI methods for *knowledge-based decision-support* in real world, open and *weak theory* domains[2] . Examples of such domains include medical diagnosis, law, corporate planning, economical assessment, and most engineering domains. A counter-example would be a mathematical domain, or a closed technical domain. A strong motivation for a case-based approach to this problem is the need for *adaptive* behaviour of our systems, i.e. the ability to continually learn from each problem solving experience. General knowledge is needed in order to achieve an acceptable degree of *competence* and *robustness* of a case-based reasoner's problem solving and learning capability. As domains get more open and complex, the more important it will become to base, e.g., the matching of cases, the modification of solutions, and the learning of new cases, on an understanding of the current problem within its *problem solving context*. This is different from relying solely on general and global criteria and metrics. It is in the tight coupling of case-based and generalization-based approaches we find the strongest potential for realizing the competent and flexible behaviour we would like to see in future AI systems. This is the hypothesis we investigate.

The fact that a domain is open and has a weak theory does not necessarily imply that there is little general knowledge available. More often it implies that the general knowledge of the domain is theoretically uncertain and incomplete. Such knowledge may be interpreted and processed by methods that are able to draw plausible inferences from a combination of the various types of knowledge available. This is here viewed as an abductive *explanation* process[3] - both at the top level where the goal for example is to explain a patient's symptoms in terms of the disease that causes them, and at a more detailed level, such as explaining why a particular diagnostic hypothesis should be preferred over another. The approach is based on the CREEK[4] architecture [Aamodt-90a, Aamodt-90b], and may be viewed as partly a specialization and concretion, partly an extension of this system as initially specified in [Aamodt-91].

The next chapter introduces the notion of explanation-driven case-based reasoning, and presents the basic model. This is followed by an outline of how case knowledge and general knowledge is represented in our system, and a more detailed description of the explanation-driven reasoning method, specified for each of the CBR tasks Retrieval, Reuse, and Learning. Finally, this approach is discussed by comparing it to some related methods.

2. Explanation-Driven Case-Based Reasoning

Core tasks of a case-based reasoner are the extraction of relevant features to describe a problem, the assessment of similarity between a new problem and previous cases, the adaptation of a previous solution within a new context, the identifying what to retain from a case just solved, and the learning of indexes for memorizing new cases. Earlier CBR systems (e.g. [Kolodner-83, Carbonell-86, Rissland-87]) adopted largely global and context-independent strategies for dealing with these problems, such as a fixed set of problem features, and syntax-

[1]Throughout this paper, the term 'general knowledge' refers to general - or generalized - domain knowledge. If general knowledge in the sense of domain-independent (e.g. common sense) knowledge is meant, this is explicity stated.

[2]An open domain is a domain which cannot be realistically modelled unless relationships between the target system (artifact or natural system) and the external, changing world are included. A weak theory domain is a domain in which important relations between concepts are uncertain.

[3]Abduction is here viewed as an "inference to the best explanation" (see, e.g. [Thagaard-88]), and covers both the generation and evaluation of hypotheses.

[4]Case-based Reasoning through Extensive Explicit Knowledge

oriented similarity measures and storage schemes. This is analogous to the earlier days of the rule-based expert systems field, when emphasis was on associational knowledge in terms of compiled or empirical rules. The deficiency of this approach in terms of problem solving competence and robustness lead to the notion of "second generation expert systems", where deep models are used to support and extend rule-based reasoning.

In CBR we have also seen a recent up-growth of methods that - to a varying degree - combine case-based reasoning with reasoning from explicit models of general knowledge [e.g. Hammond-86, Kolodner-87, Koton-89, Schank-89, Porter-90, Branting-91]. Our work follows up on this line of research, but put an even stronger emphasis on the role of general knowledge within the CBR methods. The emphasis is on identifying general principles and basic methods for how this synergy can be achieved and utilized within all reasoning tasks and for all levels and types of problems to solve.

Note that the word 'problem' is used in a general sense in this paper. For example, solving a problem may be to find the fault of a car (solving a diagnosis problem), as well as to assess a legal situation in a court (solving an interpretation problem)[1]. A problem is defined by a goal (what to achieve) which in turn sets up one or more tasks (what to do). To distinguish between external, application related tasks of a problem solver, and tasks set up by the systems own reasoning process, the first will be referred to as *application tasks*, while the latter are named *reasoning tasks*. Hence, learning tasks are also reasoning tasks. A task gets done by applying a method to it [Steels-90].

In the following a method that utilizes general knowledge extensively as an integrated part of a case-based reasoning system is described. The primary role of general domain knowledge is to produce explanations to support and control the case-based processes, which is why we refer to the approach as *explanation-driven case-based reasoning*. A generic mechanism, called the "explanation engine", constitutes the fundamental reasoning method. It splits a reasoning task into the three subtasks ACTIVATE, EXPLAIN, and FOCUS - as illustrated in figure 1.

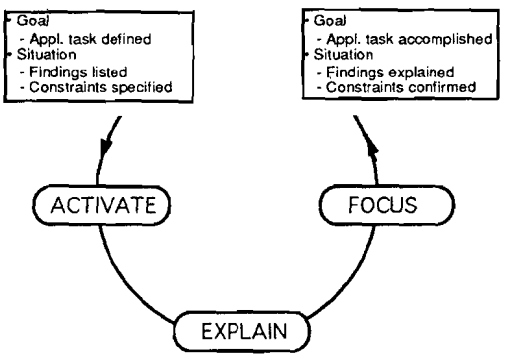


Figure 1. The Three Subtasks of the Explanation Engine

The reasoning methods of these tasks get a significant part of their power from the underlying representation system, a densely coupled semantic network where nodes as well as relations are represented as frame concepts. All knowledge, general as well as case specific, are represented as frame structures. At the low level, the frame interpreter makes use of basic inference methods such as various forms of property inheritance, low level frame matching, and constraint propagation. In addition to the system's own model of general domain knowledge, the explanation engine also assumes that there is a competent user at the terminal. Hence it will interact with the user in order to confirm conclusions, solve explanation conflicts, etc., whenever general knowledge is missing or contradictory.

The methods underlying the three above tasks operate briefly as follows:

ACTIVATE takes a problem specification in terms of a goal and a situation description and generates a set of concepts suggested as relevant for further processing. Two methods are used to achieve this: *Spreading activation* along appropriate relations in the semantic network, and *reminding* by following earlier established case links. The appropriate spreading relations are determined as part of the knowledge acquisition process.

EXPLAIN is the core method that builds up support for concepts identified by ACTIVATE. These concepts may be of any kind, e.g. an inferred feature, a proposed solution, a failure repair, an inference action to take, or a new case to be learned. The method of generating and evaluating explanations searches through activated parts of the semantic network, and follows the paths of cumulatively highest explanatory strength. In assessing this strength, it makes use of *default explanatory strengths* attached to each semantic relation and each meaningful combination of two successive relations. A 'strength-and-dependency table' for the semantic relations is defined as part of the knowledge acquisition and modeling process[2]. The strengths may have *contextual constraints* attached to them. Further, the EXPLAIN method has at its disposal a set of *explanation evaluation strategies* in the form of decision rules. An algorithm computes the resulting explanatory strength as the search proceeds.

[1]It is sometimes also useful to view learning as a type of problem solving (i.e. solving a learning problem), particularly when explicitly modeling the behaviour of systems. However, in accordance with the history of CBR, problem solving and learning are here viewed as different tasks.

[2]The basic method of utilizing combined relational strengths for search and evaluation of explanatory paths was implemented in the KNOWIT system, delivered to ESA as a prototype for knowledge-based information retrieval [Sølvberg-92].

FOCUS makes the final selection among competing concepts, when that is needed. It uses info about *given priorities*, and/or knowledge about the *reasoning goal* and possible *constraints of content or form* of the resulting concept. These constraints are typically external, pragmatic constraints defined by the application environment and current problem solving situation.

3. Knowledge Representation

The main representational concern of CBR research has been the representation of case knowledge, i.e. the contents and form of a case, the memory structures and indexes. For an explanation-driven approach, however, the representation of general knowledge is equally important.

The CREEK architecture combines case-based and generalization-based knowledge within an integrated system design. All types of knowledge are represented within the same representation system, a frame-based language called CreekL. This is an open frame system of the FRL [Roberts-77] and KRL style, implemented in CommonLisp. Concepts are represented as 4-level structures of slots, facets, value expressions, and value fields. CreekL also incorporates features of CYCL [Lenat-89], such as the explicit representation of relations as concepts and inverses for all relations. Each relation (slot) and symbolic value (filler) defining a concept (frame) is defined in its own frame. This results in a densely coupled knowledge model that integrates concept definitions, rules and cases, and object-level as well as control level knowledge. The architecture contains explicit knowledge models at the control level for application problem solving strategies and task structures, as well as for internal control of reasoning methods and learning. This enables a reflective system that reasons about its own methods.

Below, a frame representing the concept car is shown (as a pretty-printed lisp structure with most parentheses removed) with its slots, facets and values. Facets are mainly used to represent specific value types (e.g. defaults, value-dimensions or ranges), constraints on the values of a particular frame-slot combination (e.g. a class specification for legal slot values), and demons - Lisp functions that returns a value or performs an operation (if-needed, if-added, etc.).

car		
subclass-of	value	motorized-vehicle means-of-transportation sporting-gear
has-colour	value-class	colour
has-number-of-wheels	default	4
has-age	value-dimension	years
	if-needed	(time-difference *current-year* self.has-production-year)

The frames, interconnected by their slots, form a semantic network of concept nodes and relation links. Figure 2 illustrates this perspective to the CreekL knowledge structure, and shows - for a small excerpt of a knowledge base - the tight integration between general and case-specific knowledge. Example concepts are taken from a domain of car starting problems.

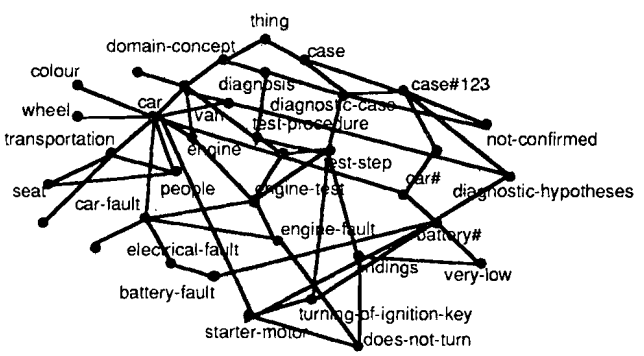


Figure 2: The Semantic Network Structure of CreekL Knowledge

Relation names are left out in the figure, and so are some node names. The interconnected, unified structure is emphasized. The links represent a wide range of relations, such as taxonomic, functional, and causal relations (has-subclass, has-instance, has-part, has-function, has-state, causes), and differential links between cases (has-differential-case).

In CreekL, concepts are described by their typical properties, which are inherited by more specialized concepts and instances. Inheritance is not limited to subclass and instance relations only, but may be described as a property of any semantic relation or combination of relations. For example, a spatial relationship expressed by a located-in relation (A located-in B) may be inherited along a has-part relation (A has-part C, inferring C located-in B). This would express that components of an object are located in the same place as the object itself - which is not universally true, but dependent on the context. The context dependency can be represented as a type of constraints on the slot inheritance-relation-for in the has-part frame.

An explanation in CREEK is a structure consisting of a single relationship or a chain of relationships. The supporting strengths of an explanation is evaluated and assigned a numeric strength value. Explanations are stored within the frame of the concept to which the explanation belongs. For example, an explanation for why a starter motor will not turn is represented as:

```
starter-motor-1
   instance-of          value              starter-motor
   part-of              value              car-1
   has-turning-status   expected-value     (turns)
                        value              (does-not-turn
                                             (0.9  (battery-1.has-voltage.very-low)
                                                   (battery-1.instance-of.battery)
                                                   ((battery.has-voltage.very-low) causes
                                                    (starter-motor.has-turning-status.does-not-turn))
                                                   (starter-motor-1.instance-of.starter-motor)))
```

Explanations are also stored in cases, for successful as well as for failed solutions. A stored case is a rich source of information, containing the following slot types (example from diagnostic domain):

problem goal	successful diagnosis	expl.of successful diagnosis
relevant findings	successful repair	expl. of successful repair
differential findings	failed diagnosis	expl. of failed diagnosis
differential cases	failed repair	expl. of failed repair

In addition, a case also contains 'book-keeping' information like its status in the problem solving and learning process, time reference, the number of times it has been used to solve new problems, etc. Three slot types of a case serve the role as index links for initial case retrieval: relevant findings (favours retrieval), differential findings (disfavours retrieval), and solutions (e.g. faults and repairs).

5. Case Retrieval, Reuse and Learning

At the top level, a case-based reasoning process is captured by the three tasks Retrieval, Reuse, and Learning. Retrieval captures the subtasks up to identification of the best matching case. Reuse includes possible modifications of a past solution as well as solution evaluation. Learning is the process that follows the successful or failed attempt to solve an application problem. The explanation engine utilizes its ACTIVATE-EXPLAIN-FOCUS method to combine case-based and generalization-based reasoning in all the three subtasks. Below, its main reasoning tasks and mechanisms are outlined:

Retrieval

The goal is to return the best matching case from the case base. Its input is whatever is known about the problem to be solved.

ACTIVATE has two subtasks. One is to *determine a relevant broad context* for the problem. We are assuming a large knowledge base, and want to activate just the part of this knowledge base that is potentially relevant. The broad context is determined by a method called goal-focused spreading activation. This method first activates all goal relevant concepts, i.e. the goal itself as well as concepts linked to the goal by taxonomic, causal, and functional relations. By spreading recursively along such relations, this produces a sphere of concepts relevant to the problem goal. Then a similar spreading process starts out from the problem findings. All concepts within the goal sphere or findings sphere, as well as concepts that lie on paths which directly or indirectly connect any two concepts in the two spheres, are marked as activated. The set of activated concepts constitute the part of the knowledge base that will be used for further inferencing. The other subtask of Activate uses the findings as indexes to the case base to *retrieve a set of cases* whose matching strength is above a certain threshold. A concept classified as a finding has a slot called relevant-finding-for which holds a list of cases and a computed relevance factor for the finding with respect to each case.

EXPLAIN is the task to *evaluate the matching* between the cases in set and the current problem. This basically means to explain the relevance to the problem for findings that matches well, and to explain away mismatches in findings. The latter is the most challenging, and is performed by a search for explanation paths in the general knowledge, by starting out from the finding and the solution of the retrieved case. This may introduce constraints on the solution of the problem, since a mismatch may be explained away only if some conditions are fulfilled. The user may be consulted here.

FOCUS makes the *final selection of the best case*, or rejects all of them. The case with the strongest explanatory justification of its findings will normally be selected. If in doubt, other pragmatic or external criteria are applied.

Reuse

The goal is to use the solution of a previous case in solving a new problem, usually involving some kind of modification of the past solution.

ACTIVATE starts out from the solution of the best matching case, and spreads activation to concepts representing *expected findings* not already accounted for. Spreading relations typically include causal and functional relations, as well as direct associations (e.g. implies and cooccurs-with relations). In addition, possibly *risky consequences* of applying the solution are also activated.

EXPLAIN has two main subtasks. One is to *evaluate the solution* proposed. It starts by attempting to justify that there is no danger for effects of risky consequences. Next, expected findings are explained as either relevant or irrelevant to the current problem context. An attempt is made to infer the relevant expectations (explain their presence in the current problem) before asking the user. If the expectations are covered for, control is given to the Focus task. If not, the second Explain task, *modification of the solution*, is triggered. An attempt is made to produce an explanation structure that justifies a replacement or tweaking of the solution. For example: In our car-starting domain findings are that the engine turns, but the car will not start. The solution of the retrieved case is faulty carburettor valve. An explanation path to the findings is carburettor-valve-stuck causes too-rich-gas-mixture-in-cylinder causes no-chamber-ignition causes engine-does-not-fire. However, the

carburettor turns out to be OK. By searching backwards along this explanation path, looking for other explanations of its states, it turns out that no-chamber-ignition is also caused by water-in-gas-mixture. The control is passed back to Activate which derives the expected finding water-in-gas-tank. This is then confirmed by the user.

FOCUS is usually a small task, unless the explanation process comes out with several solutions. It checks whether a proposed solution confirms with external requirements, and proposed its suggested solution to the user.

Learning
The goal is to capture the experience from the problem just solved, by constructing a new case and/or modifying parts of the knowledge base.

ACTIVATE here works on the structure activated by Retrieve and Reuse, and extracts potential concepts and structures for learning. This primarily means possible contents of a new case, but new or modified concepts that may have been introduced by the user is also marked for the learning process.

EXPLAIN has three subtasks. The first is to justify whether a new case needs to be constructed or not. It is done if no similar past case was found, if a retrieved solution needed significant modification, or if the set of relevant problem findings are sufficiently different from that of the case. The two latter criteria involves explanation processes, to assess the significance of a modification or of feature differences. Learning of generalizations does not play a strong role in this method, but a lazy generalization of values for findings is done if justified by an explanation or by the user. Given that a new case is to be stored, the second subtask is to assess the importance (degree of necessity) and predictive strength (degree of sufficiency) of case findings, in order to compute their relevance factors.

FOCUS is the task of putting together the new structures into a case frame, and actually storing the new or modified case and its indexes. Following this, a test is run by entering the same problem a second time, and checking whether its solution procedure is improved.

5. Discussion

Case-based reasoning has shown to be a powerful problem solving and learning paradigm for a varity of application domains. Characteristics of the domain type being addressed is important in determining the type of CBR method to use. In closed and well-defined domains, the need for supportive general knowledge is much less than in open domains, and may often be compiled beforehand into global metrics of similarity and other general operators [Richter-91]. When explicit general knowledge is integrated into the CBR processes, two synergy effects are immediately seen: One is to provide explanation-based control and guidance to the case processes, by, e.g., focusing on particular goals and tasks, constraining search, and supporting proposed hypotheses. As previously stated, this is the synergy effect aimed for here. The other synergy effect is the kind of 'inverse' effect achieved by using the cases within the explanation process itself, i.e. a case-driven explanation process (as in the SWALE system [Schank-89]). Methods developed from the latter motivation may also be useful to achieve the former effect, but this is not presently part of our method.

Several methods have been developed that make use of explicit models of general knowledge in its case-based processes [e.g. Hammond-86, Kolodner-87, Koton-89, Porter-90, Branting-91]. However, although representing very important contributions to this research, methods that have been proposed typically focus on one or a small subset of the CBR tasks. An exception is CASEY [Koton-89], but that approach relies on a strong knowledge model, and leaves out the interactive cooperation with the user which is needed in open and weak theory domains. In a sense, our approach shares the widespread use of general knowledge with CASEY and the interactive role of the user with PROTOS [Bareiss-88]. Recent suggestions for integrated architectures [Althoff-91, Ram-92, Plaza-93, Manago-93] represent interesting work towards more unified methods, but so far the problem has been addressed only partly, at a high and abstracted level, or for closed domains. The MOLTKE system [Althoff-91] is an example of the latter.

The knowledge-intensive approach we have taken has forced us to pay a lot of attention to the knowledge representation problem. Related methods have also had to address this issue to some extent. CASEY [Koton-89] uses a pre-existing causal model of general knowledge, represented as a causal network augmented with probability estimates for mere associations between features and diagnostic states. Cases are held in a separate structure, organized as a Schank/Kolodner type of dynamic memory. PROTOS [Bareiss-88] has an integrated structure consisting of a semantic network of domain categories linked by a variety of relations (causal, functional, associational, etc.), in which cases are linked as exemplars of diagnostic categories. The CREEK approach is most similar to PROTOS in this respect, but there are significant differences in the way cases are integrated with general knowledge, as well as how they are indexed and used.

Modeling and representation of knowledge in the explanation-driven CBR approach is in general viewed as a knowledge engineering problem. As such it is subject to the problems, methods and tools addressed by the knowledge acquisition community. Unlike some other motivations, for example PROTOS', we do not advocate CBR as an alternative answer to the *initial knowledge modeling* problem, rather as an approach to the problem of *continuous knowledge maintenance*. Hence, a view of knowledge modelling as basically a top-down modeling process [Steels-90, Chandra-92, Wielinga-92] is here merged with the bottom-up oriented view provided by learning from experience [Van de Velde-92]. The dominant role of top down modeling is weakened in favour of a more iterative development process for knowledge-based systems.

6. Status and Further Work

The system described here is under implementation in our department. An experimental evaluation of the method will therefore have to wait. It's plausibility, however, is supported by the integration of two approaches which each has a lot of merit, and the fact that exisiting approaches to knowledge-intensive CBR has shown promising results.

All CBR systems use indexes in one way or another for case retrieval, but there are a lot of unresolved problems here. Research issues include what type of indexing terms to use, the actual index vocabulary, the way indexes are linked to cases, possible inter-case indexes, whether indexes are direct pointers to cases or parts of an index structure, the combination of indexes during retrieval, and the assessment criteria for case similarity. A problem with too heavy reliance on indexing, however, is that indexes are a kind of pre-complied knowledge. A characteristic of case-based learning is that the generalization process is not made when learned knowledge is stored, but when this knowledge is used, i.e. during problem solving. Indexing works in the opposite direction to this, since it anticipates and pre-sets the future use of case knowledge. The alternative approach to case retrieval is search, which is time consuming, and often difficult to guide in the wanted direction. Elaborate reasoning within an extensive and rather deep model of general knowledge, is a cost demanding process. The problem, then, is to find a suitable balance between the two. An explanation-driven approach enables a search procedure which is constrained by general domain knowledge related to the context of the actual problem. We have started to study how this may allow a system to weaken its reliance on abstract indexes, in favour of making abstractions within the context of the actual problem.

Currently, the ACTIVATE-EXPLAIN-FOCUS engine uses only its general knowledge to produce explanations. A case in the CreekL representation is a rich structure, and the utilization of the cases themselves in the explanation process is a natural extension we want to look into.

References

Aamodt-90a Agnar Aamodt: A computational model of knowledge-intensive problem solving and learning. In: EKAW-90; Fourth European Knowledge Acquisition for Knowledge-Based Systems Workshop, Amsterdam, June 25,29, 1990.

Aamodt-90b Agnar Aamodt: Knowledge intensive case-based reasoning and sustained learning. In: ECAI-90; Proceedings of the ninth European Conference on Artificial Intekllligence, Stockholm, August, 1990.

Aamodt-91 Agnar Aamodt: A knowledge-intensive approach to problem solving and sustained learning. Ph.D Dissertation, University of Trondheim, Norwegian Institute of Technology, May 1991.

Althoff-91 Klaus-Dieter Althoff, Stefan Wess: Case-based knowledge acquisition, learning and problem solving for real world tasks. Proceedings EKAW-91, European Knowledge Acquisition Workshop, 1991.

Bareiss-88 Ray Bareiss: PROTOS; a unified approach to concept representation, classification and learning. Ph.D Dissertation, University of Texas at Austin, Dep. of Comp. Sci. 1988. Technical Report AI88-83.

Branting-91 Karl Branting: Exploiting the complementaryity of rules and precedents with reciprocity and fairness. In: Proceedings from the Case-Based Reasoning Workshop 1991, Washington DC, May 1991. Sponsored by DARPA. Morgan Kaufmann, 1991. pp 39-50.

Carbonell-86 Jaime Carbonell: Derivational analogy. In R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds.):*Machine Learning - An artficial Intelligence Approach,*, Vol.2, 1986. Morgan Kaufmann. pp 371-392.

Chandra-92 B. Chandrasekaran: Task-structure analysis for knowledge modeling. Communications of the ACM, Vol. 35, no. 9, September 1992 (Special issue on modeling), pp. 124-137.

Hammond-86 Kristion J. Hammond: CHEF; a model of case-based planning. Proceedings of AAAI-86. Morgan Kaufmann, 1986. pp 267-271.

Kolodner-83 Janet Kolodner: Maintaining organization in a dynamic long-term memory. Cognitive Science, Vol.7, s.243-280. 1983.

Kolodner-87 Janet Kolodner: Extending problem solver capabilities through case-based inference. *Proc. 4th workshop on Machine Learning*, UC-Irvine, June 22-25 1987. pp 167-178.

Koton-89 Phyllis Koton: Using experience in learning and problem solving. Massachusets Institute of Technology, Laboratory of Computer Science (Ph.D. diss, October 1988). MIT/LCS/TR-441. 1989.

Lenat-89 Doug Lenat, R. Guha: Building Large Knowledge-Based Systems; Representation and Inference in the CYC Project. Addison-Wesley, 1989.

Manago-93 Michel Manago, Klaus-Dieter Althoff, Ralph Traphöner: Induction and reasoning from cases. In: ECML - European Conference on Machine Learning, Workshop on Intelligent Learning Architectures. Vienna, April 1993.

Plaza-93 Enric Plaza, Josep-Lluis Arcos: Reflection, memory, and learning. Institut d'Investigacio en Intelligencia Artificial, CSIC/CEAB, Report de Recerca IIIA 93/2. Also in Proceeding from MSL-93 Workshop on Multistrategy Learning.

Porter-90 Bruce Porter, Ray Bareiss, Robert Holte: Concept learning and heuristic classification in weak theory domains. Artificial Intelligence, vol. 45, no. 1-2, September 1990. pp 229-263.

Ram-92 Ashwin Ram, Michael Cox: ... International Machine Learning Conference 1992, Workshop on integrated architectures for machine laerning and knowledge acquisition.

Richter-91 A.M. Richter, S. Weiss: Similarity, uncertainty and case-based reasoning in PATDEX. In R..S. Boyer (ed.): Automated reasoning, essays in honour of Woody Bledsoe. Kluwer, 1991, pp. 249-265.

Rissland-87 Edwina Rissland, Kevin Ashley: HYPO, a case based reasoning system. University of Massachustes, Amhearst. Dep. of Computer and Information Science. The Counseler Project, Project Memo 18. 1987.

Roberts-77 R. Roberts, I. Goldstein: *The FRL manual.* MIT AI Laboratory Memo 409, Cambridge, 1977.

Schank-89 Roger Schank, David Leake: Creativity and learning in a case-based explainer. Artificial Intelligence, Vol. 40, no 1-3, 1989. pp 353-385.

Steels-90 Luc Steels: Components of expertise. AI Magazine, 11(2), Summer 1990. pp. 29-49.

Sølvberg-92 Ingeborg Sølvberg, Inge Nordbø, Agnar Aamodt: Knowledge-based information retrieval. *Future Generation Computer Systems*, Vol.7, 1991/92, pp 379-390.

Van de Velde-92: Walter Van de Velde, Agnar Aamodt: Machine learning issues in CommonKADS. KADS-II Report, KADS-II/TII.4.3/TR/VUB/002/3.0, Free University of Brussels -VUB, 1992.

Wielinga-92 Bob Wielinga, Walter van de Velde, Guus Screiber, Hans Akkermans: Towards a unification of knowledge modelling approaches. In *Proceedings of JKAW-92, Japanese Knowledge Acquisition Workshop.* 1992

Case-Based Reasoning and Task-Specific Architectures

Dean Allemang

Swiss Federal Institute of Technology
CH-1015 Lausanne, Switzerland
allemang@lia.di.epfl.ch

Abstract. This report presents work in progress on combining case-based reasoning with task-specific architectures. The theoretical starting point of this work is the Generic Tasks of Chandrasekaran [5], while the implementation starting point is a commercially available product for building technical diagnosis systems called Diagnostic Master (tm) [1]. While the Generic Task approach eases the knowledge acquisition bottleneck by providing a task structure with which to model expertise, it does not take advantage of the fact that experts often feel comfortable telling 'war stories', or specific cases that they found interesting or instructive in the past. Typically these cases involve exceptions to well-known principles or new and unusual concepts, which may have been overlooked during knowledge acquisition.

1 Generic Tasks

Recent trends in knowledge acquisition have moved away from a knowledge mining point of view (determine *what* the expert knows) to a knowledge modeling point of view (model *how* the expert uses the knowledge). One popular style of modeling is by using *Generic Tasks* [5] as building blocks. This is not the place for a complete description of the meaning and interpretation of generic tasks, but a brief description of their features as they apply to case based reasoning is necessary.

The basic insight behind Generic Tasks it is necessary to know something about the use to which knowledge will be put in order to model it. A particular Generic Task is characterized by input/output relationships, and the type of knowledge needed to perform the task. Once a Generic Task has been identified for some expert behavior, knowledge acquisition proceeds by eliciting the particular type of knowledge needed for the task. This is in contrast with general-purpose expert system shells, in which knowledge is represented in the same fashion for all tasks, and it is left to the knowledge engineer to program the control. A half dozen particular tasks have been identified; for this paper, we are only interested in two of them, HIERARCHICAL CLASSIFICATION and ABDUCTION. The life cycle for expert system development in such an architecture is to decide, based on the input/output description of the expert task, which Generic Task is appropriate, then use information already worked out about this task (possibly in the form of automated support) to build the system. [1]

Hierarchical classification is an appropriate model when the task involves selecting a category from a pre-enumerated set of possibilities, in which the some possibilities are more general than others. In the terms outlined in the last paragraph, Hierarchical Classification is a Generic Tasks characterized by an input which is a description of some situation, the output is a particular class from a pre-enumerated set of known classes, and the knowledge that supports this classification is organized in a hierarchy of general/specific classes. Hierarchical classification is particularly useful in diagnostic settings, in which lab tests or measurements are the description of the situation, the known diseases (malfunctions) are the pre-enumerated categories, and these diseases (malfunctions) are hierarchically summarized in more general malfunction categories, which are in turn summarized into still more general categories, and so on. Knowledge acquisition for hierarchical classification proceeds by requesting categories from the expert, and a means for determining whether a particular case falls in each category. Once knowledge has been acquired in this form, any number of strategies can be used to 'run' the knowledge base. Both the knowledge acquisition phase and the run phase can be automated by a 'task-specific shell'; for acquisition this consists of an active browser for constructing the hierarchy, for running the knowledge this consists of some algorithm for searching the tree. Such a shell is intentionally less versatile than a general-purpose programming language. The test of success of a generic task and accompanying shell is whether (1) the shell is capable of solving an application problem despite its lack of flexibility, and (2) the increased guidance given by the more restrictive language makes it easier to build systems than by programming a general-purpose language.

[1] in [6] a recursive version of this life-cycle is described.

2 Case-Based Reasoning and Diagnosis

Diagnostic Master is just such a shell for Hierarchical classification. Its commercial success to date has resulted from the fact that it has succeeded on the two points above - that is, it has been successfully applied to a number of applications, and has provided sufficient guidance in system construction that it was possible for these applications to be built by users who are not trained as knowledge engineers, or even as programmers.

Golding and Rosenbloom [9] argue convincingly that the strength of a rule-based approach to expert system construction can be augmented by using a case-based method for handling exceptions. Those arguments apply equally well to generic tasks based approaches. Feret and Glasgow [8] have already applied a similar method to use case-based reasoning to enhance diagnosis performed by structural decomposition. Structural decomposition performs diagnosis on a tree similar to the hierarchical classification tree - hence these techniques should apply here as well.

We have examined two application domains to determine the applicability of the current Diagnostic Master, and the utility of these extensions. We have devised an architecture that takes advantage of the knowledge acquisition powers of DM, as well as the case extensibility of these case-based approaches.

3 Application Domains

3.1 Windows

In [11], Lauriston gives over 60 cases of problems using Windows 3.1. These cases are expressed in natural language, with features that range over functions of a dozen different software applications that run with Windows 3.1. Any particular case only refers to a few (maybe even only two) of these features. In order to process these cases, we must first determine how we will represent them.

One of the main problems in case retrieval is to determine when a new case is relevantly similar to an old case. For example, a common pattern of problem was the following:

- Some enhancement of a program is not supported by a particular platform (harware or operating system),
- The system being diagnosed uses that platform,
- The solution is to disable the enhancement.

Once one such case has been seen (say, 386E not supporting fast paste), then further similar cases (386E does not support multiprocessing features) should be identified as similar to this case. Notice that a simple match of features will not show the similarities between these cases - the multiprocessing features are used in a completely different application and context than the paste features.

Protos [3] deals with this problem by allowing for a very sophisticated match between features. Causal, structural, and functional relationships between features are recorded with each case. Protos performs a heuristic search to find a match between the features of the current case and a stored case.

But such a match is underinformed in the current situation - the fact that a particular multiprocessing feature matches fast paste in this situation is only because they are both unsupported features of the 386E. This match is relevant only if both cases involve the 386E. A single feature match, which does not also check that the platforms match, cannot possibly determine when the match is important.

The Dudu system [2] uses more information in its matching process, and is more in the spirit of EBG [12], in that the reason why a particular feature supports a particular conclusion is utilized in the matching process. We propose an architecture that will perform this sort of intentional match. In particular, rather than trying to find a simple match between the features of one case and another, we search the new case for a feature that plays the same role in the new case as played by a feature in the old. In this case, the fact that fast paste was an unsupported feature of the 386E (which is the explanation of the failure) would commence a search for an unsupported feature of the 3886E in the new case.

3.2 Textiles

In cooperation with the Swiss Federal Laboratories for Materials Testing and Research (EMPA), we have obtained a number of cases of failures in textile manufacture. The cases are unusual in that the data from the cases do not come from the machinery itself, but rather from the product, that is, the woven goods. Typical errors are dark stripes along the fabric, small holes in the fabric, and single threads under tension. The problem is to determine whether the fault comes from the spinning, weaving, or finishing process. Each of these represents a different company, who have their own experts who argue that the fault could not lie in their process. EMPA acts as an impartial agent to decide these cases.

The cases follow a pattern similar to that described in [4], in that a typical case consists of a short description of the faulty sample, followed by a number of tests performed on the sample to determine the source of the flaw. Unlike the cases examined in [4], the set of all possible tests is not known in advance. Furthermore, many of the tests are politically motivated - sometimes a certain hypothesis is pursued not because it is deemed most likely, but because it is deemed most desirable by the customer. In the long run, this does not affect the accuracy of the expert's decision (false hypotheses are ruled out, even if they are highly desirable), but it does affect the order in which tests are carried out.

Most case-based algorithms presume that the case is expressed in terms of features and values for those features. For these cases, some knowledge engineering is necessary before the case can be expressed in this form. For example, a number of the measurments are taken at intervals measured from a dark/light band. The placement of these measurements depend on the size and density of the band. Some appropriate representation of the case (e.g. in terms of 'yarn gauge deviation between dark and light bands' as opposed to simply 'yarn gauge') need to be worked out. These categories must be relevant to the task at hand, which itself must be determined. That is, in order to use the cases at all, a knowledge engineering phase supported by GTs must be carried out.

4 A combined Architecture for CBR and GT

The aim of this project is to experiment with various possibilities for combining the advantages of case-based reasoning and task-specific knowledge engineering. The architecture presented here is not final, but is intended as a workbench on which to test different ways of combining these.

The architecture follows quite simply from the two examples shown above, and the experiences found in the literature. Figure 1 shows the basic system architecture.

Since a DM tree captures general regularities in a system as provided by the expert, the overall structure of the system resembles the current DM tree. Nodes higher in the tree correspond to more general failures, nodes farther down in the tree more specific failures. A node is connected to a node higher up in the tree if it corresponds to a specialization of the concept at the higher node. This means that the tree can be searched using an establish-refine strategy, that is, establishing the relevance of the top node, and then refining this answer by establishing the relevance of the nodes beneath. This is the current state of Diagnostic Master.

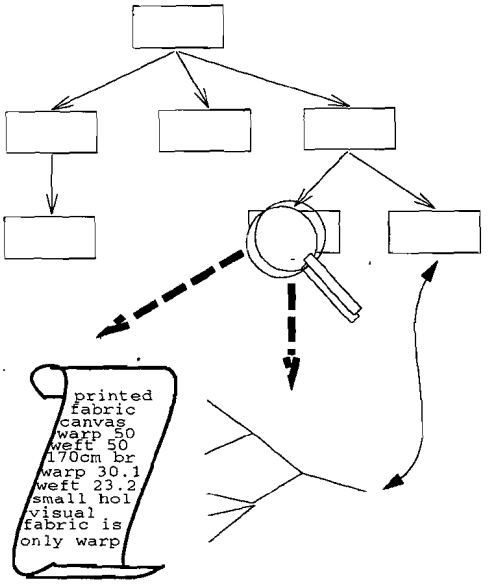


Fig. 1. Square nodes are DM concepts as in the current system. Each node corresponds to a diagnostic category, and has associated with it knowledge for establishing the validity of that category. Tip nodes correspond to diagnostic conclusions. Exceptional cases are stored in tip nodes, along with an explanation of why they should have been classified differently.

At each tip node (which, as in the case of structural decomposition [8], correspond to final diagnostic categories), the knowledge base can be extended by storing exceptional cases. In a spirit similar to Golding and Rosenbloom [9], we do not want to follow an exceptional case unless there is a compelling reason

to do so. They use inductive methods for determing how compelling a case is. We propose the use of explanation-based methods, described in more detail below.

Thus, the system construction cycle in this architecture will proceed as follows: knowledge engineering will commence as it currently does in DM. In the current version of DM, the system must be refined using the same DM architecture until its performance is satisfactory. In the new system, the features acquired by the construction of the DM case will be used to describe other cases that are particularly interesting or difficult. Rather than trying to modify the DM structure to handle all the exceptional cases, the case-based system will take over. When a particular case is misclassified, the expert explains the correct classification, and enters it in the system. In this manner, the DM tree serves as an index into the case base, or in Protos terms, as an elaborate reminding structure.

4.1 Supporting Explanations

When an exceptional case is recorded, the expert is asked for an explanation of why the case should be classified differently. The 'Tips for a teacher of Protos' [7] shows that experts are usually good at providing explanations, though the explanations they provide are not necessarily of a uniform nature. Thus it is necessary to provide the expert some assistance in structuring the explanations. This is again provided by the Generic Task structure of the problem.

Since at this point the expert is doing explanation rather than classification, the Generic Task HIERARCHICAL CLASSIFICATION is no longer the appropriate one, rather ABDUCTION. ABDUCTION is the process of inference to the best explanation. In [10], Josephson provides an inference form for abduction. That is, they provide a list of what things must be satisfied by an abductive argument in order for it to be valid: Given a set of data D to be explained, one is justified in concluding a hypothesis H as the best explanation of D when one satisfies the five following criteria:

- H explains D,
- H is *a priori* plausible,
- Sufficient alternatives to H have been considered,
- H surpasses these alternatives by a sufficient amount, and
- The data D is reliable.

This description leaves as undefined a large number of terms, like *plausible*, *explain*, *sufficient*, etc. Details of the uses of these words, and how they affect the confidence in the abductive conclusion, can be found in [10].

This form will be used to help the expert structure explanations. That is, in order to claim that something is a best explanation, it is necessary to elaborate all five of these conditions. This will both prevent the expert from making incomplete explanations, and impose a uniformity on the complete explanations.

With cases represented in this way, the question of deciding whether a new situation matches the old case is to construct an abductive justification, again using Josephson's form. The parts of the new justification will be constructed from fragments found in the previous explanations. For example, a common finding in these cases is that the fault does not occur in the raw fabric. If the leading hypothesis implies that the fault must occur in the raw fabric, perhaps we should doubt this datum. In a past case, doubt about this datum was pursued by simulating the finishing process on a sample of the raw fabric. If the fault appears, then it really was present in the raw fabric, and this datum can be safely neglected. This test can then be ordered in the present case.

Once such an abductive justification is built up, not only will we have an answer to the diagnostic problem, but an explanation of why the answer is considered correct. This is particularly important in our textile application, since the local experts will not accept a judgement without justification.

5 Advantages

One of the main advantages of case-based systems over expert systems is the reduced reliance on knowledge engineering, which has been called the bottleneck of expert system construction. Since cases are readily available, case-based reasoning alleviates this bottleneck by allowing a number of cases to replace expert effort. Unfortunately, as we have seen in the textile example above, often cases cannot be used as raw data, they need some structuring before they can be reasonably used.

A Generic Tasks approach to expert system construction also alleviates the knowledge acquisition bottleneck by providing some structure into which to fit the knowledge. The proposed architecture takes advantage of this structure in two ways:

1. As in the current DM tool, the generic task structure can be used to capture the well-known general trends in the domain, for which cases are unnecessary, and
2. a task structure can be used to determine a vocabulary for explanation of the particular cases.

6 Questions

The architecture described here is not final. There are a number of experiments we intend to carry out with our domain data.

- Perhaps one of the advantages listed in the last section will outweigh the other, for example, it could be the case that there is no reason to perform explanation-based generalization to classify the exceptional cases. In this case, the generic task structure will be used for the first phase of the classification, and to structure the cases, but not to structure an explanation of the correct classifications.
- It could also be the case that there are more exceptions than cases that follow the general trends of the domain, in which case the main problem will be to describe the case. In this case, the thrust of the system will be to use the task structure exclusively for the construction of explanations as described in section 4.1.
- Evaluate the possibility of using cases to correct faults higher in the classification tree. If a case was incorrectly classified, a systematic fault in the DM tree might be responsible. The obvious place to look for such a problem is at the most specific common generalization node of the correct classification and the misclassification.

The prototype of this system is currently under development. From the results of these (and other) experiments, we plan to integrate case based reasoning into a future release of the DM tool.

References

1. ISE Software AG. *Diagnostic Master Benutzerhandbuch.* ISE Software AG, Taegerwilen, Switzerland, 1992.
2. Dean Allemang. Using functional models in automatic debugging. *IEEE Expert*, pages 13–18, 1991.
3. Ray Bareiss. *Exemplar-Based Knowledge Acquisition.* Academic Press, Inc., Boston, 1989.
4. Ralph Barletta and William Mark. Explanation-based indexing of cases. In *Proceedings of the national conference on Artificial Intelligence*, pages 541–546. AAAI Press, 1988.
5. B. Chandrasekaran. Generic tasks in knowledge-based reasoning. *IEEE Expert*, 1(3):23–30, 1986.
6. B. Chandrasekaran. Design problem solving: A task analysis. *AI Magazine*, 11(4):59–71, 1990.
7. Dan Dvorak. Cl-protos users manual. included with CL-Protos progam, 1991.
8. M.P. Feret and J. I. Glasgow. Case-based reasoning in model-based diagnosis. In D. E. Grierson, G. Rzevski, and R. A. Adey, editors, *Applications of Artificial Intelligence VII*, pages 679–692. Computational Mechanics Publications, 1992.
9. Andrew Golding and Paul Rosenbloom. Improving rule-based systems through case-based reasoning. In *Proceedings of the 9th national conference on Artificial Intelligence*, pages 22–27. AAAI Press, 1991.
10. John Josephson and Susan Josephson. *Abductive Inference, Computation, Philosophy, Technology.* Cambridge University Press, Cambridge, 1994.
11. Robert Lauriston. Troubleshooting windows 3.1. *PC World*, pages 128–145, 1992.
12. Tom Mitchell, Richard Keller, and Smadar Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.

Case-based Reasoning at the Knowledge Level: An Analysis of CHEF

Eva Armengol Enric Plaza

Institut d'Investigació en Intel·ligència Artificial , C.S.I.C.
Camí de Santa Bàrbara, 17300 Blanes, Catalunya, Spain.
{plaza | eva}@ceab.es

Abstract. We analyze the CHEF system at the knowledge level following the Commet methodology. This methodology has been used for the design and construction of KBS applications. We have applied it to analyze the learning methods of existing systems at the knowledge level. We claim that this sort of analysis can be a first step to the integration of different learning methods.

1 Introduction

We offer a succinct analysis of CHEF system so as to show a concrete example of knowledge level analysis applied to a learning method. In [3] there are the analysis and comparison of CHEF, PROTOS and CASEY and the resulting common task structure from them. We think conceptual frameworks like KADS [9] or Commet [8] that redefine the notion of "knowledge level" from the original Newell's definition [6] may prove to be more interesting to Machine Learning theory than using the Newell's knowledge level in ML methods analysis was done by Dietterich [4]. Furthermore, having a uniform description of KBS, knowledge acquisition, and learning processes can be very useful for achieving an understanding of the issues involved in their integration in knowledge engineering. A previous conceptual study on CBR systems was made by Aamodt [1]. Aamodt decomposes CBR systems in tasks both the reasoning and the learning processes. Our study is similar but it is more formal in the sense that we try to find a common formalization for all the tasks and to analyze the knowledge used by each one. We have already made a similar study for EBL methods [2] obtaining a common structure of tasks, and models having the same conceptual information (although they have different implementations). With the study of CBR systems we try to complete the analysis of learning methods at the knowledge level.

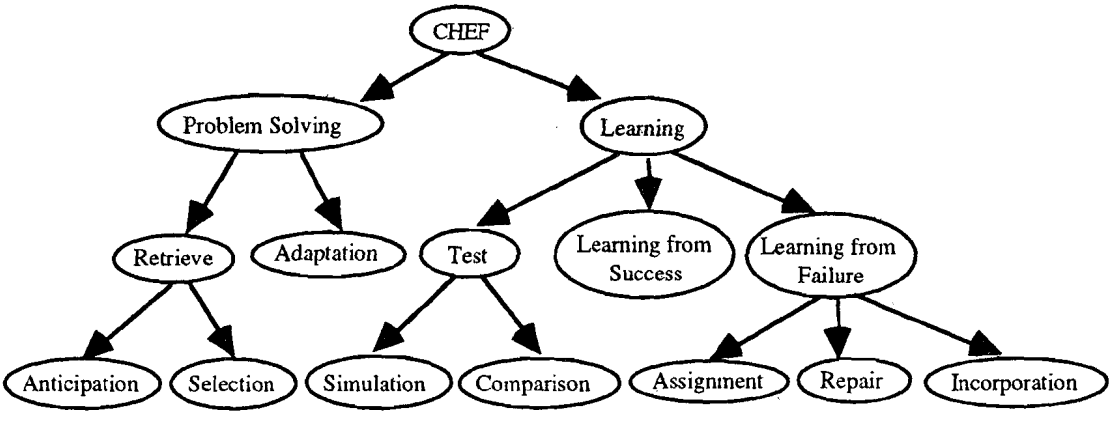


Fig. 1. Task Structure of the CHEF system at the knowledge Level

2 CHEF at the Knowledge Level

The analysis of CHEF at the knowledge level is made following the Commet methodology [8]. Commet ontology is composed of tasks, models and methods. A task is a set of goals that have to be solved. A model contains the relevant knowledge to achieve some task goals. A method is a procedure organizing and executing the activities of the construction models. According to this methodology CHEF has produced the structure of tasks in the fig. 1. In the following we analyze some of the tasks emphasizing their goal, the input and output models and the method used to solve each task. The complete analysis can be found in [3].

CHEF-Task. This task is solved using the CHEF-Method that decomposes it in two subtasks: Problem-Solving-Task and Learning-Task. Problem-Solving-Task solves the problem and proposes a plan. Learning-Task executes the plan and learns from plan failure or success. Input models are Goals, Memory of Plans and Memory of Failures. Goals model contains the set of goals that have to achieve the plan. Memory of Plans model contains a discrimination network containing all the plans that the system has produced until now indexed by predictive features. The output models are New Memory of Plans that is the memory of the existing plans to which the new plan has been added; and New Memory of Failures containing the new failure (if any has been produced). In the fig. 2 can be viewed the representation of the CHEF-Task at the knowledge level, the decomposition produced by the CHEF-Method and the models that it uses.

CHEF-Task
Goal: To improve the system behaviour.
Input: Goals, Memory of Plans and Memory of Failures models.
Output: A New Memory of Plans and New Memory of Failures model.
Method: CHEF-Method.

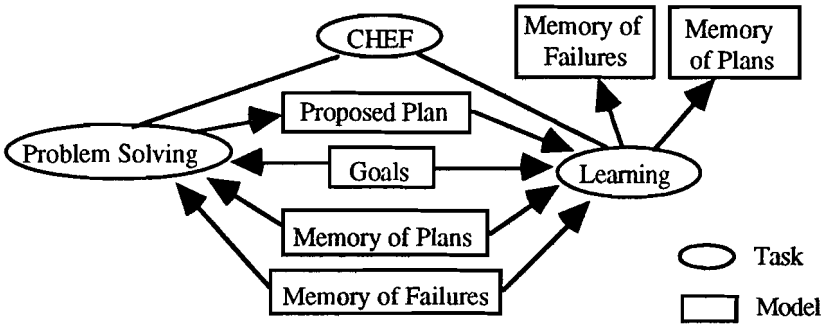


Fig. 2. Representation at the Knowledge Level, Task Structure and Models of the CHEF-Task

Problem-Solving-Task. The goal of this task is to propose a plan achieving all the goals in Goals model and without any predictable problem. CH-Problem-Solving-Method (fig. 3) is a task decomposition method that decomposes Problem-solving-Task in two subtasks: Retrieve-Task and Adaptation-Task. Retrieve-Task retrieves a past plan achieving a subset of goals contained into the Goals model. Adaptation-Task modifies the retrieved plan in order to achieve all the desired goals. Input models are those of the CHEF-Task. Output model, Proposed Plan, contains a plan achieving all the input goals.

Problem-Solving-Task
Goal: To obtain a plan achieving all the goals in Goals model.
Input: Goals, Memory of Plans and Memory of Failures models.
Output: A Proposed Plan model.
Method: CH-Problem-Solving-Method.

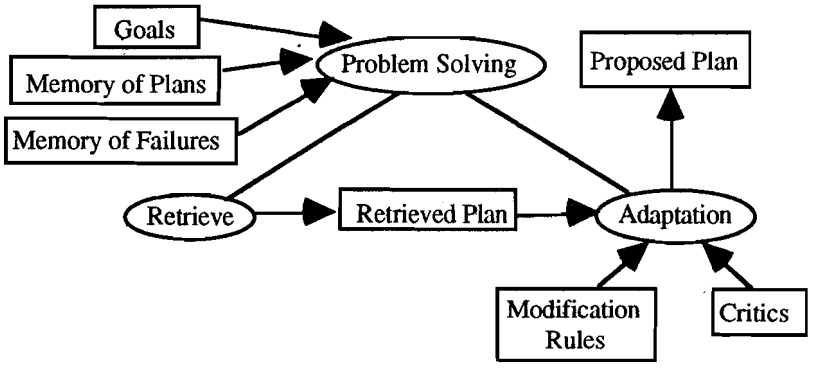


Fig. 3. Knowledge Level Representation, Task Structure and models of the Problem-Solving-Task

Retrieve-Task. The goal of this task is to retrieve a plan achieving the maximum number of important goals and which has not any predictable failure. CH-Retrieve-Method is a task decomposition method that decomposes Retrieve-Task in two subtasks: Anticipation-Task and Selection-Task. Anticipation-Task analyzes the planning goals and the situation in which they are evolved and decide if there are some states or goals predicting some problem. For the sake of simplicity it is not be explained here. Selection-Task selects a plan avoiding the

predicted problems and achieving the maximum number of goals in the Goals model. Input models are the same of the Problem-Solving-Task. Output model, Retrieved Plan, contains a plan that both avoids all the anticipated problems and achieves input goals that are considered important. Important goals are those goals that are either difficult to achieve or difficult to incorporate to a plan. In the fig. 4 can be viewed the knowledge level representation of the Retrieve-Task.

Retrieve-Task
Goal: To obtain a plan without any predictable problems achieving a set of goals in Goals model.
Input: Goals, Memory of Plans and Memory of Failures models.
Output: A Retrieved Plan model.
Method: CH-Retrieve-Method.

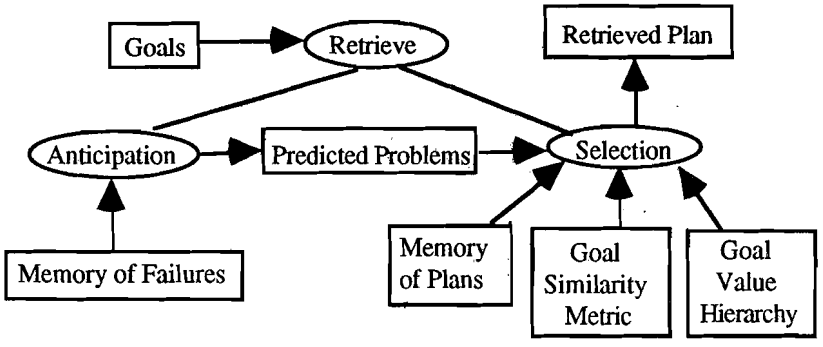


Fig. 4. Knowledge Level Representation, Task Structure and Models of the Retrieve-Task

Selection-Task. This task has to retrieve the best past plan avoiding all the predicted problems and achieving a set of input goals. Ideally it would be desirable that the retrieved plan achieves all the goals but in general this is not possible. The usual situation is to have several plans achieving a subset of goals (different for each plan). It is necessary to have a metric evaluating the similarity of the goals and a hierarchy of goals evaluating the relative utility of each plan in respect to a set of goals. The goals considered as more important are those that are difficult either to achieve or to incorporate into a plan. Input models are Goals, Predicted Problems, Memory of Plans, Goal Value Hierarchy and Goal Similarity Metric. Goal Value Hierarchy model contains the *is-a* hierarchy which nodes are sets of goals considered as similar. This hierarchy obtains the importance that a plan achieves certain goals. Goal Similarity Metric model contains an abstraction hierarchy allowing know if a plan satisfies partially a goal. The output model, Retrieved Plan, is a plan that both avoids all the predicted problems Anticipation-Task and achieves most of the Goals model. CH-Select-Method uses the Predicted Problems model, the goals and the abstractions of goals as indexes to access to the memory of plans. Then it uses the importance of the different goals to decide the choose between different competitor plans. It has to find a unique plan satisfying the maximum number of goals into Goals model taking account that both there are goals that are more important than others and some goals cannot be satisfied directly but only partially. In this case it must found a past plan satisfying similar goals and to modify it to obtain a unique plan achieving exactly the goals contained in Goals model. The representation of the Selection-Task at the knowledge level is the following:

Selection-Task
Goal: To obtain a plan achieving a subset of goals.
Input: Goals, Predicted Problems, Memory of Plans, Goal Value Hierarchy and Goal Similarity Metric models.
Output: A Retrieved Plan model.
Method: CH-Selection-Method.

Adaptation-Task. The goal of this task is to modify the plan contained into Retrieved Plan model in order to achieve all the goals into Goals model. To do this adaptation CHEF uses a set of modification rules and a set of critics containing specific information about the domain. Input models are Retrieved Plan, Modification Rules and Critics. Modification Rules model contains a library of rules that are steps that can be added to particular plans to achieve particular goals. Critics model contains knowledge about domain objects and general plans. The output model, Proposed Plan, contains the best plan that the system has found to achieve the Goals model. The representation of Adaptation-Task at the knowledge level is the following:

Adaptation-Task
Goal: To propose a plan achieving the input goals.
Input: Retrieved Plan, Modification Rules and Critics models.
Output: A Proposed Plan model.
Method: CH-Adaptation-Method.

Learning-Task. Learning-Task has to check if the proposed plan works as it is desired. If it is correct only will be necessary to incorporate the plan indexing it by the goals and the predicted failures. Otherwise it has to repair the plan. CH-Learning-Method is a task decomposition method that decomposes Learning-Task in three subtasks: Test-Task, Learning-from-Success-Task and Learning-from-Failure-Task. Test-Task checks if the proposed plan produces the desired result and activates Learning-from-Success-Task or Learning-from-Failure-Task according to the result. Input models are Goals and Predicted Problems from Anticipation-Task, Proposed Plan, Memory of Plans and Memory of Failures. The output models are those of the CHEF-Task, that is to say, New Memory of Plans and New Memory of Failures.

Learning-Task
Goal: To incorporate new knowledge (plans and failures) to the system in order to improve its behaviour.
Input: Goals, Predicted Problems, Proposed Plan, Memory of Plans and Memory of Failures models.
Output: A New Memory of Plans and a New Memory of Failures model.
Method: CH-Learning-Method.

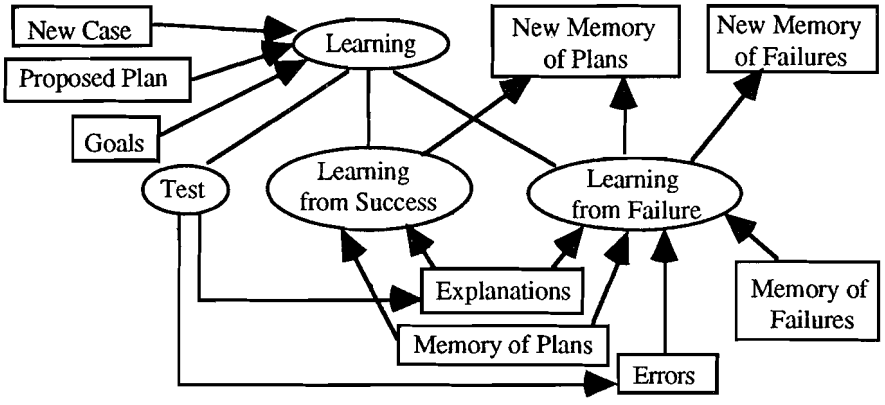


Fig. 5. Knowledge Level Representation, Task Structure and Models of the Learning-Task

Test-Task. To check if the proposed plan achieves the goals into Goals model, Test-Task makes a simulation and then the outcome is compared to the desired one. Test-Task uses as input models Proposed Plan and Inference Rules. Inference Rules model contains inference rules indicating the effects of each step in the domain of each object. Output models are Errors and Explanations models. Errors model contains which is the problem that has been occurred describing the state that defines the failure, the step where the failure is produced and the predictive conditions of the failure. Explanations model contains the state defining the failure, the step where the failure is produced and the conditions that have to be accomplished to produce the failure. Thus CH-Test-Method is a task decomposition method decomposing Test-Task in two subtasks: Simulation-Task and Analysis-Task. Simulation-Task simulates a plan execution. Analysis-Task compares the obtained states and the desired ones and allows to obtain the produced errors. This method requires strong introspective capabilities and a theory for deciding blame assignment to the system parts and decisions. The representation at the knowledge level ant the decomposition of Test-Task produced by CH-Test-Method is that of the fig. 6.

Test-Task
Goal: To check if the plan accomplishes the input goals.
Input: Proposed Plan and Inference Rules models.
Output: An Explanations and the Errors models.
Method: CH-Test-Method.

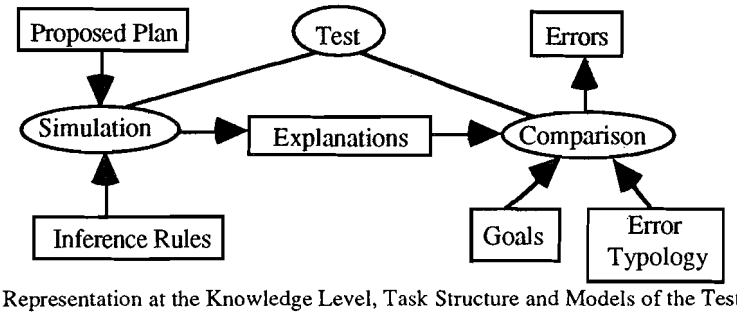


Fig. 6. Representation at the Knowledge Level, Task Structure and Models of the Test-Task

Learning-from-Success-Task. If the plan has worked as it is desired, this task is activated. Learning-from-Success-Task incorporate to the memory of plans the new plan indexing it by the goals and failures that it avoids ordered by order of importance. To index the plan are also used the generalizations of the goals. A plan is not generalized because it can be used in several situations. Only the goals are generalized. The explanation serves to generalize goals. Learning-from-Success-Task has as input models Proposed Plan, Explanations, Goals, Memory of Plans. The output model is a New Memory of Plans. The representation of Learning-from-Success-Task at the knowledge level is the following:

Learning-from-Success-Task
Goal: To improve the system behaviour by incorporation of the new plan.
Input: Proposed Plan, Explanations, Goals and Memory of Plans models.
Output: A New Memory of Plans model.
Method: CH-Learning-from-Success-Method.

Learning-from-Failure-Task. This task is activated when a failure is detected in the proposed plan. Learning-from-Failure-Task have to repair a plan and incorporate it to the memory of plans. It must also to incorporate to the memory of failures all those features predicting the failure. The used models are the Proposed Plan, Explanations, Errors, Goals, Memory of Plans and Memory of Failures as input models. The output models are a New Memory of Plans and a New Memory of Failures that incorporate the new case and the produced failure. In the fig. 7 can be viewed the models used by Learning-from-Failure-Task and how it can be decomposed.

Learning-from-Failure-Task
Goal: To improve the system behaviour by incorporation of the new plan and the predictive features of the produced failure.
Input: Proposed Plan, Explanations, Errors, Goals, Memory of Plans, Memory of Failures models.
Output: A New Memory of Plans and a New Memory of Failures models.
Method: CH-Learning-from-Success-Method.

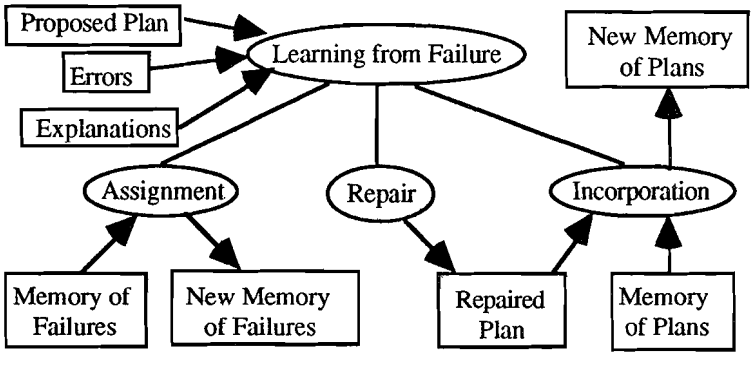


Fig. 7. Knowledge Level Representation, Task Structure and Models of the Learning-from-Failure-Task

3 Conclusions

Knowledge level analysis (KLA) permits to make explicit the relation of learning with problem solving. We take a unified approach for inference in learning and problem solving and we propose that KLA may be useful tool for understanding learning, problem solving and their relationship in architectures that integrate learning and problem solving. Using KBS conceptual frameworks for describing both is useful conceptually, as we have tried to show here, but may be very fruitful also at the practical level of building KBS applications.
Our last goal is to have a representation of different learning methods in order to represent them into a cognitive architecture. So, as well as CHEF, we have analyzed at the knowledge level PROTOS and CASEY systems [3]. From this analysis we have obtained a common task structure for them. A similar analysis about EBL methods [2] produces a structure containing knowledge level tasks that are common to the most representative EBL systems. We claim that from the obtained knowledge level common task structure (obtained applying the Commet methodology) and an adequate characterization of the models that they use, the more appropriate methods to solve a task can be chosen. This can be a first step for the learning methods integration.

References

1. A. Aamodt: A Knowledge-intensive, integrated approach to problem solving and sustained learning. Ph. D. Dissertation. University of Trondheim (1991)

2. E. Armengol, E. Plaza: Elements of Explanation-based Learning. Research Repport IIIA 93/9 (1993)

3. E. Armengol, E. Plaza: Analyzing Case-based Reasoning at the Knowledge Level. Research Report IIIA 93/14 (1993)

4. T.G. Diettrich: Learning at the knowledge level. Machine Learning 3, 287-354 (1986)

5. K.J. Hammond: Case-Base Planning. Viewing Planning as a Memory Task. Perspectives in Artificial Intelligence. Volume 1. Academic Press, Inc. 1989.

6. A. Newell: The Knowledge Level. Artificial Intelligence 18, 87-127 (1982)

7. E. Plaza, J.L. Arcos: Reflection and Analogy in Memory-based Learning, Proc. Multistrategy Learning Workshop.

8. L. Steels: Reusability and configuration of applications by non-programmers. VUB AI-Lab Research Report (1992)

9. B. Wielinga, A. Schreiber, J. Breuker: KADS: A modelling approach to knowledge engineering. Knowledge Acquisition 4(1) (1992)

Integration of Case-based Reasoning and Inductive Learning Methods

Stefan K. Bamberger, Klaus Goos

Universität Würzburg
Lehrstuhl für Informatik VI
Allesgrundweg 12
97218 Gerbrunn
bambi/goos@informatik.uni-wuerzburg.de

Abstract. In cased based reasoning the preselection of interesting cases to get an efficient case comparison is most important. In large case-bases the search for cases will be very expensive especially by using external data bases. Looking for another way to handle the preselection, we examine several inductive learning methods which generate heuristic rule knowledge out of a given case-base. Those rules can be interpreted very efficiently during the reasoning process. In this paper we compare three inductive learning approaches (BUBE, ID3/C4 and UNIMEM) for the usage with our existing case-based reasoning tool CcC+ to improve the overall performance. Finally, we propose a possible integration between CcC+ and the best one of the comparison.

1 Introduction

Cases are the fundament of many problem solving approaches. Two main categories may be distinguished:

I. Cases are used directly for problem solving.
II. Cases are pre-processed to generate the problem solving knowledge later used .

Members of the first category have the advantage that the problem solving process can start immediately after a case-base had been acquired. Furthermore newly acquired cases can be directly integrated into the inference process. With the case-base growing, however, the time for problem solving exceeds, depending on high search and retrieval costs. In contrary, the second category first needs much more time to generate the real problem solving knowledge, e.g. decision trees, heuristic rules, etc. The advantage of those methods is the generation of a knowledge structure which can be used very efficiently during the inference process.

In this paper we outline representatives of both categories and show why it will be successful to combine them to gain a better performance.

2 Different Approaches

All CBR-tools like CcC+ [Puppe&Goos91] and PATDEX [Wess93] belong to the first category. The characteristic features are:

- The basis of the problem solving process are cases stored in a database and additional knowledge about similarity between symptom-values, abnormalities of symptom-values and dynamic weighting of symptoms.
- The problem solving process consists of two steps. The first step is a preselection of possible candidates to reduce the search space. The second step is the comparison of every candidate with the search case to compute the most similar one.
- Both an adaptation of the similarity measure and an adaptation of the found solution with extra knowledge is possible to improve the problem solving process.

Approaches of the second category are inductive learning methods, like BUBE [Bamberger92, 93], ID3/C4 [Quinlan86, 90], and UNIMEM [Lebowitz90]. The features are:

ad BUBE:

- The aim of BUBE is to generate a heuristic knowledge base. Therefore BUBE uses causal

knowledge about the object dependencies to generate the rule structure. The necessary rule evidences are gained by a statistical evaluation of the case-base. To improve the quality of the generated rules an automatic heuristic adaptation is possible.
- To solve a case, the heuristic problem solver MED2 [Puppe et al. 92] is used to interpret the derived rules.

ad ID3, C4 (empirical learning):
- Both methods work only on the base of cases without any additional knowledge.
- Before the problem solving process can start these approaches generate a decision tree. ID3 is limited to discrete values in the domain. C4 is the successor of ID3 and allows numerical values. The numerical values are handled by using a threshold to divide the range into two parts. The leafs of a tree are the learned solution classes.
- The task of the problem solving process is to find a path in the tree which corresponds to the symptom values in the current case.

ad UNIMEM (similarity based learning):
- UNIMEM uses cases and additional knowledge about similarities between symptom values.
- The goal is to take similar cases and abstract them to form a hierarchy of generalisations that will be used to organize the case-base. The generation of the concept tree is expensive because of the incremental treatment of the cases. Therefore it may often be necessary to update the tree structure.
- The problem solving process uses the hierarchical organisation of the concept tree to classify the search case.

3 Comparison

To determine the performance of the different methods we compared them in seven points. The result of this analysis is shown in tabular 1.

The tabular shows that no approach is perfect, but the different approaches have different strengths and weaknesses. An integration of two or more approaches is advisable to minimize the costs and optimize the performance of inference. CcC+ seems to be the most powerful approach. The great advantage of explainability, competence assessment and incremental extension of the case-base is only diminished by the costs of the preselection during the inference process. UNIMEM, which seems to be very similar to CcC+ at first sight, causes more costs when altering the case-base and has a weak competence assessment. Therefore an integration of CcC+ and UNIMEM does not seem to be advisable, because a performance increase of the resulting system cannot be expected.

The ID3 system cannot handle all kinds of information such as numerical or unknown values. Because such values are used in CcC+, ID3 is not a powerful enough partner for linking the two. The problems of representation are overcome with C4, the successor of ID3. The quality of the derived decision tree is dependent on the correctness of the cases. In addition to C4, BUBE uses causal knowledge to deal with the effect that statistical evaluation of cases could make believe object correlations which are not real. Normally the cases have different relevance and quality. Using additional causal knowledge might be very helpful to remove these uncertainties respecting the object correlations. A further advantage of BUBE is that the structure of causal knowledge can also be used in the learning step to structure the heuristic knowledge. Because the domain expert already knows this structure the rules produced by BUBE can be better understood than the decision tree completely automatically produced by ID3/C4. Since BUBE is the only system with this capabilities an integration of CcC+ and BUBE promises the greatest success.

	Case-based	Inductive Learning		
	CcC+	BUBE	ID3/C4	UNIMEM
Costs with stable and large case-bases	The time for the pre-selection grows linear with the size of the case-base. Using K-D-trees to pre-select could reduce the costs to log(#cases). The detailed similarity computation per case is linear to the number of symptoms in the case.	Generation of the rules is linear to the case-base. The additional causal knowledge allows to focus on the really relevant symptoms. Duration of the inference process is linear to the given symptoms.	The generation of the decision tree depends on the number of symptoms. High costs to calculate the entropy of the symptoms. No possibility to check for bogus causalities. The effort of the inference process is proportional to the tree depth.	The generation of the decision tree depends on the number of symptoms. The effort of the inference process is proportional to the tree depth.
Costs with altering large case-bases.	No additional costs. Improves directly the problem solving process.	Recompilation of the complete case-base after each alteration.	Recompilation of the complete case-base after each alteration.	Additional costs to insert the new case in the existing concept hierarchy.
Problem solving with incomplete cases (some symptoms are not known or not asked)	A solution is possible with a ranking to decide the usefulness.	There is no knowledge about the quality of the solving proposal. The computed result can be too rough, lacking of enough datas to further specialize it.	In ID3 the decision of the possible branch is very difficult. C4 uses the probabilistic valuation to come to a decision.	Missing values in an actual case are no problem being every node a possible solution.
Background knowledge	Detailed knowledge of the similarity measure and symptom importances.	Only causal relations.	No.	Detailed knowledge of the similarity measure.
Background knowledge acquisition	Similarity measure may be hard to formulate.	Acquiring causal dependencies is quite straightforward	No.	Similarity measure may be hard to formulate.
Competence assessment	Well, with relation to the maximal similarity	Weak.	Weak.	Weak.
Explainability	Excellent. Direct access to the pre-selected cases and their similarities is possible.	The possibilities of the heuristic explanation are limited. No backtracking to the underlying case-base is possible.	The possibilities of the explanation are limited. No backtracking to the underlying case-base is possible.	By the path through the concept hierarchy. The training cases are available for further explanations.

Tab. 1: Comparison of representatives of the two categories.

4 Integration

The main effect of an integration should be to optimize the costly preselection in CcC+. This can be achieved by using the generated heuristic rules of BUBE to derive a set of possible solutions. The cases corresponding to this solution set are used by CcC+ for a detailed similarity computation. Other preselection techniques, e.g. K-D-trees, have been proposed in the literature [Friedmann77]. They offer an efficient indexed search, but cannot be used with a dynamic similarity measure which may be specified in CcC+.

For a successful integration a common representation und communication level is the prior condition. The representation level is already realized by a so-called base terminology. This common base terminology is defined in the environment, where the case-based, heuristic and causal problem solvers are embedded [D3]. Each kind of knowledge consists of two parts: the structure of the domain knowledge expressed in the base terminology completed with the problem solver dependent knowledge (Fig. 1). The layer architecture supports the consistency and the exchange of datas as each component works on the same central base terminology.

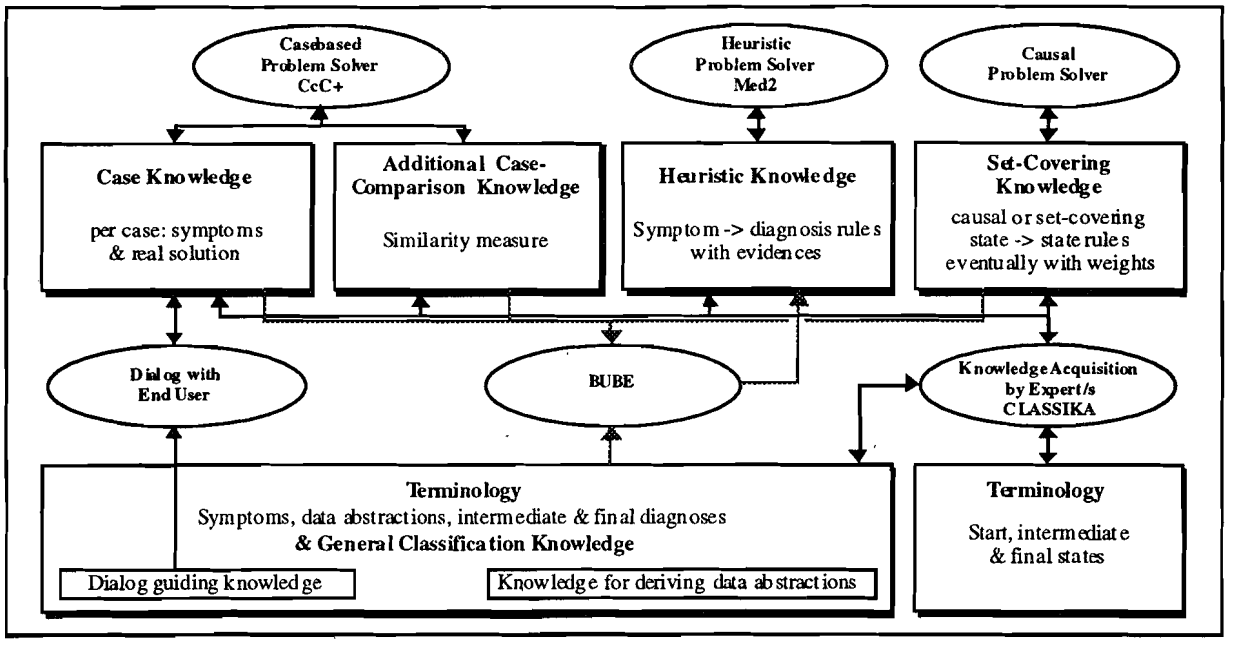


Fig. 1: Integrated knowledge representation. The knowledge of the single methods is divided into general, mainly terminological classification knowledge (lower part of the figure) and into knowledge specific one for the method (upper part). Cased based and heuristical knowledge use the same terminology, while set covering classification knowledge is based on its own terminology. Case data can not only be supplied by the expert, but is collected by the end users as well even in a larger number. The base terminology and the problem solver specific knowledge is entered using the graphical knowledge acquisition workbench CLASSIKA. BUBE uses the base terminology together with the specific knowledge of the set-covering and case-based problem solver to generate heuristic knowledge.

Connecting different modules on the communication level means that every component defines an interface making an external synchronisation possible. The idea is that a problem solving process will be influenced by messages about partial results of other problem solvers. These messages are used to control and to improve its own problem solving process. That means for our special module CcC+ and the heuristic problem solver MED2 that the preselection mechanism of CcC+ is supported by the possible partial results of MED2.

In the following we show a way to support CcC+´ preselection using heuristic rules. The cases used by CcC+ and BUBE consist of a tupel of symptoms with values and a set of intermediate and final diagnoses as the result of a case.

The way CcC+ manages its preselection is to look for a choice of symptoms out of the given search case. All cases of the case-base satisfying the choice are taken into the preselection set. The choice criteria is defined, so that the preselection set covers all relevant cases. Additionally, an arbitrary number of irrelevant cases might be present in the set. The reason is, that, in spite of a possible dynamic weighting, the examination of the symptoms by themselves is not case specific enough. The goal is to find a stronger criteria to minimize the number of irrelevant cases.

The heuristic problem solver MED2 also starts with a set of symptoms and derives intermediate and/or final diagnoses with efficient heuristic rules. To derive an intermediate diagnoses one or more symptoms are necessary. So, the intermediate diagnosis reflects a combination of symptoms, which is more solution specific than the raw symptom for itself. That's the reason why those diagnoses are used in MED2 to reduce the search space for requesting additional symptoms.

The preselection of CBR can profit from the same effect. Using the intermediate diagnoses as the choice criteria, a better focus on the relevant cases is possible. Since the number of the intermediate diagnoses are much lower than the former number of raw symptoms, the costs for the preselection is automatically reduced. Besides, the resulting lower number of cases in the

preselection set also reduces the costs for the following exact comparison. Thus, the above mentioned goal is reached.

To realise the new preselection mechanism, CcC+ has to be synchronized with the inference process of MED2. To do this, CcC+ asks MED2 for the actually derived intermediate diagnoses. If there exist any, they are used, otherwise the conventional mechanism with the raw symptoms is used. First tests showed a performance improvement from 10 to 30 percent. The more raw symptoms are neccessary and/or collected in a saved case, the greater is the gain using intermediate diagnoses.

5 Conclusion

In this paper we presented a way to integrate a case-based reasoning and inductive learning system. With the comparison of three different kind of inductive learning systems (BUBE, ID3/C4 and UNIMEM). The result is, that each of those systems has different advantages and disadvantages. Overall, BUBE takes advantage of different kind of knowledge and is therefore the most flexible approach for an integration with the CBR-tool CcC+. The synergy effect is, that the response time of the inference process is minimized while the full quality of explanation and competence assessment is kept. Both CcC+ and BUBE are completely implemented. A prototypically integration on the communication level (the representation level is fully supported) is realized. A first evaluation showed that the cost reduction may reach up to 30 percent without loosing the quality of the result. Currently we are about to evaluate the integration in a larger study.

References

[D3] (1991). Bamberger, S., Gappa, U., Goos, K., Meinl., A., Poeck, K., & Puppe, F.: Die Diagnostik-Expertensystem-Shell D3. Handbuch. Institut für Logik, Komplexität und Deduktionssysteme der Universität Karlsruhe.

Bamberger, S. (1992). Teilautomatischer Wissenserwerb für die heuristische Klassifikation auf Basis von Fällen und Fehlermodellen. Diplomarbeit am Institut für Logik, Komplexität und Deduktionssysteme der Universität Karlsruhe.

Bamberger, S., Gappa, U., Goos, K., Poeck, K. (1993). Teilautomatische Wissenstransformation zur Unterstützung der Wissensakquisition. 2. Deutsche Tagung Expertensystem (XPS-93), Puppe, F., Günter, A., (Hrsg.), 153-166, Springer.

Friedmann, J.H., Bentley, J.L., Finkel, R.A. (1977). An Algorithm for finding best matches in logarithmic expected time. ACM Trans. math. Software, 3:209-226.

Lebowitz, M. (1990). The utility of similarity-based learning in a world needing explanation. in Michalsky (ed.): Machine Learning, An Artificial Intelligence Approach III, Morgan Kaufmann 1990.

Puppe, F., & Goos, K. (1991). Improving case-based classification with expert knowledge. 15. Fachtagung für Künstliche Intelligenz, GWAI-91, Christaller, Th. (Ed.), 196-205, Informatik-Fachberichte 285, Springer.

Puppe, F., Legleitner, T., and Huber, K. (1991): DAX / MED2 - A Diagnostic Expert System for Quality Assurance of an Automatic Transmission Control Unit, in Zarri, G. (ed.): Operational Expert Systems in Europe, Pergamon Press, 1991.

Quinlan, J.R. (1986). Induction of Decision Tree. Machine Learning, Vol. 1, Nr. 1, 81–106, Kluwer Academy Publisher 1986.

Quinlan, J.R. (1990). Probabilistic Decision Trees. in Michalsky (ed.): Machine Learning, An Artificial Intelligence Approach III, Morgan Kaufmann 1990.

Wess, S. (1993). PATDEX – ein Ansatz zur wissensbasierten und inkrementellen Verbesserung von Ähnlichkeitsbewertungen in der fallbasierten Diagnostik. 2. Deutsche Tagung Expertensystem (XPS-93), Puppe, F., Günter, A., (Hrsg.), 42-55, Springer.

Explanation-based Similarity for Case Retrieval and Adaptation and it's Application to Diagnosis and Planning Tasks

Ralph Bergmann and Gerd Pews
University of Kaiserslautern
Dept. of Computer Science
P.O. Box 3049
67653 Kaiserslautern, Germany
E-Mail: {bergmann,pews}@informatik.uni-kl.de

Abstract

Case-based problem solving can be significantly improved by applying domain knowledge (not problem solving knowledge) which can be acquired with reasonable effort to derive explanations of the cases' correctness. Such explanations, constructed on several levels of abstraction, can be employed as the basis for similarity assessment as well as for adaptation by solution refinement. This paper presents a general approach for explanation-based similarity and exemplarily shows it's application for a diagnosis and a planning task.

1 Introduction and Motivation

The underlying principle of case-based reasoning is the idea to remember solutions to already known problems for their reuse during novel problem solving. The case which is most similar to the current problem is retrieved from a case base and it's solution is modified to become a solution to the current problem. One of the aspired benefits of case-based reasoning is to reduce the need to acquire and explicitly represent general knowledge of the problem domain and thereby to overcome the knowledge acquisition bottleneck. Due to the avoidance of explicit domain knowledge, the similarity between two cases is mostly assessed by a numeric computation of selected surface features of the problem description and results in a single number which reflects all aspects of the similarity. All knowledge about problem similarity is implicitly encoded into a formula which defines a similarity measure. Besides this, less attention was paid on *case adaptation* since this would also require a large amount of knowledge – it was even argued that for classification and diagnostic tasks case adaptation is not necessary.

We want to argue against the idea to avoid all kinds of explicitly represented knowledge. Knowledge which can be acquired with reasonable effort should be used for similarity assessments as well as for solution adaptation. Such additional knowledge is required for planning as well as for diagnosis tasks, in order to achieve more powerful and domain-justified case-based problem solvers. From the current experience in knowledge acquisition for "traditional" knowledge based systems, we can at least distinguish two different types of knowledge [Newell, 1982]: Domain knowledge and problem solving knowledge [Wielinga *et al.*, 1992]. Problem solving knowledge describes the *process* of problem solving in terms of steps (i.e. basic inferences or subtasks as in KADS) that should to be performed to (efficiently) derive a problem's solution. This kind of knowledge is the central target of KADS-like modeling approaches which reflects the high difficulty in it's acquisition. Unlike problem solving knowledge, domain knowledge is the knowledge about the "components" that are available to construct a problem solution and their interaction within the solution to a problem. Domain knowledge is therefore sufficient to determine whether a proposed solution really solves a given problem. This kind of knowledge is much easier to acquire than problem solving knowledge, especially in technical domains. Specific domain tailored knowledge acquisition tools have already been built to support the elicitation and formalization of such domain knowledge (e.g. [Musen *et al.*, 1987; Schmidt and Zickwolff, 1992]). Using this knowledge, an explanation of a case can be constructed. On several levels of abstraction, this explanation shows how the case's solution solves the case's problem. Based on such explanations, the similarity between two cases can be assessed and the adaptation process can also be focused to the relevant portions of the solution.

In the rest of this paper we will describe the approach in more detail and show it's application for a diagnosis and a planning task, which are both prototypically implemented.

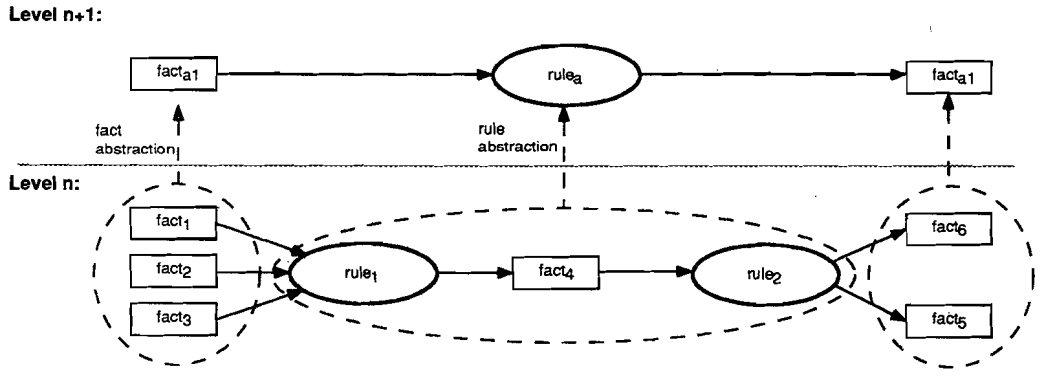


Figure 1: Multi-level explanation structure

2 Explanation-based Similarity for Diagnosis and Planning

The core idea of our approach is to use domain knowledge – which can be acquired relatively easy – to set the similarity assessment and the adaptation on a more profound basis founded on an *explanation* of a case. Such an explanation does not describe how a solution is derived (this would require problem solving knowledge) but that a solutions really solves a given problem, i.e. an explanation of the correctness of the solution. It is really important to keep this distinction in mind since it make our approach different from derivational analogy [Carbonell and Veloso, 1988]. On the other hand, the domain knowledge we employ for explaining a case is stronger than just causal relations like in other approaches [Barletta and Mark, 1988; Koton, 1988] and may be best compared to the knowledge employed at the object level of CREEK [Aamodt, 1991].

The similarity of two cases can be judged according to the similarity of their explanations. This requires that a knowledge base containing relevant domain knowledge is built on several levels of abstraction. An explanation on a lower level of abstraction is a more specific explanation and is consequently composed of a larger number of specific rules than an explanation on a higher level of abstraction. Therefore, the explanations of two cases can differ very much on a lower level of abstraction but will be identical on a higher level of abstraction. This observation leads us to a rating of the similarity of two explanations: The lower the level of abstraction is on which two explanations are identical, the higher is the assessment of their similarity.

2.1 Representation of Explanations and Similarity assessment

In diagnosis as well as in planning, an explanation on one level of abstraction can be represented in a graph structure (see figure 1) with two different kinds of labeled nodes: rule-nodes and fact-nodes. Each rule which is used in an explanation is represented by a rule-node, labeled with the name of the rule. Fact-nodes represent case specific facts in an explanation which are either given (e.g. from a problem description) or which are derived by a rule. Fact-nodes and rule-nodes are linked by directed edges. Incoming edges into a rule-node (starting at a fact-node) reflect the premisses of the rule and outgoing edges, leading to fact-nodes, stand for their conclusions.

Two explanations on one level of abstraction are called *identical*, if the graphs are identical except for the labeling of the fact-nodes which can vary. Corresponding rule-nodes must be labeled with the same name of the rule [1].

Explanations on two consecutive levels of abstraction are linked by abstraction mappings, which can relate several level $Level_n$ fact-nodes to a single level $Level_{n+1}$ fact-node. Additionally, a subgraph containing several rule-nodes can be mapped onto a single more abstract rule on the next higher level of abstraction. The similarity between two complete multi-level explanations can now be determined according to the level of abstraction on which the explanations are identical. The higher the level of abstraction the lower is the similarity rating.

The definition of explanation-based similarity presented so far requires complete explanations for both cases to be compared. For the similarity assessment within a case-based reasoning process, a complete case from the case-base (including an explanation) has to be compared with the – currently not solved – problem description not containing an explanation. To allow similarity assessment nevertheless, the

[1] Finer differentiations in explanation similarity can be reached if a similarity rating between rules can be provided.

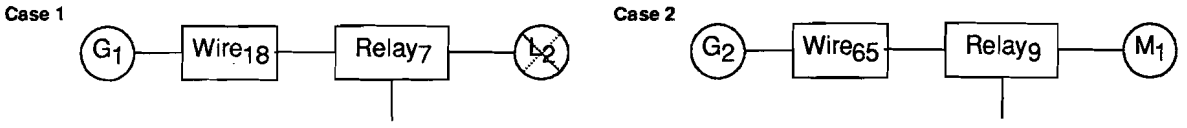


Figure 2: Two cases from a technical diagnostic domain

explanation of the case in the case base is mapped to the current problem description as far as possible. Starting at the highest level of abstraction, it is checked whether the current problem description meets the requirements of the case's explanation at that level [2] . If this is the case, the explanation and thereby also the case's solution is mapped and this process proceeds with next, more concrete level. The level at which the explanation cannot be mapped any more will then indicate the degree of similarity between the case and the current problem. Note, that by the described way of similarity assessment one step of a solution adaptation (mapping of components) has already been performed. An additional adaptation step is required to refine the abstract solution towards the aspired level of detail, which can be achieved by a limited search in the space of possible solutions. The computational cost for the search strictly depends on the number of abstraction levels which have to be bridged and consequently on the degree of similarity between the current problem and the case in the case base. If the similarity is not high enough, the search space which had to be traversed can become so large, that no solution can be found. Such a situation is a strong indication that a new case has to be added to the case base.

2.2 An Example from a Diagnostic Domain

This general idea will now be applied for a diagnostic domain. The goal of diagnostic problem solving is to identify one or sometimes more than one faulty components (call diagnosis) of a system that shows some unintended behavior. The (partially unintended) system's behavior is usually described by a set of symptoms. A complete case consists of a collection of known symptoms (the problem description) together with a diagnosis (solution) which is sufficient to explain all of the observed symptoms. In order to explain the diagnosis domain knowledge about the correct functioning of the system components and their interaction within the system is required. Moreover, the hierarchical part-of decomposition of the system leads to a natural description of the component's behavior on multiple levels of abstraction. Another way for abstracting components is provided by a hierarchical structure abstracting single components and component-groups. An abstraction of `Relay` as well as of `Valve` might be `Switch`.

A simplified example from a technical domain is shown in figure 2. A generator `G1` supplies via a wire and a relay an electric bulb `L2` (see case 1). If the wire breaks, even if the relay is shut the lamp will stay dark. A similar case (case 2) appears by replacing the bulb by a motor `M1`. A description of a machine component will include information about the component's in- and outputs, especially about the value range and the component's behavior. A wire, for example, can be described as follows: It can transmit voltages between 0V and 20,000V (value range). If the wire functions correctly, any input-value will be transfered to the output, but if it is broken, voltage at the output will always be zero (behavior). In an explanation, a certain input- or output-value will be represented as a fact. The actual behavior of the device transforming input values into output values will be represented as a rule. For our two example cases, this way of modeling leads to explanation structures as shown in figure 3.

The two level-1 explanation structures turn out to be not identical because on the considered abstraction level the rules describing the behavior of a motor and the behavior of a bulb are different. But if we look at the explanation the next higher level of abstraction, the behavior of the both different components can be condensed into a single rule which reflects a `doesn't operate` behavior. So, the explanations of both cases are identical at the second level of abstraction.

If we now consider the case-based diagnosis process involving the mapping of the level-2 explanation from case 1 to the problem description of case 2 (generator works, relay is shut, but bulb stays dark) we already achieve the mapping of the faulty component (`wire18`) from case 1 to the related component (`wire65`). In this situation we can see, that the required diagnosis adaptation is completely performed by the explanation mapping. In general, however, an adapted abstract solution needs to be refined towards a concrete diagnosis as it will be shown in section 3.1.

The presented approach has been completely implemented as the MoCAS-system [Pews *et al.*, 1992; Pews and Wess, 1993] which performs a case-based diagnosis task including the described type of solution adaptation for a machine consisting of about 100 components.

[2] Even if the complexity for graph matching is very high [Read and Corneil, 1977], the fact that our nodes are typed by the names of the rules drastically reduces this complexity.

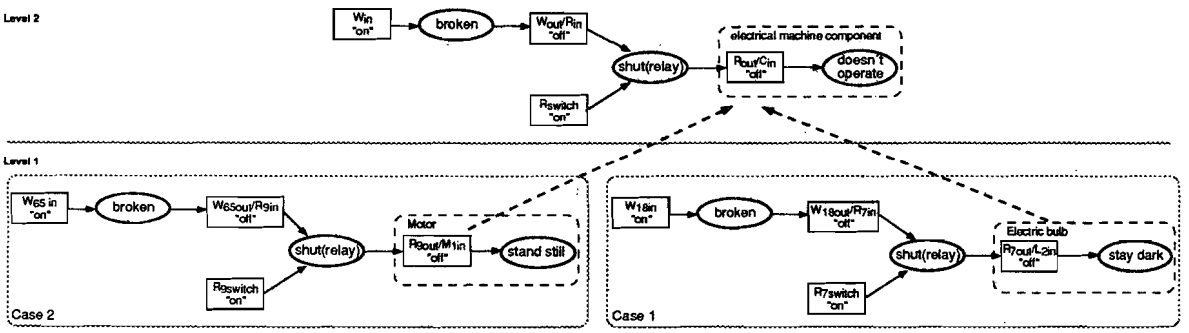


Figure 3: Explanation structures for two diagnostic cases

2.3 An Example from a Planning Domain

In planning, the goal of problem solving is to derive a sequence of actions (or operators), which, when applied, transform a given initial state into a desired goal state. Initial state and goal state together constitute the description of a planning problem and the operator sequence forms the desired solution. The domain knowledge required to explain the correctness of a problem's solution must describe the effects of each available operator in terms of a state transformation function. The operators of a domain can be described on several levels of abstraction, an idea already intensively investigated in research on hierarchical planning [Sacerdoti, 1974]. A plan on a higher level of abstraction consists of fewer, less detailed operators and corresponds to an abstract explanation.

To demonstrate the application of the explanation-based similarity approach for case-based planning, we will employ the well known Towers-of-Hanoi (ToH) domain. A plan which solves a given ToH problem is a sequence of legal single disk move actions. The rules required to explain the correctness of such a plan are the STRIPS representations of the available operator **`Move(Source,Destination)`**. Additionally, operator descriptions on higher levels of abstractions are required such as: **`MoveLargeDisk`**, **`SplitTower`**, or **`JoinTower`**. To achieve the correspondence between the two levels of abstraction, knowledge about plausible abstraction mappings is included, describing, for example, which grouping of single disks on the concrete level forms a tower on the abstract. This entire multi-level modeling is based on an elaborated plan abstraction methodology published elsewhere [Bergmann, 1992].

For the two-disk ToH-case Cp_1 (move the two disks from peg a to peg c), the two-level explanation structure is presented in figure 4. Each rule node in this structure reflects one operation of the solution plan. The fact-nodes show the states of the different pegs during the execution of the plan. The explanation at the abstract level is composed of three, more abstract rules, each of which representing an abstract operation.

If we want to assess the similarity between the case Cp_1 and a new problem Cp_2 in which the two disks have to be moved from peg b to peg a, we can see that the explanation given in figure 4 can be completely mapped for the new problem. The only difference between the explanations for the two problems is the

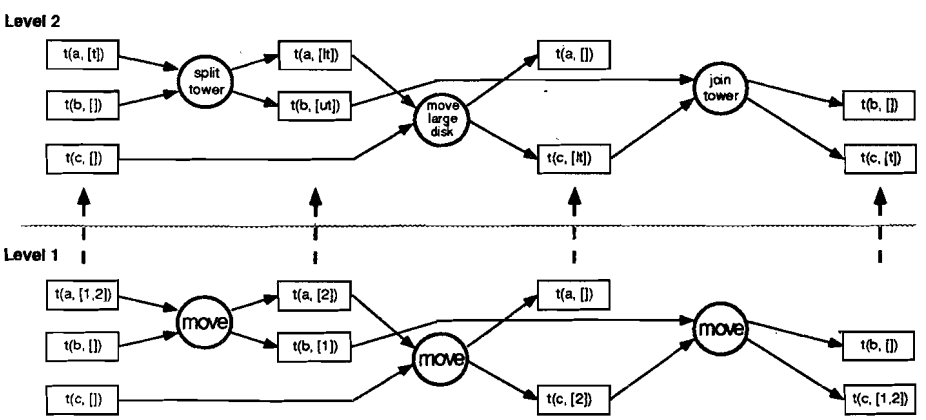


Figure 4: An explanation structure for the Towers-of-Hanoi plan

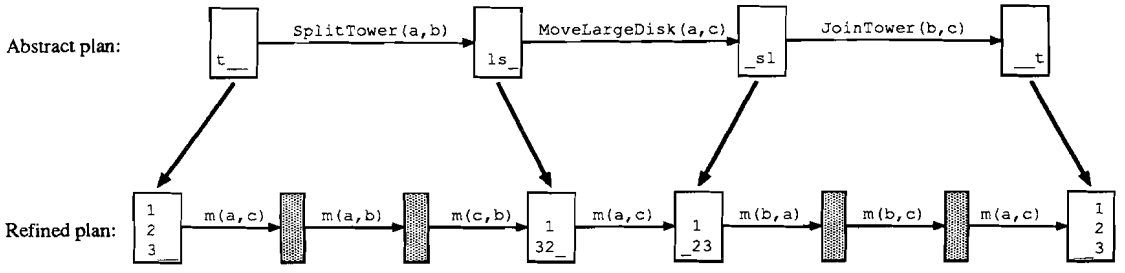


Figure 5: Refinement adaptation for the 3-disk Towers-of-Hanoi problem

labeling of the face-nodes which reflect the alteration of the source and destination pegs. But the fact labels are irrelevant for determining the similarity between two explanation graphs. By comparing case Cp_1 to a third problem Cp_3 in which *three* disks have to be moved from peg a to peg c, it turns out that the explanation at the abstract level can be mapped (a complete abstract tower does now consists of all three disks), while the concrete-level explanation cannot be mapped. So, the cases Cp_1 and Cp_2 are more similar than the cases Cp_1 and Cp_3.

3 Case Adaptation

The result of the up to now described similarity assessment process is not only a (numeric) rating of the similarity, but also a partial mapping of the explanation structure from a case in the case base to the current problem. The computational effort required to achieve a full concrete-level solution (a specific diagnosis or a plan composed of concrete operators) highly depends on the level of concreteness that could be reached by the attempt to map the explanation structure. The more similar the case in the case base, the lower is the required amount of search for solution adaptation. The possibility of solution refinement significantly extends the scope of case reuse with respect the standard case-based and explanation-based approaches.

3.1 Refinement Adaptation in Diagnosis

In case-based diagnosis, the refinement of a diagnosis means specializing a known fault of a more complex component to one (or more) faults in it's sub-components. This only requires a limited search for a potentially faulty sub-components to those which belong to the already known faulty component. As an example, we recall our example cases from figure 2 and consider a third case which differs from case 1 in that the wire is replaced by a more complex transmission component such as an infrared-sender and receiver including several amplifiers.

An explanation mapping at an appropriate level of abstraction will now lead to the identification of the compound component with the broken wire and results therefore in the diagnosis that the compound component might be damaged. The search will now be focussed on the sub-components the compound component consists of; the solution needs to be refined by model-based techniques using the model knowledge already explored during the similarity assessment. Fortunately, in real life it is very often not necessary to identify which sub-part of a component causes the defect when the whole component can simply be replaced.

3.2 Refinement Adaptation in Planning

In case-based planning, refinement adaptation means specializing each operator of the abstract solution plan to a sequence of concrete operators. This is merely a planning task, but performed in a limited search space and consequently assumed to be tractable if the similarity is high enough. If we look, for example, at the adaptation which is required for refining the mapped explanation from the planning case Cp_1 to the problem Cp_3 defined in section 2.3, all of the three abstract operations must be refined as shown in figure 5. An experiment showed that 36 nodes at the concrete level search space had to be visited for this refinement task. 2785 nodes where required to adapt the 2-disk ToH solution for a 4-disk problem. More complexity results for different ToH-problems can be found in [Bergmann, 1993; Surmann, 1993].

4 Discussion

The presented task-independent approach allows for an integration of domain knowledge – which is relatively easy to acquire – into the case-based reasoning process for similarity assessment and solution adaptation. Dependent on the degree of similarity between the current problem and a case in the case base, the system behaves more like a case-based reasoning system or a like model-based reasoning system. As a consequence, the scope for which a case can be employed is increased dependent on the amount of domain knowledge that is entered into the system. A knowledge engineer applying this method can decide whether to enter more cases into the case base or whether to spend additional domain knowledge on more elaborated levels of abstraction to achieve the same competence.

Currently, there are a few other approaches which favor the integration of additional problem solving knowledge knowledge (e.g. [Carbonell and Veloso, 1988]) or more simple causal relationships (e.g. [Barletta and Mark, 1988; Koton, 1988; Janetzko *et al.*, 1992]) into case-based problem solving, while others aim at the integration of different reasoning paradigms (e.g. [Aamodt, 1991]) but mostly in a task-specific manner. In [Birnbaum and Collins, 1988] an approach is described with some similarities with respect to the use of abstraction hierarchies for solution adaptation but with the focus on cross-domain transfer.

Acknowledgements
This research was partially funded by the Commission of the European Communities (ESPRIT contract P6322, the INRECA project). The partners of INRECA are AcknoSoft (prime contractor, France), tecInno (Germany), Irish Medical Systems (Ireland) and the University of Kaiserslautern (Germany).

References

[Aamodt, 1991] Agnar Aamodt. *A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning.* PhD thesis, University of Trondheim, 1991.

[Barletta and Mark, 1988] R. Barletta and W. Mark. Explanation-based indexing of cases. In J. Kolodner, editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 50–60, San Mateo, California, 1988. Morgan Kaufmann Publishers, Inc.

[Bergmann, 1992] R. Bergmann. Learning plan abstractions. In H.J. Ohlbach, editor, *GWAI-92 16th German Workshop on Artificial Intelligence*, volume 671 of *Springer Lecture Notes on AI*, pages 187–198, 1992.

[Bergmann, 1993] R. Bergmann. Integrating abstraction, explanation-based learning from multiple examples and hierarchical clustering with a performance component for planning. In Enric Plaza, editor, *Proceedings of the ECML-93 Workshop on Integrated Learning Architectures (ILA-93)*, Vienna, Austria, 1993.

[Birnbaum and Collins, 1988] L. Birnbaum and G. Collins. The transfer of expertise across planning domains through the acquisition of abstract strategies. In J. Kolodner, editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 61–79, San Mateo, California, 1988. Morgan Kaufmann Publishers, Inc.

[Carbonell and Veloso, 1988] J. Carbonell and M. Veloso. Integrating derivational analogy into a general problem solving architecture. In J. Kolodner, editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 104–124, San Mateo, California, 1988. Morgan Kaufmann Publishers, Inc.

[Janetzko *et al.*, 1992] D. Janetzko, S. Wess, and E. Melis. Goal-driven similarity assessment. In H.J. Ohlbach, editor, *GWAI-92 16th German Workshop on Artificial Intelligence*, volume 671 of *Springer Lecture Notes on AI*, 1992.

[Koton, 1988] P. Koton. Reasoning about evidence in causal explanations. In J. Kolodner, editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 260–270, San Mateo, California, 1988. Morgan Kaufmann Publishers, Inc.

[Musen *et al.*, 1987] M. Musen, L.M. Fagan, D.M. Combs, and E.H. Shortliffe. Use of a domain model to drive an interactive knowledge-editing tool. *Int. J. Man-Machine Studies*, 26:105–121, 1987.

[Newell, 1982] Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982. KIWI.

[Pews and Wess, 1993] G. Pews and S. Wess. Combining model-based approaches and case-based reasoning for similarity assessment and case adaptation in diagnositc applications. Submitted for EWCBR'93, 1993.

[Pews *et al.*, 1992] G. Pews, F. Weiler, and S. Wess. Bestimmung der Ähnlichkeit in der fallbasierten diagnose mit simulationsfähigen maschinenmodellen. In K.D. ALthoff, S. Wess, B. Bartsch-Spörl, and D. Janetzko, editors, *Workshop: Ähnlichkeit von Fällen beim fallbasierten Schliessen*, SEKI WORKING PAPER SWP-92-11, pages 101–106, University of Kaiserslautern, Germany, 1992.

[Read and Corneil, 1977] R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.

[Sacerdoti, 1974] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Schmidt and Zickwolff, 1992] G. Schmidt and M. Zickwolff. Cases, models and integrated knowledge acquisition to formalize operators in manufacturing. In *Proceedings of the 7th Knowledge Acquisition for Knowledge-based Systems Workshop (Banff)*, 1992.

[Surmann, 1993] D. Surmann. Implementierung und vergleichende Untersuchung verschiedener Varianten abstraktionsbasierter Planungsverfahren . Projektarbeit, Universität Kaiserslautern, 1993. in press.

[Wielinga *et al.*, 1992] B. Wielinga, W. VandeVelde, G. Schreiber, and H. Akkermans. Towards a unification of knowledge modelling approaches. In *Proceedings of the 7th Banff Knowledge Acquisition for Knowledge-based Systems Workshop*, 1992.

A Hybrid KBS for Technical Diagnosis Learning and Assistance

David Macchion[1] & Dinh-Phuoc Vo[2]

(1) IRIT (Research Institute in Computer Science of Toulouse) -ARAMIIHS
(2) MMS-F (Matra-Marconi Space France) -ARAMIIHS
31, Rue des Cosmonautes
31077 Toulouse Cedex, France

This paper presents a fault diagnosis design which builds in Model-Based, Case-Based and Rule-Based Reasoning techniques. Within the Model-Based Reasoning layer, the system to be diagnosed is described under functional, structural and causal aspects; basic principles operating on this technical model allow to follow a systematic diagnostic process. Within the Case-Based Reasoning layer, the MBR or Expert's resolutions of past incidents are organized and indexed so that they can be quickly reused and possibly adapted by a four steps CBR engine; a hierarchy of symptoms and some adaptation principles are also defined. Within the Rule-Based Reasoning layer, the solutions of the most frequent incidents are synthesized into rules which can be either manually supplied or automatically generated. Combining these techniques in a predefined but suitable resolution strategy improves the efficiency of the target Knowledge-Based System and increases the scope of its initial competences.

1. Introduction

Technical diagnosis is one of the most active application fields in AI research. Building diagnostic tools for real world systems is both an industrial need and a research challenge. As the application domains get more and more complex, the assessment of the right diagnosis demands more and more sophisticated techniques. Hardware detector devices can be early inserted in the design of such systems in order to track their abnormal behaviors. Assistance tools supported by various AI techniques can be in addition developed for performing faster and more precise diagnoses. While proposing solutions for those complex problems, existing AI techniques are refined and rationale principles in Knowledge-Based System (KBS) design [5] created.

2. Motivations

Our work consists in developing a KBS dedicated to the diagnosis of a complex sub-system of the Ariane-4 launcher. From the detailed study of this application under the supervision of a domain expert, three keypoints were identified. First, we observed that *different types of knowledge* are available for supporting diagnostic performance [18], especially technical documentation, incident forms, general and specific electrical principles... Second, we noted that the expert sometimes calls on *personal experience* to empirically optimize and confirm his diagnostic conclusions. Finally, we noticed that the resolution of frequent incidents seems to involve surface knowledge whereas the resolution of unusual ones demands a *deeper knowledge* [12]. Taking these facts into account led us to design what is labelled a «Hybrid Knowledge-Based System» for technical diagnosis [2]. An implementation of this design is currently being developed.

3. Ariane-4 Vehicle Equipment Bays

The Vehicle Equipment Bays (VEB) of the Ariane-4 launcher are assembled in the Matra-Marconi Space center based in Toulouse. A VEB must schedule the launcher flight by computing guidance actions, telemetry emissions, order diffusions... Engineer teams are in charge of the assembling of their various equipments (On-board computer, electronic sequencing unit, interface unit, telemetry modules...) before running out exhaustive electrical tests. In the course of these test procedures, a huge amount of VEB parameters is acquired by surveillance devices (relay state control mechanisms, analogical and digital acquisitions, equipment status word generations...) situated either in the VEB or in the test bench. These parameters are sent to a ground computer which automatically detects branchings between effective and expected values. Specific anomaly messages are then reported on a listing so that human operators can start their investigations.

4. Combining rules, cases and a technical model

We have just presented the main issues of our expertise domain i.e., the availability of different knowledge sources, the limits of a systematic diagnostic process and the heterogeneous complexity of the treated incidents.

Although the nominal running of the VEB is perfectly defined, we in short noticed that it is insufficient for deducing its disfunctioning behavior. In this context, neither pure Model-Based Reasoning [20] nor pure Expert System [15] techniques are convenient. Consequently we chose to combine different techniques, namely Model-Based (MBR), Case-Based (CBR) and Rule-Based (RBR) Reasonings as partially discussed in [1], [8], [14]. Relying on their different representation formalisms, we include in a single KBS a detailed technical model, a memory of encountered incidents and a set of shorcutting rules.

5. Model-Based Reasoning layer

The basic layer of our KBS is MBR and assigned to three major purposes : it characterizes the *domain concepts* into hierarchies; it describes in a technical model some *functional, causal and structural aspects* of the VEB [11]; it provides some heavy and slow *propagation principles* to operate on the technical model [7]. The domain concepts are available for the CBR and RBR layers to fill the description of the cases and the premisses of the rules. The technical model can be used at different phases of the CBR resolution, especially retrieval and adaptation time. The diagnostic capabilities of this layer are activated as long as the RBR and CBR layers fail in providing a satisfying solution to a given incident.

5.1. Domain concepts

The technical vocabulary of the VEB is organized into classes of *equipments* (relays, fuses, sensing units...), *functions* i.e. equipment roles (transmission, acquisition, alimentation...), *flows* i.e. spreading information (orders, status words...), *parameters* (cycles, voltage values...), *anomaly messages* (telemetry, analogical acquisitions...)... Methods for computing functioning details, testing specific equipments, asserting message dependencies... are also proposed.

5.2. Technical model

From the functional point of view, an equipment is split into its different functions which must reflect its Input/Output (IOs) dependencies and its roles within the VEB. As complex functions are implemented by lower ones, significant functions are often difficult to extract from the VEB design. Therefore many function classes and decompositions have to be distinguished. Nominal behavior can be expressed for producing expected outputs from expected inputs.

Causal knowledge is gathered in a set of causal links. A *causal link* stresses a relation between one or several anomaly messages and a function breakdown. As detector hardware devices are placed along some of the spreading flows, some causality links can be directly deduced from the structure of the VEB. This primary causal knowledge is not yet sufficient to precisely and reliably localize the guilty function. This type of knowledge must therefore be improved in the light of incident cases while real faults imply real symptoms.

From the structural aspect, equipment functions are connected by buses, wires or cables. Any function whether it is complex or atomic, its subsequent and following neighbors and the *connection* type are therefore mentioned. This knowlegde allows to quickly retrieve the path of a given flow. CBR adaptation mechanisms can also use this information to transform the solution of a similar case.

5.3. Diagnosis capabilities

The resolution capabilities of the MBR layer are first supported by a straightforward principle : the fault must be in the intersection set of the functions placed upstream the ones which have sent an anomaly message. Then, improved or primary causal knowledge can be applied to discard a suspected function F : if the anomaly messages observed in a past case involving F do not reappear, then F is suppressed; the alternate way to discard F is to verify that one of its dowstream detector functions has not emitted. Finally, a reduced number of manual tests has to be performed to fix the right diagnosis. Consequently, the resolution process we adopted is split into two main phases : we first *delimit a minimal space* of hypotheses; we then *compare each function expected and received IOs* to find out the faulty one. Expected values must be computed by the MBR functioning knowledge whereas received values are given by the user.

6. Case-Based Reasoning layer

The second layer of our KBS is case-oriented and works on four different types of case. It stores in *solved cases* the resolutions processed by the MBR solver so that they can be quickly recalled for forthcoming incidents. During learning dialogues with an instructor, it retains in *unexplained cases* the solution of the diagnoses the MBR layer does not succeed in solving because they include temporal aspects or demand too deep technical information; this task can create and update the causal knowledge since new relations between faults and symptoms are discovered. It stores *exception cases which* diagnosis is different from the one deduced by a diagnostic rule which

premisses nevertheless match the case description. Finally, it can generate *situation cases* which organize under labelled situations a collection of cases which shares a set of common indexes. A four phases CBR engine operating on a dedicated case structure and memory organization is activated to attempt to give the solution to the RBR unsolved incidents.

6.1. Case structure

For our requirements, we structure a diagnostic case under six attributes. Its *description* consists in the list of its observed anomaly messages. Its *situation* may attach the case to a more prototypical one i.e. a situation case. Its *process* represents the ordered list of the proceeded tests and associated results. Its *diagnosis* then establishes the function which was in fault and the state of its IOs. Unexplained cases may contain specific details in the *context* attribute instead of a list of proceeded tests. The *repair* attribute is filled with the actions to be taken.

6.2. Case memory

Our diagnostic cases are organized in a *hierarchy* [19], [17] of situation cases i.e., more or less abstract classifications of incident types. In the first levels of the hierarchy, the cases are stored according to their description. Two diagnosis cases with at least one similar anomaly message may be placed under a situational case indexed by the commom symptoms. Situation cases representing prediagnostic conclusions drawn by the expert out of well-known combinaitions of symptoms can be manually added [21]. Otherwise, the symptom presence, importance or absence are taken into account to automatically construct consistent clusters. The intermediate levels of the hierarchy systematically divide the situational cases according to test results and functioning conditions. Then, the leaves of the hierarchy contain diagnosis cases with their associated diagnosis and repair actions.

Indexes relating one or several symptoms to a case have either a positive or negative influence on the associated diagnosis. *Positive indexes* can be of remembrance, sufficiency, necessity or equivalence types : remembrance index allows to extract a case from the memory; a sufficiency index allows to retain a case for presentation; a necessity index allows to discard a retrieved case which does not present the associated symptoms. *Negative indexes* can be of inhibition or exclusion types [4] : an inhibition index decreases the remembrance strength of a case; an exclusion index definitively discard a case for consideration.

6.3. Retrieval phase

The two phases of the MBR diagnostic process also underlie the CBR resolution. First, positive indexes are first used to *recall a plausible set of cases*. Negative and necessity ones can *reduce this hypothesis set*. The strength of the useful indexes are combined to *form a ranked list of cases*. If the hypothesis list is empty, indexes containing symptoms that share the same superclass [10] with the observed ones are attempted. Then, once an a priori satisfying situation case is assessed, the intermediate levels of the hierarchy support the systematic comparison between the past case configuration and the present problem. If several sub-cases compete, their appearance rating determines the presentation order. As a leave is reached, its diagnosis is proposed for approval. If a solved case is directly retrieved, it is proposed before any concurrent situational case. If a MBR unexplained case is retrieved, the applicability of its context is checked before presenteation.

6.4. Adaptation phase

As the CBR retriever may have retained some cases with symptoms different from the current ones, adaptation is sometimes required to transform the past solutions. We have defined a very limited set of adaptation principles since it seems rather incoherent to manipulate the diagnosis of a source case which does not share the same symptoms than the target. The first principle is applicable when only one symptom is present : we can then *interchange the past detector* function with the new one and propose it for diagnosis. The second principle is applicable when two correlated symptoms are present : their *common upstream contributors* can then be proposed. The last principle is rather an escape clause : considering that breakdowns on neighbor functions may yield the same symptoms, *subsequent or following functions* may become the right diagnoses. This convenient technique allows to traverse the hierarchies of cases. For future reuse, those patched successes can be stored in *branching links*. If available, an explicit mean of distinction may fill the branching link.

6.5. Approval phase

Three significant CBR inferences must be presented to the user for approval. First and if available, the primary situational levels must be approved since they define the set of diagnostic cases which arouses suspicion. Second, the diagnosis process must be fully applied to verify that the suspected functions have the same IO states as they did : the effective IOs of the functions to be tested and the comparisons to the expected IOs are asked and presented. Finally, it must be proved by manual tests that the incriminated function has got correct inputs and abnormal outputs. If the user is not expert, the first approval can be avoided : this will only have a possible impact on the number of cases the system will try before reaching the right solution.

6.6. Storage phase

Once the diagnosis has been established by the CBR or MBR layers or by the instructor, the target case has to be stored in the hierarchy. In the most favorable situation, it can be *merged* to the source case : the strength of its indexes can be appreciably increased; if it is dependant of a situational case, the appearance rating is also augmented. If a failure occurs but is successfully adapted, *branching links are created.* If the system proposes an incorrect case, *negative indexes can be placed* or positive indexes can be decreased. If the right diagnosis is not proposed, maybe it is possible that new *tests only need to be added* to end up at the solution. If the instructor decides on his own that the case brings to the fore a *new situation*, he must enter new indexes, resolution steps and diagnosis. If the case is finally unexplained, the instructor supplies some more context precisions and its solution.

7. Rule-Based Reasoning layer

The last layer of our KBS consists in a set of shorcutting rules. Its purpose is to *retain the solution of the incidents which often reappear.* Diagnosis rules can be manually supplied to catch well-known situations and then to refine, generate and validate their associated hypotheses. Factual rules conclude with any intermediate results of a diagnosis rule. They are defined in the MBR layer for establishing the function states and dependencies between messages. Diagnosis rules can also be generated by the *index analyser* from index evolutions. For instance, as soon as a remembrance index strength exceeds the sufficiency threshold, an attempt is made to convert it into a suspicion rule of its associated situation. During next interactive sessions, *exception cases* weakening this rule can nevertheless be encountered. Before activating the competences of the CBR layer, the diagnosis rules of the RBR layer are tried.

8. RBR, CBR and MBR interactions

The knowledge of our KBS is shared out among MBR, CBR and RBR layers. Diagnosis rules can be initially provided by the domain expert or automatically created by the index analyser out of sufficency, exclusion, necessity and equivalence indexes. Exception cases are created as soon as the conclusion of a fired rule does not turn out to be the right diagnosis. Solved cases are associated to the incidents successfully treated by the MBR layer. Unexplained cases diagnosis and context are provided by the expert when the MBR solver reports a failure. Situation cases are proposed for creation to the expert as soon as several diagnosis cases share a common set of symptoms. Causal knowledge is extracted from diagnosis cases or VEB design and is used by the MBR layer for hypothesis suppression. Figure 1 sums up the possible relationships between the three layers.

The strategy including the RBR, the CBR and the MBR competences is fixed. Solving a target incident consists in a *sequential activation of the RBR, the CBR and the MBR competences.* Within the RBR layer, the solution can be provided by a diagnostic rule or an exception case. If the RBR layer fails, a solved or unexplained case of the CBR layer may give the solution. Otherwise, the MBR capabilities inspect each of the functions involved in a minimal set. If the diagnosis is not yet found, the instructor has to supply the solution. From the cognitive point of view, we claim that this global strategy tends to reflect our expert's resolution process : if the incident keeps recurring, he can automatically recognize its situation; if the problem already occurred, he can recall its past global or partial resolution. Otherwise, he must activate a heavier process based on a technical representation of the equipment to be diagnosed.

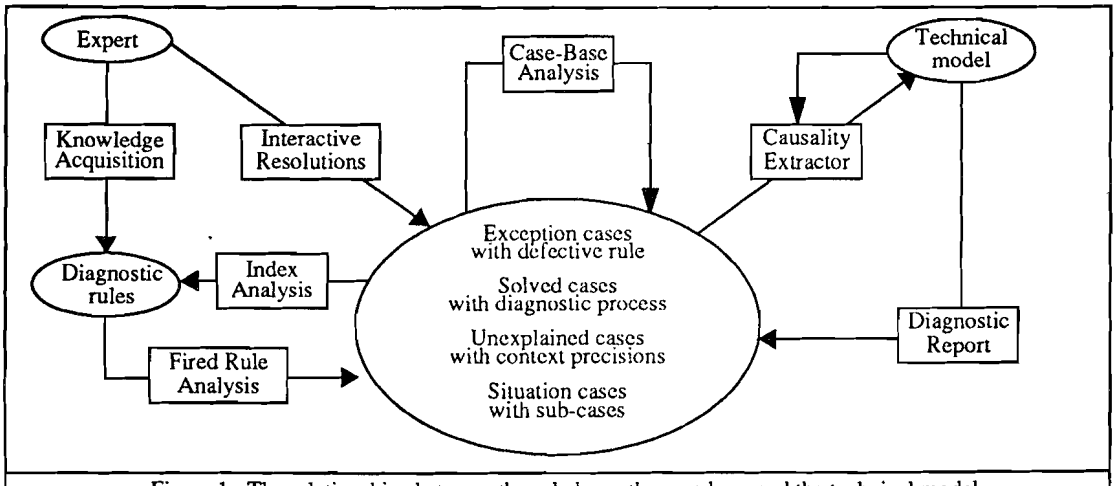


Figure 1 : The relationships between the rule base, the case base and the technical model.

9. LOIR (Lisp Objet Inférence Réflexe)

We are developing our KBS with LOIR, a hybrid language realized by a team [3] of the IRIT laboratory. It provides various functionalities such as a Common Lisp layer, a reflex inference engine, a frame-based formalism, pattern matching mechanisms and a dialogue and action models. Frames are convenient to describe the technical vocabulary of the VEB. Our cases are implemented in dedicated frames accessed by typically CBR methods : retrieval, indexing, adaptation... Any kind of rule is triggered with backward, forward or mixed inferences. All the methods managing objects are Lisp written.

10. Related works

The main originality of our research is to integrate in a technical diagnosis system Rule-Based, Case-Based and Model-Based Reasonings whereas most of the CBR systems only combine rules and cases or cases and a model. The Casey system [14] also offers the possibility to build solutions from scratch by activating its MBR solver. However, the MBR layer only reasons with a pure causal knowledge relating patient's states to others states or diseases. The Cabaret design [22] provides sophisticated heuristics for controlling and interleaving reasoning with cases and reasoning with rules. We only retained the possibility to activate an exception case within the rule-based layer. The Mud-Creek system [1] exploits functional and structural links at retrieval time in order to enhance the intial set of findings. We thought it better to use this knowledge type within the exhaustive diagnostic process of our model-based layer.

11. Conclusion and future works

Diagnostic tasks in real world applications cannot be properly described in a unified and clear-cut manner. Combining different reasoning techniques seems to be a significant and promising way to build more powerful KBS. By including MBR paradigm, the domain concepts and some basic functioning principles can be expressed. By including CBR paradigm, the system can learn and improve its competences in the course of being used. By including RBR paradigm, shorcutting rules can quickly provide the diagnosis of well-known incidents. The design we presented in this paper attempts to make its contribution to this hot topic research.

Although many implementation problems still retain our current efforts, we have already stressed two design points to improve :

- The different *layers neither confront nor cooperate* : the first validated solution stops the search process and only the supplier layer worked. Therefore, the knowledge base may become inconsistent while new cases are learned. To limit these inconsistency risks, the different formalisms could be unified in a more complex case notion which would include systematic rules, past situations and hesitant processes as discussed in [16, 13]. However, we claim first that the relative independence of the presented layers allows an incremental development and second that the layer competences may remain complementary.
- The *domain dependent concepts and most of the technical aspects are definitively fixed* : the CBR and RBR layers could not work without this prerequisited knowledge. How did we a priori decide which knowledge would be significant for our coming CBR and RBR needs ? This complex task was in fact manually performed out of acquisition phases. Moreover, many past incidents influenced our decisions to define the boundaries of the domain knowledge. Improving problem solving competences cannot be therefore separated from learning strategic knowledge [9, 6] as well as domain knowledge [4].

12. References

1. A. Aamodt, «A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning», Ph.D Thesis of the Trondheim University, Norway, 1991.
2. K. Althoff & S.Wess, «Case-Based Knowledge Acquisition, Learning and Problem Solving For Diagnostic Real World Tasks», Proceedings of EKAW, Mai 1991.
3. U. Arronategui & F. Mieulet, «Le langage LOIR : objets, règles et actions pour la modélisation», Ph.D Thesis of the Toulouse III University, France, June 1992.
4. R. Bareiss, «Exemplar-based knowledge acquisition : A Unified Approach to Concept Representation, Classification, and Learning», Academic Press, 1989.
5. J. Breuker, G. Schreiber & B. Wielinga, «KADS : A modelling approach to Knowledge Engeneering», Knowledge Acquisition Vol 4, KADS issues pp.5-53, Summer 1992.
6. G. Carbonell, «Derivational Analogy : A Theory of Reconstructive Problem Solving and Expertise Acquisition», B. Buchanan & D. Wilkins editors, pp. 727-738, M. Kaufman, 1993.

7. R. Davis, «Diagnostic Reasoning Based on Structure and Behavior», Artificial Intelligence 24, pp.347-410, 1984.
8. A. Goel, «Integrating Case-Based and Model-Based Reasoning : A Computational Model of Design Problem Solving», AI MAGAZINE, pp. 50-53, Summer 1992.
9. T. Gruber, «The Acquisition of Strategic Knowledge», Academic Press Inc, 1989.
10. K. Hammond, «Case-Based Planning : Viewing Planning as a Memory Task», Academic Press, 1989.
11. A. Keuneke, «Machine Understanding of Devices : Causal Explanation of Diagnostic Conclusions», Ph.D Thesis of the Colombus Univeristy, Ohio, 1989.
12. D. Klein & T. Finin, «What's in a Deep Model ? : A Characterization of Knowledge Depth in Intelligent Safety Systems» pp. 559-562, IJCAI, 1987.
13. J. Kolodner, «Towards an understanding of the role of experience in the evolution from novice to expert», International Journal of Man-Machine Studies No 19, pp. 497-518, Academic Press, 1983.
14 P. Koton, «Using Experience in Learning and Problem Solving», Technical Report, Massachussetts Institute of Technology, October 1988.
15. J. Moustafiadès, «Formation au diagnostic technique : l'apport de l'IA», Masson, 1990.
16. R.Schank, «Dynamic Memory : A theory of Reminding and Learning in Computers and People», Cambridge University Press, 1982.
17. R. Schank & C. Riesbeck, «Inside Case-Based Reasoning», Lawrence Erlbaum Associated, 1989.
18. F. Schmalhofer & J. Thoben, «The Model-Based Construction of a Case-Oriented Expert System», AICOM Vol 5 No 1, pp. 3-18, Mai 1992.
19. P. Sycara, «Case-Based Reasoning», CBR tutorial of the EM2SL, July 1991.
20. R. Reiter, «A theory of Diagnosis from first Principles», Artificial Intelligence 32, pp. 57-95, 1987.
21. ReMind™,Functional specifications for REMIND development version 1.0
22. E. Rissland & D. Skalak, «Combining Case-Based Reasoning and Rule-Based Reasoning : A Heuristic Approach», IJCAI-89, Vol 1, pp. 20-25, August 1989.

Induction and Reasoning from Cases

Michel MANAGO [1], Klaus-Dieter ALTHOFF [2], Eric AURIOL [1], Ralph TRAPHÖNER [3], Stefan WESS [2], Noël CONRUYT [1], Frank MAURER [2]

1 Introduction

We present the INRECA european project (ESPRIT 6322) on integration of induction and case-based reasoning (CBR) technologies for solving diagnostic tasks. A key distinction between case-based reasoning and induction is given in [1]: "In case-based methods, a new problem is solved by recognising its similarities to a specific known problem then transferring the solution of the known problem to new one (...) In contrast, other methods of problem solving derive a solution either from a general characterisation of a group of problems or by search through a still more general body of knowledge". In this paper, we distinguish between a pure inductive approach and a case-based one on the basis that induction first computes an abstraction of the case database (ex: a decision tree or a set of rules) and then uses this general knowledge for problem solving. During the problem solving stage, the system does not access the cases

2 INRECA's Inductive and Case-Based Approaches

Induction is a technology that automatically extracts general knowledge from training cases. KATE is the inductive component of INRECA. It builds a decision tree from the cases by using the same search strategy, hill-climbing, and same preference criteria that is based on Shannon's entropy as ID_3 [2]. Unlike most induction algorithms, KATE can handle complex domains where cases are represented as structured objects with relations and it can use background knowledge. At each node, KATE generates the set of relevant attributes of objects for the current context and selects the one that yields the highest information gain. For instance, an attributes such as "pregnant" for a patient whose sex is known to be "male" further up in the decision tree is eliminated before the information gain computation. Background domain knowledge and class descriptions allow to constrain the search space during induction [3].

Case-based reasoning is a technology that makes direct use of past experiences to solve a new problem by recognising its similarity with a specific known problem and by applying the known solution to the new problem. PATDEX is the case-based component of INRECA. It consists of two case-based reasoning subcomponents for classification and test selection. A procedure that dynamically partitions the case base enables an efficient computation and updating of the similarity measures used by the CBR subcomponents. For the classification subcomponent, the applied similarity measures are dynamic. The underlying evaluation function is adapted using a connectionist learning technique (competitive learning). For the test selection, the adaptation of similarity measures is based on an estimation of the average costs for ascertaining symptoms using an A*-like procedure. PATDEX can deal with redundant, incomplete, and incorrect cases and includes the processing of uncertain knowledge through default values. PATDEX is described in [4] and [5].

[1] AcknoSoft , 58a rue du Dessous des Berges, 75013 Paris - France. [2] University of Kaiserslautern, Dept. of Computer Science, PO Box 3049, 67653 Kaiserslautern - Germany. [3] tecInno GmbH, Sauerwiesen 2, 67661 Kaiserslautern - Germany.

3 The Need for Integration

INRECA integrates induction and case-based reasoning so that they can collaborate and provide better solutions than they would individually. Before describing how integration is performed, we first state why the two approaches are complementary. Induction presents some limitations for building an identification system that can handle missing values during consultation. Consider the following case base drawn from an application that identifies marine sponges developed at the Museum of Natural History in Paris.

CASE	CLASS	SHAPE(BODY)	TEETH-TIP(MACRAMPHIDISQUES)	...
Ex1	PARADISCONEMA	ELLIPSOID	LARGE	...
Ex2	COSCINONEMA	CONICAL	LANCET-SHAPE	...
Ex3	CORYNONEMA	ELLIPSOID	LANCET-SHAPE	...
...	...	...	...	...

Table 1 - A database of cases for an application which identifies marine sponges

KATE works in two steps: it first learns a decision tree and then uses the tree to identify the unknown class of a new incoming sponge. Consider what happens when the user does not know how to answer the first question asked during consultation of the tree of figure 1.

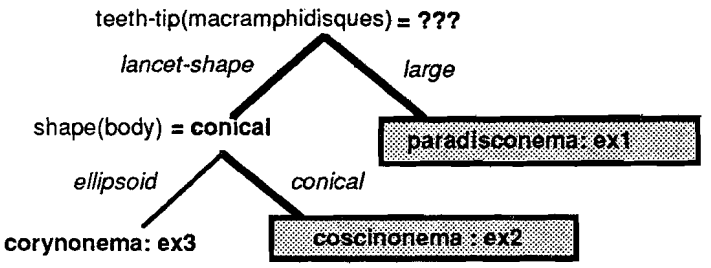


Figure 1: A consultation of the decision tree learned by KATE

When the user answers "unknown", KATE proceeds by following both branches "lancet-shape" and "large" and combines the conclusions found at the leaves. In the "large" branch, it reaches the "Paradisconema" leaf node. In the "lancet-shape" branch, it reaches a test node and the user is queried for the value of the "shape" of the object "body". He answers "conical". KATE reaches the "Coscinonema" leaf and combines the two leaves to conclude that the current case is a "Paradisconema" with a probability of 0.5 or a "Coscinonema" with a probability of 0.5. Consider case ex1 at the "Paradisconema" leaf node. The feature "shape(body)" of ex1 has the value "ellipsoid" unlike the current case where it is "conical". Thus, the current case is closer to ex2 than to ex1 and the correct conclusion is "Coscinonema" with a probability of 1. Unfortunately, the information about the "body shape" of ex1 was generalized away during induction and is no longer available during consultation.

Note that there are other methods for handling unknown values during consultation of a tree. Instead of combining branches, one can assign a probability to the branches [6] and follow the most probable one. However, this does not remove the problem presented above. This problem is not caused by a flaw of the particular induction algorithm used by KATE since we could have used another algorithm and encounter a similar problem. It is not a flaw of the decision tree representation formalism since we could have used production rules generated automatically or manually and still run into this same problem. It is caused by the fact that we are reasoning using an abstraction of the training cases and have generalized away and thus lost some discriminant information. If the consultation system is to handle any configuration of unknown values, such as for applications that deal with photo-interpretation of objects whose features may be hidden in any combinations, case-based reasoning will always perform better than rule-based, decision tree-based or even neural network-based identification systems.

This has been confirmed by a set of experiments conducted using PATDEX. We have measured its ability to reach a correct solution when the working case is incomplete (i.e. contains unknown values). Experiments have been conducted with a training set of one hundred cases. The test set also consists of one hundred cases. For every test case the number of known symptom values has been stepwise reduced. Classification accuracy is measured against reduction of the presented information. The results are shown in table 1. Here, a reduced information of 70% means that every case is classified based on 30% of its known symptom values (where 60% of such cases have been correctly classified).

Reduced information (%)	0	10	20	30	40	50	60	70	80	90	100
Classification accuracy (%)	100	99	97	96	91	90	76	60	28	11	0

Table 2 - Measuring Correctness against Reduction of Information

As confirmed by this set of experiments, up to a certain limit, classification accuracy is not significantly decreased by reducing the number of known attribute values in the current case. For instance, when half of the values are missing the system still correctly identifies 90% of the test cases. When using induction, a single missing value for an attribute in the decision tree (this corresponds to a 0.5% reduction in the information available) yields a loss of 50% in accuracy. When a feature is unknown, a case-based reasoning tool looks for alternative features to identify the current case. CBR reacts dynamically and exploit all the information available. In addition, a CBR system is more resilient to errors made by the user during consultation since it computes a similarity measure from the global description of the cases and not a minimal subset like with the inductive approach. It can confirm the conclusions by asking additional questions that modify the similarity measure accordingly.

This does not imply that CBR always performs better than induction. During the first year of INRECA, we have defined a catalog of industrial criteria to conduct experiments and compare the two technologies. Our criteria catalog does not merely adresses technical issues such as performance and effectiveness, but also ergonomic and economic aspects such as user acceptance of the technology (domain specialist, naive end-user, data clerk, case engineer etc.), ease to build, validate and maintain the application and so on. After analysis, we claim that induction and CBR are complementary techniques and that integrating these will improve their standalone capabilities. Our comparison is summarized in the next section. The criterias have been introduced in hierarchical weighted grids to compare in an objective and exhautive manner the induction and CBR components of INRECA as well as other existing tools.

4 Comparision of Induction and CBR

We summarize the respective merits of the techniques in the following table. Although the experiments have been conducted using PATDEX and KATE, the conclusions drawn are applicable to the underlying technologies in general. Note that according to the distinction between induction and CBR that has been explained in the introduction, we view tools that access the training cases to incrementally maintain the induced rules or trees as CBR tools.

Advantages of PATDEX (CBR)	Advantages of KATE (Induction)
The application is always up-to-date because CBR can work incrementally.	The consultation is consistent: what is true today will be true tomorrow (unless the tree has been updated).
CBR handles missing values during consultation and makes optimal use of the information available.	The decision tree can be compiled into a runtime that does not require the case base to do diagnosis. It can be easily integrated in the customer's environment.
CBR can widen the set of current hypothesis whereas induction only shrinks it.	The system supports exploratory data analysis and does consistency checks in the data base.
The CBR consultation is more flexible for the user of the consultation system. It can be driven by the user who supply the information he wants instead of being guided step by step through a decision tree. It can handle sensor input and react dynamically to the data.	The domain specialist can influence or even impose how the consultation is done by modifying the tree by hand. He controls the consultation process.
The CBR consultation is more resilient to errors. After finding a conclusion, the current solutions can be confirmed or refuted.	A classification of the data can be constructed based on the information contained in the tree.
Analogies can be made based on the whole case description instead of a minimal subset.	Induction produces a generalisation of the cases and turns data into knowledge.
The similarity measure used by PATDEX can evolve over time and is adaptable.	
The current consultation can be explained to the user by presenting previous cases.	The current consultation can be explained to the user by presenting the classification rule.
CBR interprets cases dynamically.	The consultation of the learnt tree is more performant than the CBR consultation

Table 3 - Cost-Benefit Analysis of Induction and CBR

5. Integrating Induction and CBR

Four critical levels of integration have been identified. For the first level, the two techniques are seating side-by-side and are provided as stand-alone modules that work on the same case data expressed in the CASUEL object-oriented language (**toolbox strategy**). This is useful because a single technique may match the user's needs for a particular application, while a combination of both may not. In addition, a decision tree produced by induction allows to detect the inconsistencies of a case database before its use by a case-based reasoning module. For the second level of integration, the two techniques are able to exchange results via the CASUEL representation language (**cooperative strategy**). The results of one may help to improve the efficiency and to extend the classification capabilities of the other. More precisely, a decision tree produced by induction can speed up the consultation by the case-based reasoner. The case-based reasoner can supplement the decision tree when choosing among different conclusions (case-based reasoning is started at the end of the consultation of the tree or during consultation when encountering unknown values). The third level of integration allows the combination of individual modules of the tools (**workbench strategy**). For instance, the information gain measure module may be used to choose the next attribute to be asked during an interactive CBR consultation. The last level fulfils the final goal of INRECA (**seamless integration**) by mixing the most relevant parts of the two technologies in a single system. Two critical modules are identified: the information gain computation module for the induction technique, and the similarity computation module for the case-based reasoning technique.

Our main point is that a single system will never meet the needs of everyone. INRECA offers several integration possibilities and must be configured to meet the requirements of a particular application or of a particular category of users. For instance, a naive end-user must be guided

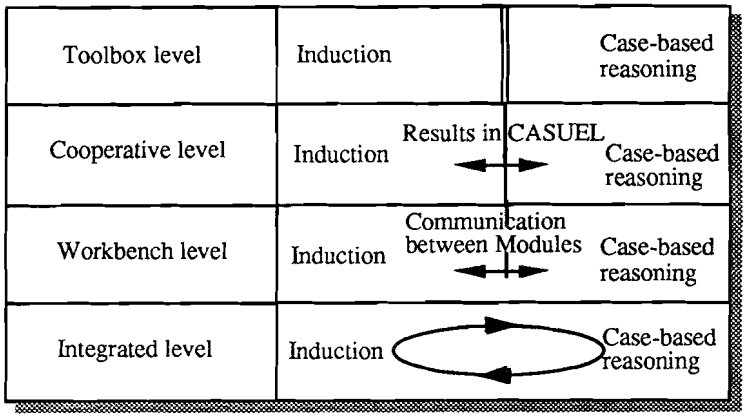


Figure 2. Four integration levels between Kate and Patdex

step-by-step by the consultation system in a decision-tree like fashion. On the other end, a domain specialist wants to directly supply whatever information he feels is relevant and remain in control of the consultation system. Moreover, what may be viewed as an advantage of a technology in a given context may turn out to be a drawback in another. For instance, incrementality can be seen as an advantage of CBR over induction to maintain the consulation system automatically and keep up with the knowledge that workers learn through their daily experience. On the other end, we are currently working with an equipement manufacturer who distributes the diagnostic system to his customers and who wants to control the advices that are given to the users (let it be for legual reasons). Thus, he prefers a system that does not evolve permanently and that behaves in a predictable way. In that context, the incrementality is a drawback since he wants to compile the case data into an induction tree that is maintained by him periodically. Finally, one technique may be better adapted at a specific stage of the application life cycle (for example, CBR at the begining to enrich the case database) but not at a later stage (for example, induction can compile the case database when it becomes too big and when efficiency becomes a problem). Thus, INRECA provides several options for the four levels of integration and can be configurated by the application developper . In the next section, we present an architecture that deals with the problem of handling unknown values using CBR, but that pre-index the cases using a decision tree for efficiency.

6. An Integration Archicture to Handle Missing Values Efficiently

As stated in section 3, one main drawback of a decision tree consultation occurs if the user answers "unknown" to a test. Unknown values propagate an uncertainty along all the branches of the "unknown node" - we define an unknown node as a node where the user answers "unknown" during the consultation of the tree although a subsequent test may remove this uncertainty. Moreover, the final diagnosis is probabilistic which is confusing for a non expert user. One way to deal with unknown values in the consultation of a tree is to switch to a case-based reasoning procedure after consulting the tree. When an unknown value is encountered, the consultation of the tree is stopped and the case-based reasoner is used to choose the next tests. The probabilistic diagnoses delivered by Kate may also be refined by using the similarity measure of the case-based reasoner. A workbench integration is needed. The procedure when encountering an unknown value in the consultation of the decision tree is presented below:

```
1. Get the current situation given by the first tests
   of the tree.
2. Get the current subset of the cases listed under the
   unknown node.
3. Switch to Patdex by using the current situation and
   the current set of cases.
```

Procedure for Switching between Kate and Patdex

This procedure combines the advantages of both techniques for efficiency and correctness. In the worst case, the user answers unknown at the root node and we are left with a classical CBR consultation. In the best case, the user never answers unknown and we are left with a classical decision tree traversal mechanism that is very efficient.

Conclusions

Induction and case-based reasoning are complementary.approaches for developing experience-based diagnostic systems. Induction *compiles* past experiences into general knowledge used to solve problems. Case-based reasoning directly *interprets* past experiences. Both technologies complement each other. Induction is used for detecting inconsistencies in the case data base, case-based reasoning is used during consulation to retrieve similar cases when there are missing values. The induction system can compute a tree to index cases on a predefined number of levels in order to improve the efficiency of case-based reasoning. After traversing that partial tree (interactive consultation), we are left at a leaf node with an initial candidate set that can be passed to the case-based reasoning system. As a consequence, the case-based reasoner works on a much smaller set of candidates. The partial decisions can be confirmed or refuted by the case-based reasoner. In the latter case the tree needs to be updated.

Acknowledgement

Funding for INRECA has been provided by the Commission of the European Communities (ESPRIT contract P6322). The partners of INRECA are AcknoSoft (prime contractor, France), tecInno (Germany), Irish Medical Systems (Ireland), the University of Kaiserslautern (Germany). KATE is a trademark of Michel Manago. We thank Prof. Claude Lévi and Mr Jacques Le Renard at the Museum of Natural History in Paris for providing the sample application used to illustrate some of the ideas presented here. We also thank Mr Thomas Schultz who has helped us refine our criteria list and who validated and filled our comparison grids for several CBR tools.

References

[1] Bareiss, R. (1989). Exemplar-Based Knowledge Acquisition. London: Academic Press

[2] Quinlan, R. (1983) Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds), *Machine Learning: An Artificial Intelligence Approach* (Vol. 1). Morgan Kaufmann.

[3] Manago M. (1989). "Knowledge Intensive Induction", proceedings of the sixth "International Machine Learning workshop", Morgan Kaufmann.

[4] Althoff, K.-D. & Wess, S. (1991). "Case-Based Knowledge Acquisition, Learning and Problem Solving in Diagnostic Real World Tasks". *Proc. EKAW-91, Glasgow & Crieff*; also: GMD-Studien Nr. 211 (edited by M. Linster and B. Gaines)

[5] Richter, M. M. & Wess, S. (1991). "Similarity, Uncertainty and Case-Based Reasoning in PATDEX". *Automated Reasoning - Essays in Honor of Woody Bledsoe*, Kluwer Academic Publishers

[6] Quinlan, J. R. (1989). "Unknown Attribute Values in Induction". *Proceedings. of the Sixth International Workshop on Machine Learning*, pp. 164-168,. Morgan-Kaufmann.

Tuning Rules by Cases

Yoshio Nakatani
Industrial Electronics & Systems Lab.
Mitsubishi Electric Corp.
Amagasaki, Hyogo 661, Japan

David Israel
CSLI
Stanford University
Stanford, CA 94305-4115, USA

Abstract

A new method is proposed for combining rule-based and case-based systems, especially in domains in which precise and exceptionless rules are known to be unavailable. When the result of execution of a rule is not satisfactory, the system stores the name of the executed rule, the conditions under which the rule was executed, the evaluation of the execution, the attributes and values to be modified, and hypothesized alternatives, as a case. The next time the rule is to be executed under the same or similar conditions, the relevant attributes and values are temporarily modified, by replacement by their hypothesized alternatives. After a certain number of such experiments, the maintainer of the system can reconstruct the whole rule base by referring to the stored cases. This methodology is implemented as a system, A LA CARTE, in the domain of cooking.

1 The Problem

In many domains such as controlling objects, rules are preferred because of their performance. Such rules, however, may be not precise and exceptionless from the first because the number and variety of contingencies are simply too great to be reduced to algorithmically realizable order [5]. In such domains, we are faced with the problem of trying to construct and apply a rule base in an essentially experimental, trial-and-error manner. One effective way to assist in the construction of the rule base is to acquire knowledge as cases [4, 5]. Although we can not start out with nothing, it is often difficult or not useful to use the cases of other people because the background knowledge is different. We can sometimes refer to some textbook knowledge of the domain as general rules of how to execute certain procedures. Even then, we may start out with what we know to be not very reliable rules of procedure, the successful application of which is contingent on various not clearly foreseen conditions.

Thus, as we construct and revise a rule-base for such an application, we must generate hypotheses about relevant parameters through trial and error. To take an example, in the domain of cooking, we may not know in advance what kind of recipes will please us, nor how various conditions, both external to us---like the weather---and internal---how tired we are---may affect our enjoyment of certain dishes. To obtain more and more detailed and reliable information about all these things, we must run experiments, that is, actually cook and eat various dishes.

2 An Approach to the Problem

Our approach to this problem is to tune the rules step by step and on line by using cases which store improprieties and their hypothetical alternatives of the rules.

When the result of the execution of a rule is not satisfactory, the user stores in the case base the name of the executed rule, the relevant conditions, an overall evaluation, and a representation of what the user judges to be the factors of the rule that account for the unsatisfactory performance together with his/her judgment as to what modifications are required. The next time the rule is to be executed under the same condition, the case base is searched for a relevant case. If there is no such case stored, the rule is executed without modification. If the most relevant case is decided, the relevant elements are modified by reference to the alternatives in the case. If the result of the execution of the modified rule is not satisfactory again, another alternatives are proposed as a new case.

We must consider possible complicated interaction among rules [6]; if we presume that the original rule base is in a consistent state, we don't want to run the risk of rendering it into an inconsistent one. Moreover, revision hypotheses may not be always true. Our method doesn't change the rules in the rule base at all. The rule to be executed is copied into the working memory and the copied rule is temporarily modified by reference to the case. After the modified copy is executed, this copy is removed from the working memory. After enough cases to realize a satisfactory rule base are obtained, the system maintenance people can update the whole rule base.

Although this method has the advantages described above, it does require a flexible mechanism for temporarily modifying rules. We must decide what should be stored in the cases, which attributes and values are to be modified and how to modify the rules by using such cases. We present a solution to these problems below.

Our approach suggests a general architecture. In order to show the effectiveness of this method, we adopt this framework to the domain of cooking. We present the architecture in section 3. In section 4, an example from the cooking domain is presented in some detail. In section 5, we compare our framework with related works.

3 The Architecture

3.1 Rules

Rules are presumed to be represented as shown in **Figure 1** . The first element is the rule name. The second is the list of conditions under which the rule is to be executed. When no conditions are specified, the rule can be applied under any conditions whatsoever. The third and the forth elements are the lists of resources and tools used in executing the rule. The fifth element is the list of procedures to be executed. Each procedure is represented by a list of its order in the sequence of procedures , the type of action, the target upon which the action is executed, and relevant parameters of the action, such as how long it is to be performed. The position of the procedure in the order can be mentioned as an argument in other procedures, in the form: act*n*. Such terms refer to the result of the *n*th step in the list. Each procedure is executed sequentially when the rule name and the conditions match the working memory.

Figure 2 gives an example of a rule in the domain of cooking. This rule shows a recipe for *broccoli with tofu* (see [4]). According to the condition: "hot," this rule can be applied in hot weather. The ingredient in the second list: [r_pepper, piece, 6] means six pieces of red pepper is needed. The third list contains the tools used in executing the recipe. The procedure: [6, stir_fry, [act5, r_pepper], 1] means that the sixth step in this recipe is to stir-fry the result of the fifth step together with the red peppers for one minute.

```
rule( <rule name>,
    [<condition>, ...],
    [<resource>, ...],
    [<tool>, ...],
    [<procedure>, ...] ).
```

Fig.1 Rule representation

```
rule( broccoli_with_tofu,
    [hot],
    [[tofu, lb, 0.5],
     [soy_sauce, tablespoon, 1],
       . . . . . .
     [broccoli, lb, 1],
     [r_pepper, piece, 6]],
    [[bowl],
     [cutting_board],
       . . . . . .
     [flat_spatula]],
    [[1, divide, broccoli, small_flowret],
     [2, boil, [act1, salted_water], 2],
       . . . . . .
     [6, stir_fry, [act5, r_pepper], 1],
     [7, add, act1, act6],
     [8, stir_fry, act7, 3]] ).
```

Fig.2 Example of a rule representation

3.2 Cases

Cases are stored when the evaluation of a rule execution is judged to be not satisfactory. Alternative hypotheses as to which of the elements of the rule are 'to blame for' the unsatisfactory evaluation must be proposed by a human user or a system component. These hypotheses are represented as a combination of *replacement, addition,* and *deletion* of certain ele-

ments of the conditions and/or the procedures of the rule.

Figure 3 presents a case representation, consisting of the case name, the target rule name, the condition list, the evaluation, and a set of quadruples. The condition list may include conditions other than those listed in the rule base, which are the result of the user's judgments as to causal relevance. The last element is a list of quadruples: [*<attr current-val> case-val hyp-alt reason*]. The first element is an attribute-value pair taken from the rule based on the result of the user's judgments as to causal relevance. The second is the actual value of the attribute with which the rule was executed this time; the third is the value to be used next time---as judged by the user, and the fourth is the reason why this value is to be modified, again as hypothesized by the user.

Figure 4 gives an example of the case in the domain of cooking. The target rule is the rule for broccoli with tofu. What this case represents is an occasion on which the user, suffering from a fever, made broccoli with tofu, in accordance with the recipe in the rule base, except that he used 4 pieces of red pepper rather than the required 6. The results were not too satisfactory, let us say 6 on a range of 0-10. The dish was adjudged both too spicy and insufficiently crispy. Moreover, the user hypothesized that his having a fever when he ate the meal played some part in the unsatisfactoriness of the occasion. Finally, the case represents the user's judgment that, given that he has a fever, if he wants broccoli with tofu, the recipe should be altered as follows: cut the amount of a red pepper to 3 pieces and reduce the time for the final mixture to be stir fried to one minute.

```
case( <case name>,
      <target rule name>,
      [<condition>, ...],
       <evaluation>,
      [{ <original attribute-value>,
         <the present value of the attribute>,
         <the hypothesized value of the attribute>,
         <reason for modification> },
         . . . ] ).
```

Fig.3 Case representation

```
case( case4,
      broccoli_with_tofu,
      [fever],
      6,
      [[[r_pepper, piece, 6],
        [r_pepper, piece, 4],
        [r_pepper, piece, 3],
        too_spicy],
       [[8, stir_fry, act7, 3],
        [8, stir_fry, act7, 3],
        [8, stir_fry, act7, 1],
        more_crispy]] ).
```

Fig.4 Example of case representation

3.3 Rule Tuning Algorithm

The central idea of our approach is realized by the following process. The current conditions are represented in the working memory.

Until the problem is solved do:

1. Select the target rule to be executed.
2. Find the case most relevant to the current condition.
3. If such a case as (2) is found, then
   (a) copy the rule, modify the copy using the case, and execute the modified rule; else
   (b) execute the rule.
4. If the copy is executed, then remove the copy.
5. Evaluate the result. If further modifications are needed, these are hypothesized and a new case is created.

The most relevant case is decided as follows: (1) All cases with the target rule name are selected. (2) Among them, the cases whose conditions match the condition of the working memory are selected. (3) If more than one cases is found, the latest is selected. The older cases can be referred to as the history of modification of a rule.

3.4 Rule Tuning Example

Figure 5 shows an example of rule tuning. >> denotes a prompt for user input. The user wants to make broccoli with tofu. He/she has a fever. First, the original recipe (**Figure 2**) is retrieved because the recipe under the condition of having a fever is not found. Next, the system searches for the most relevant case (case 4). The copied rule is modified based on this case. Because the result is judged to be still too spicy, a new case (case 11) is entered, with a hypothesized value of two pieces of red pepper. Note that another hypothesis about the time for the final mixture is kept in the new case.

3.5 Tuning Operators

When the rule is modified temporarily by referring to the relevant case, we allow three kinds of rule modification: *replacement*, *addition*, and *deletion*. In multiple modifications, we execute addition and deletion first and replacement last. This execution order is important because addition and deletion of a resource may be accompanied by modifications of other procedures and such modifications can effect replacement. The targets of modifications involves all elements of the rules. In particular, addition or deletion of a procedure may require updating of the numerical indices of other procedures, especially as they occur in act*n* terms within procedures.

3.6 Evaluation

After executing the rule, the result must be evaluated. This evaluation may be done either by the human user or by

```
Dish >> broccoli_with_tofu.
Condition [list] >> [fever].

!! Original recipe is retrieved !!

** Relevant cases **
  Case name:                case4
  Dish:                     broccoli_with_tofu
  Condition:                [fever]
  Evaluation:               7
    #Target attribute:      [red_pepper, piece, 6]
      Previous value:       6
      Hypothesized value:   3
      Viewpoint:            too_spicy
    #Target attribute:      [8, stir_fry, act7, 3]
      Previous value:       3
      Hypothesized value:   1
      Viewpoint:            more_crispy

** Rule Execution **
  Dish :          broccoli_with_tofu
  Condition :     [fever]
  Ingredients:
                  [tofu, lb, 0.5]
                    .......
                  [r_pepper, piece, 3]
  Procedures:
                  [1, divide, broccoli, small_flowret]
                    .......
                  [8, stir_fry, act7, 1]

Evaluation [0-10] >> 8.
```

```
Case name         >> case11.
Selection
      1. Modify an attribute-value pair
      2. Add a new ingredients
      3. Add a new procedure
      4. Remove an ingredient
      5. Remove a procedure
      6. Information
      7. End
                              >> 1.

Target attribute [list]   >> [r_pepper, piece, 3].
Hypothesized value        >> 2.
Viewpoint                 >> too_spicy.

Selection
      1. Modify an attribute-value pair
                  ........
                              >> 7.

** New case: case11 **
  Dish :                    broccoli_with_tofu
  Condition :               [fever]
  Evaluation:               8
    #Target attribute :     [red_pepper, piece, 6]
      Previous value:       3
      Hypothesized value:   2
      Viewpoint:            too_spicy
    #Target attribute:      [8, stir_fry, act7, 3]
      Previous value:       3
      Hypothesized value:   1
      Viewpoint:            more_crispy
```

Fig. 5 Example of rule modification

a system component. For our cooking examples, we have imagined that the measure of evaluation is single-dimensional and scalar (a range of 0-10). We assume that hypotheses are generated by the user.

3.7 Analogical Problem Solving

If there are no rules or cases which match the current conditions completely, our method executes analogy-based (similarity-based) modifications of the rule. We consider five cases when such analogical modification is needed:

1. When no rules satisfy the conditions of the working memory completely. A rule with similar conditions is selected. Any one of a number of similarity metrics might be used.
2. When no cases satisfy the conditions of the working memory completely. Cases are searched for which match the various components of the conditions. Again various similarity metrics might be used.
3. When there are conditions that remain unsatisfied by any cases. The user can command the system to replace them by similar conditions.
4. When the condition of the working memory says that the necessary resources or tools for executing the relevant rule are not available. Appropriately equi-functional resources are proposed. For example, when broccoli is unavailable, the system might propose *asparagus*---another green vegetable---as a similar resource.
5. When the user specifies the rule name to be executed that is not found in the rule-base. Another rule with a similar name is selected and is executed with the relevant rule elements modified. For example, when a user-specified recipe asparagus_with_tofu cannot be found, *broccoli_with_tofu* is proposed as the similar dish.

The concepts used in the conditions, resources and tools are represented as a list of attribute-value pairs. **Figure 6** shows the resource representation. Here, attributes are ordered according to their importance in classifying them in the domain. When deciding on replacement for an unavailable resource r, the candidate sharing the most attribute-value pairs with r is selected.

resource(<name>, [[<attribute>, <value>], ...]).

Fig.6 Representation of resources and tools

4 Application: A LA CARTE

A LA CARTE (A LeArnable CAse-based RulE TunEr) is a prototype system of our method [7]. Its target domain is cooking. Each recipe is represented in a single rule. The user evaluates the results of rule execution and comes up with rule-modification hypotheses of the resulting dish. In order to support this modification task, **A LA CARTE** offers the history of modification of a rule, which helps the user decide which conditions, resources, tools, and/or procedures are to be modified. **A LA CARTE** is implemented on the engineering workstation written in Prolog.

The basic operation of **A LA CARTE** is as follows: First, the user specifies a dish and the current conditions. Second, **A LA CARTE** searches for a recipe for the dish in the rule-base and presents the recipe. The case base is then searched for cases in which the recipe was executed under the most similar conditions. Depending on the most relevant case, **A LA CARTE** modifies the copied recipe. The user executes the modified recipe, evaluates the result, and perhaps suggests further modifications.

5 Related Work

CHEF is a pure case-based planner in the domain of Szechwan cooking [4]. Its task is to build a new recipe based on the user's request for certain ingredients and tastes. **CHEF**'s recipes are represented as cases which include ingredients and actions. **CHEF** searches for cases relevant to the request, integrates them into a new dish to meet the request, and stores that in a case base. The advantage of **CHEF** is that it has variety of functions which create and adapt new recipes. Its disadvantage is that its ability to revise the original recipes is limited.

There are some types of methods for combining rule-based reasoning and case-based reasoning. **Anapron** supplements the rule-based systems by using cases as a library of exceptions [3]. If there is a contradictory case to the selected rule, the procedures of the case are executed; otherwise the rule is executed. This method requires a less complex problem-solving mechanism than that of **A LA CARTE**. **GREBE** uses cases to reduce the problem of matching specific case conditions with open textured terms in rules to the problem of matching two sets of cases [2]. Rules, in turn, are used for term reformulation and infering facts that are not stated in the case. The advantage of this method is the flexible use of rules and cases. However, these methods require the existence of a body of cases at the beginning of reasoning. Moreover, the cases are not used in the process of rule modification. The method, whose rules are derived from generalized cases [1], requires some primary problem-solving mechanism independent of the rules.

6 Discussion

A new method is proposed for tuning a rule base by cases in domains in which precise and exceptionless rules are known to be unavailable. When the result of execution of a rule is not satisfactory, the system stores the name of the rule, the conditions under which the rule was executed, the evaluation of the execution, the rule element to be modified, and hypothesized alternatives, as a case. The next time the rule is to be executed, the relevant rule elements are temporarily modified, by replacement by their hypothesized alternatives. After a certain number of such experiments, the maintainer of the system can reconstruct the whole rule base by refering to the stored cases. This method is implemented as a system named **A LA CARTE** in the domain of cooking.

We have not presented any experiments. This is really a prototype proof of our concept. Directions for future work involve applying this framework to more complex rule-base, introducing a stronger measures of similarity, and constructing an automatic hypothesizer for rule modification.

References

1. Allen, J. A. and Langley, P. : "A Unified Framework for Planning and Learning", Proc. of Workshop on Innovative Approaches to Planning, Scheduling, and Control. (1990).
2. Branting, L. K. and Porter, B.W. : "Rules and Precedents as Complementary Warrants", Proc. of AAAI-91 (1991).
3. Golding, A. R. and Rosenbloom, P.S. : "Improving Rule-Based Systems through Case-Based Reasoning", Proc. of AAAI-91 (1991).
4. Hammond, K. J. : Case-Based Planning: Viewing Planning as a Memory Task. Academic Press 1989, San Diego.
5. Kolodner, J. L. : "An Introduction to Case-Based Reasoning", Artificial Intelligence Review, 6 (1992).
6. Minsky, M. : "Logical Versus Analogical or Symbolic Versus Connectionist or Neat Versus Scruffy", AI Magazine, SUMMER (1991).
7. Nakatani, Y. and Israel D. : An Architecture for Tuning Rules by Cases, Report No. CSLI-92-173, CSLI, Stanford University (1992).

Combining Case-Based and Model-Based Approaches for Diagnostic Applications in Technical Domains

Gerd Pews and Stefan Wess
University of Kaiserslautern
Dept. of Computer Science
P.O. Box 3049
D-67653 Kaiserslautern, Germany
e-mail: pews,wess@informatik.uni-kl.de

Abstract

Calculating similarity by comparing syntactical features is a quite common approach for case-based reasoning systems for diagnostic applications. In this paper, we present an approach which uses a similarity assessment based on a qualitative model of the technical system under consideration. This provides the capability of classifying the system's behavior, leading to an improved retrieval and case adaptation process. Furthermore, the case-base can be used more efficiently. The knowledge used by this approach is employed in model-based diagnostics; there, qualitative simulation of the technical system leads to a diagnosis. This model-based approach is known to have a very huge search space and thus, to be very expensive in computation. Our approach is an attempt to cut down the search space of model-based diagnostics by using appropriate heuristics: *cases*.

1 Motivation

Human experts are using different kinds of problem-solving strategies and knowledge sources. In research, this is reflected by the development of rule-based, model-based and case-based systems. Rule-based and model-based approaches for diagnostic applications have been a matter of research for a longer period of time till now, whereas the interest on case-based techniques has quite recently increased e.g. CASEY [7], PROTOS [4], CREEK [1] and PATDEX [3]. The main point in case-based reasoning is to decide if certain items (such as objects, situations, problems) are similar or not. Here, similarity means, similarity concerning certain criteria which often are not explicitly described but implicitly determined by the items context. (For instance, among a set of triangles, a blue circle and a red circle rather are considered to be similar, but not among a set of colored circles). In case-based diagnosis, searching for similarity means searching for a case with a solution (diagnosis) that is useful for guiding the search for a solution for the current problem at hand. Unfortunately, usefulness can only be estimated a posteriori and therefore a retrieval process based on this aspect is impossible.

Some CBR systems use a numeric similarity measurement to determine similarity, presupposing that a syntactical similarity of features implies this utility. Syntactical comparison is easy to perform and, in general, seams not to expect much background knowledge. Nevertheless, the assumption that syntactical identity implies utility needs not to be true. In fact, this assumption will only be justified if a large amount of domain knowledge was used in coding the cases. The person who brought the cases from the real world into a syntactic description will have made certain abstractions and selected only a few of all possible features which could be selected. This person has to find a way between over-abstraction on the one hand (which makes it easy to find similar cases, but reduces the information contained in the case and the solution it can offer) and over-specialization on the other hand which enables the system only to find identical cases[1]. So, there are a lot of disadvantages which normally occur in surface-based similarity assessments, e.g.:

- Structural similarity will not be detected. If two objects of the same kind but with different names show similar features, identification will not be possible.
- Solution transformation and adaptation is not supported by the assessment process.
- The formula for calculating the similarity value cannot be controlled and modified easily.

[1] Identity, here of course, means identity on a certain abstraction level. If a situation is coded mentioning, for example, time or the position of the object relative to the sun, identity would lead to never finding a case in the case-base.

- For satisfying results, the case base has to be rather large.
- To reduce retrieval time, the case base should be small.
- Knowing just the similarity value, it is not possible to decide whether a solution is significant or not, i.e. to provide a good explanation.
- Similarity, relevance and abstraction are context-dependent. Even syntactically identical features may have different semantics.

To avoid these well known problems our approach is to combine a structural similarity assessment with a model-based diagnostic system to support case retrieval and adaptation. The importance of detecting structural similarity and therefore supporting case adaptation in technical domains becomes clear when looking at a domain where our approach is being used: technical diagnosis for computerized numerical control (CNC) machines.

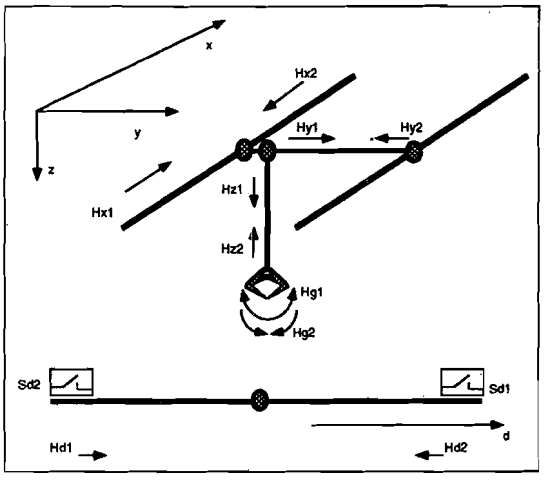


Figure 1: An example from a CNC domain: a grip

In figure 1, an example from this domain is given. It shows a grip which can be moved on bars in the x-, y- and z-direction. Of course, the drive systems for all directions are identical in structure, only the used components will have different names. In fact, there are several possibilities of adapting and transforming one case into another:

1. A symptom a implies another Symptom b. The symptom a is part of Case-1, symptom b is mentioned in Case-2. Adapting cases means to detect that a match between the symptoms a and b in the two cases is allowed.
2. Case-1 describes a fault that occurred when moving the grip in x-direction. Case-2 describes the same fault occurring when moving the grip in z-direction. Here, adaptation of cases means to detect that the parts which are described by the symptoms are similar and can be identified with each other. Capability to perform such an adaptation can dramatically reduce the amount of cases needed in the System. For the grip-domain this means a reduction by factor 3. Considering that grip movements forward and backward will be performed by similar drive systems, the factor rises to 6.
3. Syntactically different values can be semantically identical. In certain situations, variations of a symptom value can be tolerated (e.g. voltage variations between 4.5V and 5.5V), while this might not be possible in other situations.

2 A model-based approach for structural similarity assessment

Before defining a model-based approach to similarity assessment it is necessary to look closely at the diagnostics domain first. To find a diagnosis means to identify faulty behavior of some components. Sometimes this behavior can be observed directly, sometimes this behavior has to be inferred from the state a component is in, showing only the final state which the behavior led to. From this point of

view diagnostics can be interpreted as a kind of classification of components behavior. To estimate its relevance, it is important to know if a certain behavior was intended or not. Unintended does not mean faulty - an electric bulb which stays dark because of a broken wire surely shows unintended behavior though the bulb is not faulty. So, we can figure out some criteria which should be fulfilled for relevant components in similar cases:

Components:	The components themselves should be similar.
Behavior:	The behavior of the components should be similar.
Topological context:	The components should be connected in a similar way.
Intentional context:	The components are expected to behave similar.

Dealing with these criteria requires additional knowledge of the domain, describing the domain up to a certain abstraction level and containing information concerning structure and functionality of the technical system under consideration.

- *Object knowledge:* This knowledge describes the components of which the domain consists of and how these components can be joined to more complex components.
- *Topological knowledge:* This knowledge specifies, how the single components are connected to each other and therefore their causal effects on each other.
- *Functional knowledge:* This knowledge defines how the components can or should behave.
- *Abstraction knowledge:* This knowledge describes how objects and an object's behavior can be described on a more abstract level.

This knowledge is normally used in model-based diagnostics [5, 7]; there, simulation on some abstraction level leads to a solution. This model-based approach is known to have an very large search space and so to be very expensive. Thus, our approach is an attempt to cut the search space of model-based diagnostics by using appropriate heuristics: cases. This does not mean to use a case-based problem solver and a model-based problem solver side by side, but to seamlessly join both approaches into one single approach.

2.1 Assessing similarity between cases, components and behavior

Calculating similarity in our approach bases on the principle that similarity means identity on a higher level of abstraction. The levels of abstraction induced by the above described criteria imply an ordering of similarity: The lower the level of abstraction, the higher the similarity. An example for abstraction levels is: A certain transistor $BC107$ can be viewed as a transistor in general, as a switch, as a machine component and finally as a thing. The process of assessing similarity between a case and a certain situation can be divided into the following steps:

Symptom expansion The given Symptoms have to be interpreted and propagated. If two components are connected with each other, values will be propagated via this connection. Value equivalents on higher abstraction levels are calculated. Finally, the shown behavior will be determined (if possible). For example, if the input of a wire is 3A and the output is 3A, this means that the behavior *transmit* takes place.

Relevance determination The symptom's relevance is estimated. Components which show unintended behavior are more relevant than those with intended behavior. Further, unintended behavior of components which is not based on other components' unintended behavior is of high relevance for the similarity assessment. To identify unintended behavior, the intended behavior of the component is simulated and compared to the actually given behavior of the components.

Retrieval/Hypothesis generating and testing This step will be iterated on several levels of abstraction till satisfying hypotheses have been generated. The abstraction level started with is the lowest (most concrete) one. If retrieval fails, the next higher level will be chosen.

- *Retrieval:* Only relevant values will be used for retrieval. If similar components show similar behavior (similarity here means identity on the current level of abstraction),
- *Hypothesis generating* takes place. Similar parts are identified with each other, using topological knowledge. A fail of hypothesis generating means a fail of retrieval.
- *Hypothesis testing* Test, if the assumed behavior of the hypothesis' component can cause the actually given symptoms.

Test selection The Retrieval step normally generates several different hypotheses; one hypothesis has to be selected (considering statistical data of former failures).

Learning The situation can be added as a case to the case base.

It is obvious that a component will be similar to any other component (at least at the highest abstraction level) and that abstraction levels do not build a simple hierarchy (different abstractions of certain aspects are possible) so that it is not possible to tell if similarity basing on a certain abstraction is higher than abstraction basing on a different abstraction. This process for calculating the preferred level of abstraction can be seen as a search for the *Minimal Common Generalization* [8] of a case and the actual situation with respect to the given model of the technical system.

3 Summary

This approach to similarity assessment and case adaptation has been successfully realized in MoCAS (Model-based Case Adaptation System) which is used in combination with the PATDEX system [11] in the domain of fault diagnostics for CNC machines. The proposed approach provides advantages like: case adaptation and transformation. The results (hypotheses) are all proved by the used model of the technical system under consideration and therefore plausible and consistent. They can be explained using the domain model, describing the causal relations between the components [9], and by the case base, explaining why the current situation is similar to the cases the hypotheses are generated from (similar components, topology, behavior) to make it clear why just a few of all possible hypotheses were picked out. On the other hand, this approach requires a qualitative model of the domain. In the actual implementation MoCAS is able to use the knowledge of the model-based component [10] of the MOLTKE-Workbench [2] for diagnostic applications. In fact such a model which is used by MoCAS is easy to obtain in strong technical domains. In domains where the knowledge of components and their causal interactions is more vague, like in medical diagnosis, the proposed approach is not usable.

References

[1] Agnar Aamodt. Towards Robust Expert Systems that Learn from Experience. In John Boose, Brian Gaines, and Jean-Gabriel Ganascia, editors, *Proceedings of the 3rd European Knowledge Acquisition Workshop EKAW'89*, 1989.

[2] K.-D. Althoff, F. Maurer, S. Wess, and R. Traphöner. MOLTKE - an integrated workbench for fault diagnosis in engineering systems. In S. Hashemi, J.P. Marciano, and G. Gouarderes, editors, *Proc. 4th international conference Artificial Intelligence & Expert Systems Applications (EXPERSYS-92)*, Paris, October 1992. i.i.t.t international.

[3] Klaus-Dieter Althoff, Sabine Kockskämper, Frank Maurer, Michael Stadler, and Stefan Wess. Ein System zur fallbasierten Wissensverarbeitung in technischen Diagnosesituationen. In ÖGAI, editor, *Proceedings 5. Österreichische Artificial-Intelligence Tagung*, pages 65–70. Springer-Verlag, 1989. Igls, Österreich.

[4] R. Bareiss, B. W. Porter, and C. C. Wier. PROTOS: An Exemplar-Based Learning Apprentice. In Yves Kodratoff and Ryszard Michalski, editors, *Machine Learning: An Artificial Intelligence Approach*, volume III, pages 12–23. Morgan Kaufmann, 1987.

[5] R. Davis. Diagnostic reasoning based on structure and function. *AI-Journal 24*, 1984.

[6] Janet L. Kolodner, editor. *Proceedings Case-Based Reasoning Workshop*, San Mateo, California, 1988. DARPA, Morgan Kaufmann Publishers. Clearwater Beach, Florida, USA, May 10–13, 1988.

[7] P. Koton. Reasoning about Evidence in Causal Explanation. In Kolodner [6], pages 260–170. Clearwater Beach, Florida, USA, May 10–13, 1988.

[8] Debbie Leishman. Analogy as a constrained partial correspondence over conceptual graphs. In *Proc. IJCAI*, 1989.

[9] Michael J. Pazzani. *Creating a Memory of Causal Relationships.* Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, 1990.

[10] R. Rehbold. Deriving Causal Rules from Structure Descriptions in a Technical Diagnosis Domain. In *Proceedings of the Workshop on Expert Systems, Avignon*, 1989.

[11] M. M. Richter and S. Wess. Similarity, Uncertainty and Case Based Reasoning in PATDEX. In Robert S. Boyer, editor, *Automated Reasoning - Essays in Honor of Woody Bledsoe*, pages 249–265. Kluwer Academic Publishers, 1991.

A Reflective Architecture for Integrated Memory-based Learning and Reasoning

Enric Plaza Josep-Lluís Arcos

Institut d'Investigació en Intel·ligència Artificial, C.S.I.C.
Camí de Santa Bàrbara, 17300 Blanes, Catalunya, Spain.
{plaza | arcos}@ceab.es

Abstract. In this paper we will discuss the role of case-based reasoning and learning as a tool for integrating different methods of inference and different methods of learning. The Massive Memory Architecture, an experimental framework for experience-based learning and reasoning, is described. Its reflective capabilities are described and we put forth the hypothesis that learning methods are inference methods able to inspect the problem solving process and modify the system itself so as to improve its behavior. Therefore, learning methods require a self-model of the system. Self-models and method implementation are based on conceptual, knowledge-level descriptions of inference.

1 Introduction

In this paper we will discuss the role of case-based reasoning and learning as a tool for integrating different methods of inference and different methods of learning. Case-based reasoning (CBR) systems offer the advantage of an integrated framework for both problem solving and learning. However, every CBR system combines in a peculiar way several specific inference methods and associated learning methods. Research toward a conceptual and computational framework able to encompass disparate CBR systems can be very important for theoretical understanding and practical applications.

In a companion paper [Armengol 93] we show how a conceptual framework like the Components of Expertise [Steels 90] can be used to describe at the knowledge level the reasoning and learning methods of several classic CBR systems. It can be then observed that CBR systems share a common pattern of task/subtask decomposition and they differ by the methods chosen to perform each task and subtask. Needless to say, this election is determined by the kind of application for which the CBR system has been developed. The knowledge level analysis show that CBR systems can be unified at least conceptually, and this we think is independent of the conceptual framework used, that is to say KADS [Wielinga 92] or Generic Task analysis [Chandrasekaran 89] would show the same. Another analysis of this kind is [Aamodt 90].

In this paper we present how this analysis can be implemented in a computational framework that supports task/subtask decomposition, the Massive Memory Architecture (MMA). The MMA is an experimental framework for experience-based learning and reasoning. It is based on memorisation of past episodes of problem solving and in a default behavior that resorts to analogous past cases to solve new situations. This is a default behavior in the sense that it is used when no concrete domain knowledge is available. The analogical inference is modelled as an inference pattern Retrieve / Select / Adapt. This pattern is reified into an analogical inference method object, where different retrieve or select methods can be used. The fact that inference methods are first class objects means that inference methods can be programmed also.

In our approach, learning methods are just inference methods able to inspect the problem solving process and modify the system itself so as to improve its behavior. Therefore a learning system requires reflective capabilities able to self-inspect the system and a theory of the system that specify how it can be modified in order to be improved. In section 3 we develop the self-model used in MMA for case-based learning and reasoning, but first a description of the architecture is necessary.

2 The Massive Memory Architecture

Inference methods in MMA are methods that follow a Retrieve/Select/Adapt pattern. Thus, an inference method is a reification of the basic inference pattern of the architecture. Analogical methods are inference methods that follow a Retrieve by similarity methods and then may have Select methods also of similarity or using domain-based, knowledge-intensive methods. Inheritance is also represented and implemented by explicit inference methods that use a retrieve method that follows a

link (e.g. the `type` link, but other inheritance methods are used, like the `species` link that accesses knowledge in a `homo-sapiens` theory). In this approach, analogy and inheritance are integrated into a uniform architecture.

2.1 Episodic Memory

Every episode of problem solving of MMA is represented and stored as an episode in memory. This is the main point of the reification process: create the objects that can be usable for learning and improving future behavior. MMA records memories of successes and failures of using methods for solving tasks. Since inference methods are also methods, learning can be applied to different types of retrieval methods and selection methods used in the process of searching and selecting sources of knowledge in the form of precedents.

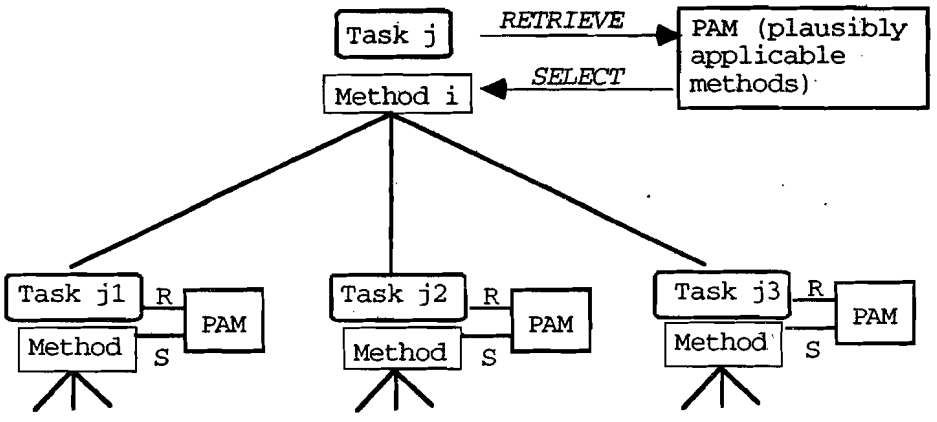


Figure 1. Task decomposition by methods. Any query engages the system into a task. First of all, a method has to be selected from available methods for that kind of task. A method is decomposed into subtasks (queries to other objects) recursively, until some method is direct. Method failure engages backtracking to other available methods.

2.2 Analogical Inference

Analogical methods are inference methods that follow a task decomposition of Retrieve/Select/Adapt. Since different methods can be used for these subtasks, multiple methods of case based reasoning can be integrated. Moreover they can be indexed in different tasks where they are appropriate. The characteristic of analogical methods is that the `Retrieve` method uses a similarity-based method. `Select` methods can also be based on similarity or can be domain-based, knowledge-intensive methods. All inference methods are such because they are able to search for sources from which some knowledge may be retrieved. The types of knowledge retrieved is either domain knowledge (as methods) and experiential knowledge (situations of failure and success). Experiential knowledge is used by MMA to bias the preferences of future actions using precedent cases stored in past episodes. The uniform nature of MMA (every query to an object engages a task) supports learning at all decision points of the system.

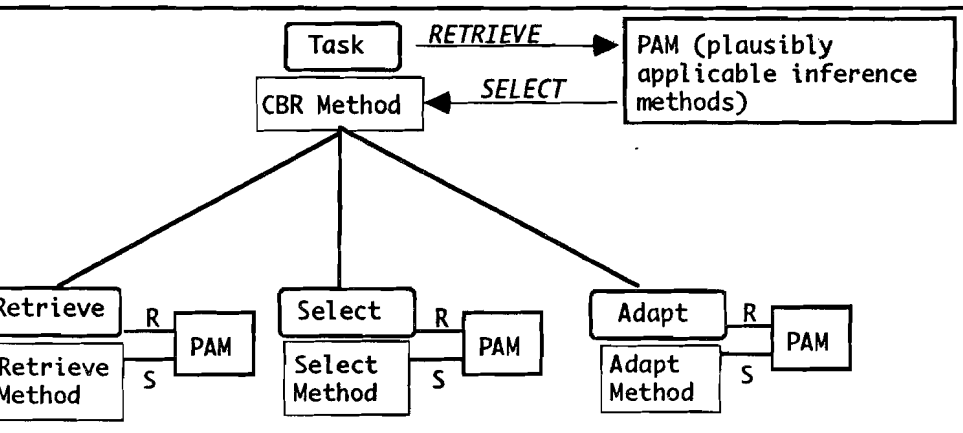


Figure 2. Task decomposition of a CBR method. A CBR method for solving a task is implemented by the specific methods for retrieveing precedent cases, selecting the most appropriate case, and adapting its solution for the current task.

2.3 The Abstract Inference Process

The Massive Memory Architecture is implemented in the frame-based representation language NOOS. The basic inference process of NOOS follows the Retrieve / Select / Adapt pattern. The other notion needed to explain the inference process are *impasses*. When a query like `(>> father of John)` is evaluated a new task is started. Then either

(i) task `"father(John)"` has a method to compute that value like `(>> husband mother of self)`, or

(ii) a `no-method` impasse occurs.

Case (i) is called *spontaneous inference* and occurs at the base level. However, in (ii) the impasse causes NOOS to search at a meta-level for possible methods to use. Impasses are handled by metaobjects, that is to say MMA is an impasse-driven reflective architecture. The architecture specifies which type impasses can appear, and which kind of metaobject will handle them (see Table 2). The `no-method` impasse in a task is handled by its metafunction (set of applicable methods). Applicable methods can be retrieved and selected (maybe trying them out) and the solution is cached in the task-object (see Fig, 3). Every impasse is an opportunity for learning and the reification process creates and stores the objects needed to represent the situation (so that it can be useful in the future). In the task-object example, the information stored is the successful method and the methods that failed.

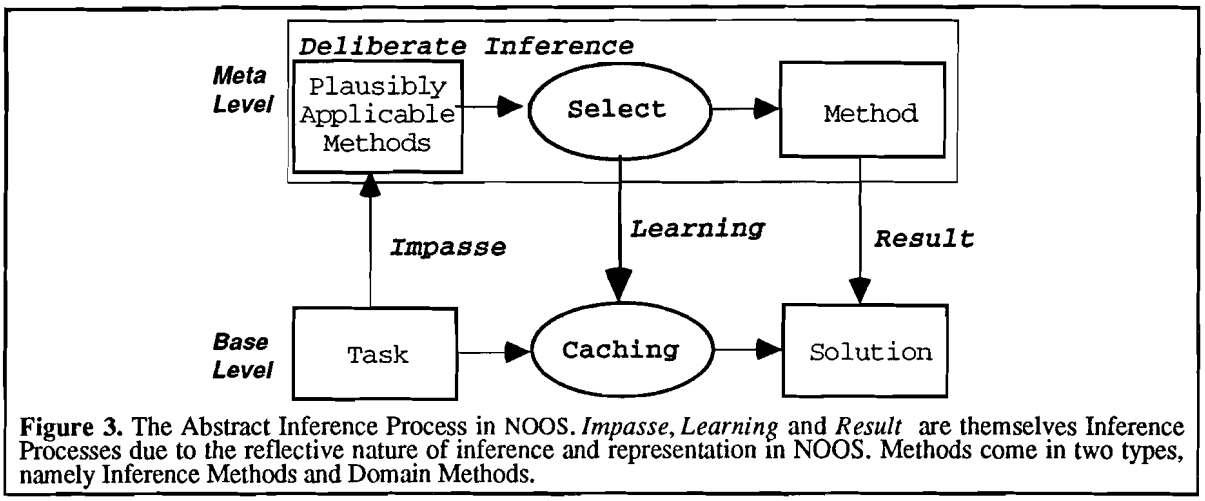


Figure 3. The Abstract Inference Process in NOOS. *Impasse*, *Learning* and *Result* are themselves Inference Processes due to the reflective nature of inference and representation in NOOS. Methods come in two types, namely Inference Methods and Domain Methods.

The inference can be more complex, e.g. maybe the applicable methods for a task `(>> father of John)` are unknown. This is a new kind of impasse: the `no-metafunction` impasse and is handled by the metatheory of John that possesses inference methods able to search, retrieve and select methods in other objects. Currently several inheritance and case-based methods are implemented as inference methods. The inference theories specify when each method is applicable or likely to be useful in solving a task.

TABLE 2. Impasse and response to impasses

Impasse Type	Handled by	Metaobject	Processing Level
No method for F(U)	Meta("F(U)")	Metafunction	Domain Methods
Multiplicity of Methods	Select Theory	Strategic Cliché	Preference Methods
No metafunction for F(U)	Meta("U")	Inference Theory	Inference Methods
Multiplicity of Infer. Methods	Select Theory	Strategic Cliché	Preference Methods

The point to notice is that the uniformity of NOOS treats all situations in the same way. Table 2 summarizes the impasses recognised by NOOS and the corresponding metaobject to handle them. As in Soar, every impasse arises from lack of knowledge: either because the system does not know what to do, or it has several possibilities to act and has to decide among them. The first type of impasses is handled by inference methods that know how to retrieve sources of knowledge. Multiple possibilities are handled by strategic clichés, objects that know about preferencing and selecting among choices. The long paper in the proceedings will include an example of case-based and explanation-based reasoning for a technical diagnosis task.

3 Reflection and Self-models

The *reflection principles* specify the relationship between a theory T and its meta-theory MT. The *upward principles* specify the reification process that encodes some aspects of T into ground facts of MT. That is to say, reification constructs a particular model of T in the language used by MT. The nature of reification and the model constructed is open, i.e. it depends on the purpose for which the coding is made. We will use in MMA a knowledge-level model of task/method/theory decomposition (explained in §2) as a meta-model of the base-level inference. We follow a framework similar to the Components of Expertise [Steels 90] and the task-decomposition framework of [Chandrasekaran 89]. However, we do not follow them strictly, except in the general idea of using as "elements of inference" goals, methods, and theories. A similar approach is taken in [Akkermans et al 93] where the meta-model is the KADS modelling framework [Wielinga 92] for expert systems. The meta-theory contents knowledge that allows to deduce how to extend this model deducing new facts about it. This deduction process is called meta-level inference, and the content of this theory is again specific to the purpose at hand (the meta-theory is indeed no more than a theory). Finally, *downward principles* specify the reflection process that given a new, extended model of T has to transform the theory T to a new theory T' that complies to that new model. A more detailed explanation of the reflective principles and of the semantics of NOOS can be found in [Plaza 92].

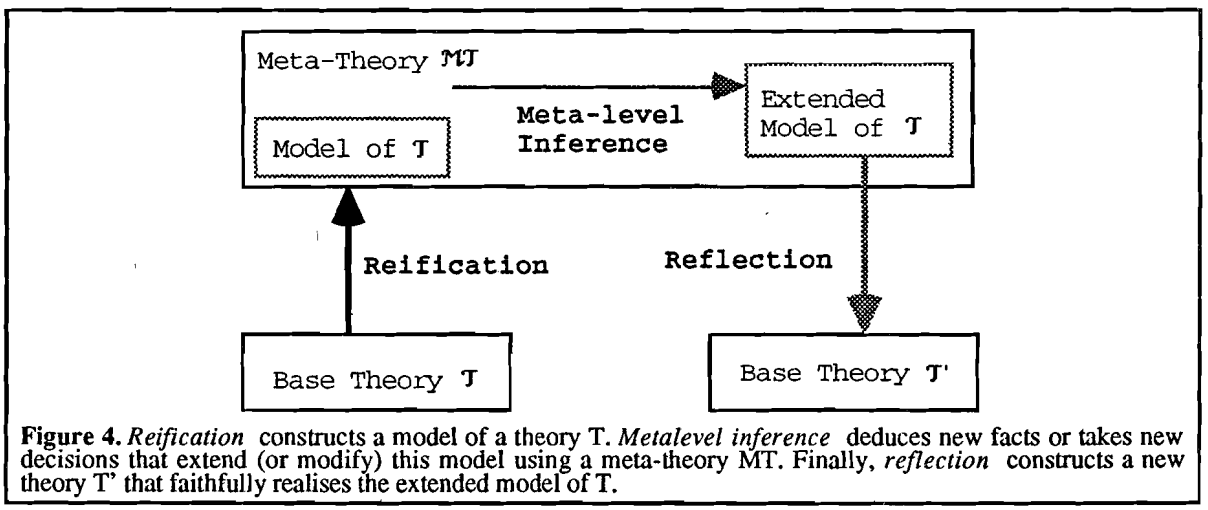


Figure 4. *Reification* constructs a model of a theory T. *Metalevel inference* deduces new facts or takes new decisions that extend (or modify) this model using a meta-theory MT. Finally, *reflection* constructs a new theory T' that faithfully realises the extended model of T.

Our hypothesis is that different types of learning methods would require different self-models of the architecture. The current implementation of MMA has a model of the methods used *for each task*: methods that have been proposed (by an inference method), methods that have been tried but failed, and the method that has succeeded. This information is indexed therefore for each slot-query made and is stored in an object called `task-object`. In the following we will use quotes "X" to designate the reification of X. For instance:

`Age(John)` denotes the query `(>> age of John)`

whose result is 32 years, while

`"Age(John)"` denotes the query `(>> reify (>> age of John))`

i.e. the object reifying the task `(>> age of John)` whose print-name is:

`#<task-object [age of John]>`

The task-object reifies the current state of the process solving the task; specifically the system can know the following about the current state:

```
Access-Name("Age(John)")        =>    "Age"
Domain("Age(John)")             =>    #<John>
Method("Age(John)")             =>    #<Method Age-method-3>
Failed("Age(John)")             =>    #<Method Age-method-5>
Referent("Age(John)")           =>    #<32-Years>
```

This self-model is used by inference methods to retrieve and transfer the metafunction (containing the available methods) from a task solved into a precedent case to a task in the present problem, and for inferring preferences over method selection based on their success or failure in those precedents. For instance, the MMA can obtain the method that successfully computed the age of John using this query:

```
(>> method reify (>> age of John))        => #<Method Age-method-3>
```

Other learning methods that we are incorporating to MMA (see [Plaza 93]) use this self-model but also require its extension. This is as expected because of our Self-model Hypothesis implies that every learning method may need to know different aspects of the architecture. We are then in a process where an analysis of those learning methods elucidate which aspects of NOOS that are hidden or internal to its implementation are to be reified and made accessible to the architecture.

4 Related Work

Our work on architectures is related to cognitive architectures like SOAR [Newell 90], THEO [Mitchell 91], and PRODIGY [Carbonell 91]. At first sight, MMA language resembles THEO since NOOS is a frame language with caching, TMS, and "available methods" for slots. However, THEO does not provide a clear metaobject definition, does not reason about preferences over methods, and does not incorporate analogical reasoning or explicit inference methods. At a deeper level MMA resembles Soar in that MMA is a uniform, impasse-driven architecture with a built-in learning method. The differences are that spontaneous learning here is episode memorization and that our "learning as metalevel inference" hypothesis shapes another approach to inference and learning by the use of reification, self-models and the explicit representation of inference methods.

The introspective use of meta-explanations in Meta-AQUA [Ram 92] is also related to MMA approach that exploits the reflective approach to learning. Meta-AQUA is not impasse-driven but proposes a mapping between classes of situations and learning methods that can improve the system. Meta-Router [Stroulia 92] combines planning and case-based reasoning in a task-decomposition framework (based on [Chandrasekaran 89]) and defines a typology of errors and methods for repair.

Related work on reflection is [Kiczales 91], [Giunchilia 90], and [Smith 85], and specially related viewpoints of inference-level reflection are projects like REFLECT and KADS-II [Akkermans et al 93]. Meta-level architectures have been used for strategic reasoning [Godo 89] [Lopez 93], for non-monotonic reasoning [Sierra 93] [Treur 91], and for modelling expert systems [Akkermans et al 93]. Precedents on using reflection for learning are [Ram 92][Lopez 93]. Our current NOOS language is to be considered a descendant of languages RLL-1 [Lenat 80] and KRS [van Marcke 87].

Related work on knowledge-level modelling of AI systems includes the Commet (or components of expertise) framework [Steels 90] , and the KADS methodology [Akkermans et al 93]. Our approach is closer to the Commet in that the ontology of models, tasks and methods proposed by Commet is related to MMA's ontology of theories, methods and tasks. However, NOOS considers two layers: base-level domain theories and methods, and meta-level inference theories and methods, while the Commet approach is not reflective and only is concerned with the domain layer. This is reasonable, since Commet is intended as *a prescriptive framework* for expert systems where all options searched for in MMA are dictated by the expert's knowledge through the process of knowledge engineering. Although this may involve lack of flexibility in general, it has evident advantages regarding efficiency in most expert system applications. The KADS methodology is much more different but they have used a reflective framework to describe the KADS four-layer architecture. Their reflective framework, called "knowledge-level reflection" uses the KADS model to specify the system self-model of structure and process, very much like our inference-level model of theories, tasks, and methods allows MMA to have a self-model. However, neither Commet nor KADS have been used to perform learning tasks, and in fact MMA is the first attempt to apply knowledge level analysis to learning tasks and to develop a computational architecture that embodies that approach.

Acknowledgements

The research reported on this paper has been developed at the IIIA inside the Massive Memory Project funded by CICYT grant 801/90.

References

[Aamodt 90] Aamodt, A., Knowledge-intensivecase-based reasoning and learning. *Proc. ECAI-90*, Stockholm, August 1990.

[Akkermans et al 93] Akkermans, H., van Harmelen, F., Schreiber, G., Wielinga, B., A formalisation of knowledge-level model for knowledge acquisition. *Int Journal of Intelligent Systems* forthcoming.

[Armengol 93] Armengol, E. and Plaza E., Analyzing case-based reasoning at the knowledge level. *European Workshop on Case-based Reasoning EWCBR'93 Preprints.*

[Carbonell 91] Carbonell, J. G., Knoblock, C. A., Minton, S., Prodigy: An integrated architecture for planning and learning. In K Van Kehn (Eds.), *Architectures for Intelligence.* Lawrence Erlbaum Ass., Hillsdadale, NJ, 1991.

[Chandrasekaran 89] Chandrasekaran, B., Task structures, knowledge acquisition and machine learning. *Machine Learning* 2:341-347, 1989.

[Giunchilia 90] Giunchilia, F., and Traverso, P., Plan formation and execution in an architecture of declarative metatheories . *Proc of META-90: 2nd Workshop of Metaprogramming in Logic Programming..* MIT Press, 1990.

[Godo 89] Godo, L., López de Màntaras, R., Sierra, C., Verdaguer, A., MILORD: The architecture and the management of linguistically expressed uncertainty. *Int. J. Intelligent Systems,* 4:471-501, 1989.

[Lenat 80] Greiner, R., Lenat, D. RLL-1: A Representation Language Language, HPP-80-9 Comp. Science dept., Stanford University. Expanded version of the same paper in *Proc. First AAAI Conference.,* 1980.

[Kiczales 91] Kiczales G., Des Rivières J., Bobrow D. G., *The Art of the Metaobject Protocol,* The MIT Press: Cambridge, 1991.

[Lopez 93] López, B. and Plaza, E., Case-based planning for medical diagnosis, In Z Ras (Ed.) *Methodologies for Intelligent Systems.* Lecture Notes in Artificial Intelligence, 689, p. 96-105. 1993. Springer-Verlag

[Mitchell 91] Mitchell, T.M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., Schlimmer, J. C. , Theo: a framework for self-improving systems. In K Van Lenhn (Ed.) *Architectures for Intelligence.* Laurence Erlbaum, 1991.

[Newell 90] Newell, A., *Unified Theories of Cognition.* Cambridge MA: Harvard UniversityPress, 1990.

[Plaza 92] Plaza, E, Reflection for analogy: Inference-level reflection in an architecture for analogical reasoning. *Proc. IMSA'92 Workshop on Reflection and Metalevel Architectures,* Tokyo, November 1992, p. 166-171.

[Plaza 93] Plaza, E. Arcos J. L., Reflection and Analogy in Memory-based Learning, *Proc. Multistrategy Learning Workshop.,* 1993. p. 42-49.

[Ram 92] Ram, A., Cox, M .T., Narayanan, S., An architecture for integrated introspective learning. *Proc. ML'92 Workshop on Computational Architectures for Machine Learning and Knowledge Acquisition,* 1992.

[Sierra 93] Sierra, C., and Godo, L. (to appear) Specifying simple scheduling tasks in a reflective and modular architecture. In J Treur and T Wetter (Eds.) *Formal Specifications Methods for Complex Reasoning Systems,* Ellis Horwood.

[Smith 85] Smith, B. C., Reflection and semantics in a procedural language, In Brachman, R. J., and Levesque, H. J. (Eds.) *Readings in Knowledge Representation.* Morgan Kauffman, California, 1985, pp. 31-40.

[Steels 90] Steels, L., The Components of Expertise, *AI Magazine,* Summer 1990.

[Stroulia 92] Stroulia, E. and Goel, A. K., An architecture for incremental self-adaptation. *Proc. ML-92 Workshop on Computational Architectures for Supporting Machine Learning and Knowledge Acquisition.* July 1992: Abeerdeen, Scotland.

[Treur 91] Treur, J., On the use of reflection principles in modelling complex reasoning. *Int. J. Intelligent Systems* , 6:277-294, 1991.

[van Marcke 87] van Marcke, K., KRS: An object-oriented representation language, *Revue d'Intelligence Artificielle,* 1(4), 43-68, 1987.

[Wielinga 92] Wielinga, B., Schreiber, A., Breuker, J., KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition* 4(1), 1992.

Using Case-Based Reasoning to Focus Model-Based Diagnostic Problem Solving

Luigi Portinale, Pietro Torasso, Carlo Ortalda, Antonio Giardino

Dipartimento di Informatica - Universita' di Torino
C.so Svizzera 185 - 10149 Torino (Italy)

1 Introduction

The use of Case-Based Reasoning (CBR) plays a fundamental role in many important AI tasks like diagnostic problem solving [8] or planning [5], since in many situations it can mimic the capability of human experts in solving a new case by retrieving similar cases solved in the past. The suitability of CBR to solve complex problems has been widely discussed in the last few years and this led to the combination of case-based reasoning with more traditional problem solving approaches like *rule-based* [2], *prototypical* [12] and *model-based reasoning* (MBR) [4, 10]. In domains where a *strong* model is present (i.e. where a precise domain theory is available), the use of CBR could seem less obvious, however it can still provide advantages when the precise computation of a solution is very complex; this has been studied in tasks like design [4], planning [7] and diagnosis [10]. The identification of previously analyzed problems can be a useful tool for improving the performance of a model-based system by using experience in problem solving. There are two basic possibilities in combining CBR and MBR: 1) CBR is the main problem solving method and MBR is just used to provide guidance to it; 2) CBR is used to focus MBR in the attempt to augment the basic mechanisms of MBR by taking experience into account.

In this paper we will concentrate on the second aspect and in particular on adaptation criteria that can be used in a diagnostic system combining case-based and model-based reasoning. Such adaptation criteria strictly rely on well-defined formal notions of diagnostic problem and diagnostic solution and their adoption can be viewed as a focusing technique for the model-based inference engine[1].

2 Outline of System Architecture

In the diagnostic system we can identify the following basic components: (1) a case memory with an E-MOP-based organization of cases [9]; each case represents a diagnostic problem already solved and it is composed of a set of atoms *feature*(*value*) together with the solution of the problem; (2) a module able to store and retrieve cases from the case memory and to evaluate the degree of match between the current case to be solved and the retrieved ones; (3) a knowledge base, represented through a *causal model* identifying the faulty behavior of the system to be diagnosed; (4)

[1] Another important aspect that will not be discussed here concerns the organization of the case memory that can greatly influence the system performance (see [1] for a discussion of this aspect in the use of the CASEY system).

a model-based reasoner able to perform diagnostic reasoning on the causal model in the form of *abduction with consistency constraints* [3]; (5) a module performing adaptation on retrieved solutions and able to invoke the model-based reasoner if adaptation criteria fail to provide a solution.

The diagnostic system, when presented with a new case, first invokes the case-based reasoner in order to retrieve the most similar cases solved in the past and then it tries to use the solutions of retrieved cases in order to focus the model-based reasoner in the search for the actual solution. The emphasis of the paper is on the adaptation strategies working on the solutions retrieved from the case memory.

Let us briefly discuss the causal model formalism, while in the next section we will address the problem of the formal characterization of diagnostic problems upon which the model-based reasoner performs its task. A causal model is composed by a set of logical formulae which express different kinds of relationships among entities belonging to different types[2]. We identify the following entities: *states* represent non-observable internal states of the modeled system; *findings* represent observable parameters (manifestations) in the modeled system and are the features that are used to characterize cases; *initial_causes* represent the initial perturbations (initial states) that may lead the system to a given behavior. Each one of these entities is characterized by a set of admissible values so that we can identify different instances of them. Two main types of relationships are defined in the model: *causal* relationship represents a cause-effect relation among states, while *ham* (has as a manifestation) relationship is an ordered relation from a *state* S to a *finding* M and represents the fact that the finding M is an observable manifestation of the internal state S. Relationships can be either necessary or possible. In the second case they are modelled introducing a new entity, named *assumption*, that is put in conjunction with the rest of the precondition and represents the incompletness in the specification of the relation [11].

3 Characterization of Diagnostic Problems

In [3] a formal theory of model-based diagnosis is proposed from a logical point of view; this theory defines a logical spectrum of definitions able to capture classical notions of model-based diagnosis, i.e. *consistency-based* and *abductive diagnosis* (see [3] for more details). In the present work we rely on such a theory in order to precisely define a notion of diagnosis on causal models and exploiting such a framework in the adaptation of a retrieved solution. A **diagnostic problem** DP can be described as a triple $< T, HYP, < \Psi^+, \Psi^- >>$, where: T is the set of logical formulae constituting the causal model; HYP is a set of ground atoms denoting the initial causes in terms of which diagnostic hypotheses have to be expressed; Ψ^+ is a set of ground atoms denoting the set of findings that must be accounted for in the case under examination; Ψ^- is a set of ground atoms denoting the set of findings that are known to be false in the case under examination.

It follows from this definition that if OBS is the set of all the observed data in the current case, $\Psi^+ \subseteq OBS$ while Ψ^- will contain, for each observed finding, all

[2] The formalism is actually more structured than as presented here (see [11]); we will sketch here only what is relevant for our discussion.

the instances of such a finding that have not been observed. Since we abstract from time, we impose that a finding cannot have more than one (normal or abnormal) value. This means that a conjunction of atoms representing different instances of the same entity of the causal model yields an inconsistency, so the consistency check is done through the set Ψ^-.

Given a diagnostic problem $DP=< T, HYP, < \Psi^+, \Psi^- >>$, a set $H \subseteq HYP$ is a solution for DP (alternatively an explanation for the observations) if and only if:

$$\forall m \in \Psi^+ \;\; T \cup H \vdash m \quad \text{and} \quad \forall n \in \Psi^- \;\; T \cup H \nvdash n$$

This means that H has to account for all observations in Ψ^+, while no atom in Ψ^- must be deduced from H. It should be clear that a solution H identifies a ground causal chain on the causal model T, starting from the initial causes mentioned in H and containing all their causal consequences. Such a chain is stored in the case structure and is used as the starting point of the model-based inference engine when the case is retrieved.

4 Adaptation Strategies

The goal we pursue in adding a case-based component to a model-based reasoner concerns the possibility of guiding the latter in the search for a solution to a new problem, by reminding solutions to similar problems already solved. Unless the unusual situation when the case under examination is characterized by exactly the same features of the retrieved one, the domain theory is invoked to check whether the retrieved solution is suitable for the case under examination. In particular, we precisely characterize the notion of "suitability" by adopting the formal notion of consistency of a diagnostic solution; this corresponds to put into set Ψ^-, for each observed finding, every instantiation of such findings different than the observed one.

If consistency is verified, then the retrieved solution can be used as potential solution for the new case under examination, unless the user requires that some findings, that are not covered by the retrieved solution, have to be covered in the current case (i.e. they have to be put into Ψ^+). In this situation or in the case when the consistency check fails, adaptation strategies are needed in order to single out a solution taking into account all the requirements (both consistency and covering).

We identify some basic adaptation mechanisms such that the whole adaptation process can be obtained by suitably invoking them; such mechanisms can be viewed as processes of removing inferences that are responsable for inconsistency and processes building explanations for data to be covered. We will describe how these mechanisms work by providing two simple examples (a more formal treatment independent on the particular example will be included in the final version of the paper).

Let us consider the following causal model T representing a small fragment of a more detailed model in the domain of car faults;

$causes((\alpha_1, eng\_mileage(betw\_50000\_and\_100000\_km)), piston\_ring\_wear(moder))$
$causes((\alpha_1, eng\_mileage(more\_than\_100000\_km)), piston\_ring\_wear(severe))$
$causes(piston\_ring\_wear(moder), oil\_consumpt(low))$
$causes(piston\_ring\_wear(severe), oil\_consumpt(high))$

$causes((\alpha_2, eng\_mileage(betw\_50000\_and\_100000\_km)), piston\_wear(moder))$
$causes((\alpha_2, eng\_mileage(more\_than\_100000\_km)), piston\_wear(severe))$
$causes(piston\_wear(moder), oil\_consumpt(low))$
$causes(piston\_wear(severe), oil\_consumpt(high))$
$causes(oil\_consumpt(low), oil\_lack(medium))$
$causes(oil\_consumpt(high), oil\_lack(high))$
$causes((\alpha_3, key(turned\_on)), engine(on))$
$causes((\alpha_4, road\_condition(uneven), ground\_clearance(low)), oil\_sump(holed)$
$causes(oil\_sump(holed), oil\_lack(high))$
$causes((oil\_lack(medium), engine(on)), engine\_temp(high))$
$causes((oil\_lack(high), engine(on)), engine\_temp(very\_high))$

$ham(piston\_ring\_wear(moder), state\_of\_piston\_rings(worn)$
$ham(piston\_ring\_wear(severe), state\_of\_piston\_rings(very\_worn))$
$ham(oil\_consumpt(low), exhaust\_smoke(grey))$
$ham(oil\_consumpt(high), exhaust\_smoke(black))$
$ham(piston\_wear(moder), state\_of\_pistons(worn)$
$ham(piston\_wear(severe), state\_of\_pistons(very\_worn))$
$ham(oil\_lack(medium), oil\_warning\_light(yellow))$
$ham(oil\_lack(high), oil\_warning\_light(red))$
$ham(engine\_temp(high), temp\_indic(yellow))$
$ham(engine\_temp(very\_high), temp\_indic(red))$
$ham(oil\_sump(holed), hole\_in\_oil\_sump(present))$

A causal relation is represented by a *causes* predicate whose first argument represents its precondition involving a conjunction of states, initial_causes and assumptions (indicated with α_i) and the second argument represents the effect. A "ham" relation is represented by a *ham* predicate; the first argument of a *ham* predicate is a state instance whose observable manifestation is represented by the second argument.
In this causal model the set of hypotheses HYP consists of the ground initial causes $eng\_mileage(betw\_50000\_and\_100000\_km)$, $eng\_mileage(more\_than\_100000\_km)$, $key(turned\_on)$, $road\_condition(uneven)$, $ground\_clearance(low)$ and the assumptions α_1, α_2, α_3, α_4.

Example 1. Let us suppose that the case under examination is characterized by the following observations:

$OBS_1 = \{exhaust\_smoke(black), temp\_indic(red), oil\_warning\_light(red)\}$

Let us also suppose to retrieve the following case from the case memory:

$\{state\_of\_pistons(very\_worn), exhaust\_smoke(black), oil\_warning\_light(red)\}$

with associated solution consisting in the conjunction of the ground initial cause $eng\_mileage(more\_than\_100000km)$ and the assumption α_2. From this conjunction and from the domain theory it is easily to derive $piston\_wear(severe)$ (whose manifestation is $state\_of\_pistons(very\_worn)$), $oil\_consumpt(high)$ (whose manifestation is $exhaust\_smoke(black)$) and $oil\_lack(high)$ (whose manifestation is $oil\_warning\_light(red)$).
Since the manifestations in the retrieved case differ from those in OBS_1, consistency check occurs; consistency is checked by using the model T and by putting into Ψ^- the following manifestation instances (i.e. all the manifestations that are alternatives

with respect to the observed ones): *exhaust_smoke(normal)*, *exhaust_smoke(grey)*, *oil_warning_light(normal)*, *oil_warning_light(yellow)*, *temp_indic(green)*, *temp_indic(yellow)*. It is easy to see that consistency check succeds, therefore the retrieved solution can be considered a solution also for the case described by OBS_1 in case the user does not require stronger notion of explanation based on covering.

Example 2. The role of adaptation strategies is made clear by the following example where we assume the case under examination to be characterized by the following observations:

$OBS_2 = \{\neg state\_of\_pistons(very\_worn), exhaust\_smoke(black), temp\_indic(red), oil\_warning\_light(red)\}$

Let us also suppose that the case retrieved from the case memory is the same of Example 1. We can immediately notice that the solution of the retrieved case (i.e. $\{eng\_mileage(more\_than\_100000km), \alpha_2\}$)is not consistent with OBS_2, since it derives the manifestation *state_of_pistons(very_worn)* that is negated in the case under examination; for this reason adaptation is required.
Let us suppose that the set Ψ^+ is formed by *exhaust_smoke(black)*, *temp_indic(red)* and *oil_warning_light(red)*; the adaptation strategy first tries to disprove the causal chain having *state_of_pistons(very_worn)* as a final conclusion. In particular, by removing the assumption α_2, the state *piston_wear(severe)* and the manifestation *state_of_pistons(very_worn)* are no longer supported and consistency is reestablished. Since *oil_consumpt(high)* is necessary to account for manifestation *exhaust_smoke(black)* and it is not supported after removing α_2, adaptation mechanisms have to find out an alternative cause for it.
Looking at the causal model, *piston_ring_wear(severe)* can be used to support *oil_consumpt(high)* and then, by adding assumption α_1, part of the retrieved solution (i.e. *eng_mileage(more_than_100000km)*) can be reused.
Notice that no additional work is needed in order to cover *exhaust_smoke(black)* and *oil_warning_light(red)* which are the findings that are common to both cases. Finally, in order to complete the adaptation, we have to find a cause accounting for *temp_indic(red)* which is present in the case under examination and not in the retrieved one.
This is accomplished by assuming the initial_cause *key(turned_on)* and the assumption α_3 which allows one to infer *engine(on)* that in conjunction with *oil_lack(high)* allows one to derive *engine_temp(very_high)* that explains *temp_indic(red)*.
In conclusion, the solution to the current case is represented by the initial causes *eng_mileage(more_than_100000km)*, *key(turned_on)* and the assumptions α_1, α_3.

Notice that the adaptation of the retrieved solution saves significant amount of work with respect to a diagnostic process that do not exploit retrieved cases. This saving occurs not only when the solutions of the actual and the retrieved case are quite similar, but also when they have significant differences. Example 2 shows that the retrieved solution can be actually used as a focusing mechanism even when the differences in the features characterizing retrieved and current case have a significant impact on the solution of the current case. In such an example, if the solution had to be computed without exploiting case-based mechanisms, the diagnostic system would reach the same conclusion, but the computational effort would be significantly greater because the system had no guide in choosing among multiple alternatives

present in the causal model.

5 Discussion

The system described in the present paper is similar in some aspects to the CASEY system [10], however a major difference concerns the fact that we rely on a well-defined characterization of diagnostic problems and on a precise and general theory of model-based diagnosis. The paper reports an ongoing research; one of the topics that is currently actively investigated concerns the design of opportunistic control strategies for deciding how far it is worth to proceed in adapting the tentative solution of a retrieved case with respect to the solutions provided by other retrieved cases. Such control strategies should decide to abandon adaptation in some special cases when most of the retrieved solution has to be refused. In such situations the "pure" model-based reasoner should be in charge of the computation of the solution.

References

1. D.S. Aghassi. Evaluating case-based reasoning for heart failure diagnosis. Technical report, Dept. of EECS, MIT, Cambridge, MA, 1990.
2. P.P. Bonissone and S. Dutta. Integrating case-based and rule-based reasoning: the possibilistic connection. In *Proc. 6th Conf. on Uncertainty in Artificial Intelligence*, Cambridge, MA, 1990.
3. L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991.
4. A. Goel. Integration of case-based reasoning and model-based reasoning for adaptive design problem solving. Technical report, (PhD Diss.) Dept. of Comp. and Inf. Science, Ohio Univ., 1989.
5. K.J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, 1989.
6. W. Hamscher, L. Console, and J. de Kleer. *Readings in Model-Based Diagnosis.* Morgan Kaufmann, 1992.
7. E.K. Jones. Model-based case adaptation. In *Proc. AAAI 92*, pages 673–678, San Jose', 1992.
8. J. Kolodner and R. Kolodner. Using experience in clinical problem solving: Introduction and frameowrk. *IEEE Trans. on Systems, Man and Cybernetics*, 17(3):420–431, 1987.
9. J.L. Kolodner. *Retrieval and Organization Strategies in Conceptual Memory: a Computer Model.* Lawrence Erlbaum, 1984.
10. P. Koton. Using experience in learning and problem solving. Technical Report MIT/LCS/TR-441, MIT, Cambridge, MA, 1989.
11. P. Torasso and L. Console. *Diagnostic Problem Solving: Combining Heuristic, Approximate and Causal Reasoning.* Van Nostrand Reinhold, 1989.
12. P. Torasso, L. Portinale, L. Console, and M. Casassa Mont. Approximate reasoning in a system combining prototypical knowledge with case-based reasoning. In L.A. Zadeh and J. Kacprzyk, editors, *Fuzzy Logic for the Management of Uncertainty.* John Wiley & Sons, 1992.

Integrating Rule-Based and Case Based Reasoning with Information Retrieval: The IKBALS Project.

John Zeleznikow[1] Daniel Hunter[2] George Vossos[3]

1. Introduction

Over the past decade there has been a growing emphasis on reasoning from experience (case based reasoning). It is our view that intelligent systems need to reason with both rules (given as in statutes, or modelled as in heuristics) and experience (such as legal precedents or medical cases). Because medicine and the law provide excellent examples of the need to integrate rule-based and case-based reasoning, we use both of them as our application domains.

Whilst this project is concerned with multi-modal reasoning in legal knowledge based systems, most of the issues we discuss can easily be generalised to application domains other than law. Indeed the project can be more accurately represented as one concerned with integrating rule-based reasoning, case-based reasoning and intelligent information retrieval.

The main features of our integrated system are:

- The use of cooperating agents;
- The use of an Application Programming Interface to act as a bridge between the agents in the IKBALS III system and the Artificial Intelligence kernel in the Knowledge Base;
- The use of a customised induction algorithm that generates the indices into the case base;
- The use of background information to supplement the induction process;
- A method for converting the decision tree produced by the induction algorithm into quantitative knowledge based rules;
- The heuristics used to justify explanations;
- The query facility that enables users to investigate the relationship between cases and arguments in the system.

These features are pertinent to the construction of general integrated reasoning systems. A paper discussing our work on the integration of case-based reasoning and rule-based reasoning in health care planning can be found in the proceedings of this conference.[4]

The goal of this research project is to investigate ways of reducing the problems associated with modelling law using a strictly heuristic rule-based expert system approach. A heuristic rule-based legal expert system suffers from many problems including explanation deficiencies and control. Our approach to modelling legal reasoning involves integrating a case-based reasoning

---

1 Database Research Laboratory, Applied Computing Research Institute, La Trobe University, Bundoora, Victoria Australia 3083 and INFOLAB, Katholike Universiteit Brabant, Tilburg, Netherlands. E-mail address: johnz@latcs1.lat.oz.au

2 Law School, University of Melbourne, Parkville Victoria, Australia 3052. E-mail address: dah@rumpole.law.unimelb.edu.au

3 Database Research Laboratory, Applied Computing Research Institute, La Trobe University, Bundoora, Victoria Australia 3083. E-mail address: vossos@latcs1.lat.oz.au

4 Bradburn, C. and Zeleznikow, 'The Application of Case Based Reasoning to the Task of Health Care Planning'.

module into a knowledge-based system. Briefly, this multi-modal approach to the problem of modelling legal reasoning has the following advantages over a strict rule-based approach:

- It improves the problem solving performance of a rule-based reasoner. Case-based reasoning can assist a legal rule-based reasoning module represent and reason with open textured statutory predicates. Open textured legal predicates contain questions that cannot be structured in the form of production rules or in logical predicates and which require some legal knowledge on the part of the user in order to answer.

- It improves the explanation facilities of the system. Case-based reasoning can enhance the legal rule-based reasoning module's explanation by identifying and analysing relevant case(s) and argument(s) that support a particular categorisation.

Our work provides a methodology and architecture for constructing such legal knowledge-based reasoning systems. The framework developed provides developers of legal knowledge-based systems with a unified approach to the problem of combining legal rule-base reasoning with case-base reasoning. The IKBALS (Intelligent Knowledge BAsed Legal System) system is an application developed using this framework which aims to provide interactive knowledge-acquisition tools that aid developers of legal knowledge-based systems design and test both their case-bases and their rule-bases. A separate run-time module allows end-users to run consultations with the knowledge-base developed above. The system itself is comprised of intelligent cooperating objects (agents) which combine to solve tasks by reasoning with symbolic representations. The system establishes a common communication protocol that agents use when sending messages across the system.

The development module of the current system consists of a Case Based Reasoning Editor and an Rule Based Reasoning Editor. The Case Based Reasoning Editor uses a customised inductive learning algorithm INDUCE2 to index cases in the case-base. Cases are not discarded by the inductive algorithm but are used to provide an explanation-based analysis. In addition, the algorithm takes advantage of any background information provided by the developer. This background information concerns attributes included in the learning task, and is used by the algorithm to build better decision trees. Background information is stored as symbol hierarchies in the knowledge-base. The decision tree produced by INDUCE2 is converted into quantitative production rules which are then used by IKBALS III to locate relevant similar cases in the case-base. Reasoning with arguments is also supported. These arguments are brought into the legal analysis during Case Based Reasoning explanation. Finally, a query facility is provided that enables either the developer or the end-user of the run-time system to investigate the relationship between cases / arguments in the knowledge-base.

In using a distributed approach of cooperating agents to perform the integration of rule-based and case-based reasoning (rather than the centralised blackboard approaches of CABARET[5] and PROLEXS[6]) we are performing fundamental Artificial Intelligence and Database research on building multi-modal reasoning systems. This work is readily generalisable to other domains.

---

5 Rissland, E.L. and Skalak, D.B., 1991, 'CABARET: Rule Interpretation in a Hybrid Architecture', 34 *Int' J. Man-Machine Studies* p839.

6 Walker,R.F., Oskamp,A., Schrickx,J.A., Opdorp,G.J., Berg,P.H. van den, 1991, 'PROLEXS: Creating Law and Order in a Heterogeneous Domain', 35 *Int'l J.Man-Machine Studies* p35

This project commenced with the construction of a purely production rule-based system IKBALS I[7] which was constructed using the Goldworks environment. The IKBALS II[8] prototype used a blackboard approach to integrating rule-based and case based reasoning in the domain of Workers' Compensation. Our latest prototype IKBALS III[9] also integrates rule-based and case-based reasoning, but this time using a distributed agent approach. Using the domain of the Credit Act (Victoria, Australia), we have constructed and optimised the operation of a rule-based reasoning module (used to model legislation) using class/object structures. The rule-based module of IKBALS III has been used in a commercial system.[10] We show how these class/object structures can be further used by a case-based reasoning module that retrieves and analyses similar relevant cases from a case-library using induction.

The Case Based Reasoning module of IKBALS III comprises a number of separate agents. There are three active agents in our system: the Case Based Reasoning Editor agent , the Rule based deductive agent and the Case based agent. Each of these agents has been designed and implemented separately and they themselves decide whether they want to communicate with any other agent, via a Common Communication Layer. Our system has been designed to adhere to the principles of Distributed Artificial Intelligence by supporting design autonomy and communication autonomy.

2. The Case Based Reasoning Editor

The Case Base Reasoning Editor uses a series of interactive editors which allow developers to define their case-based reasoning modules. The Editor itself is comprised of several distinct utility agents that cooperate to represent and index cases, as well as providing facilities that enhance explanation via access to deep domain models. All sub-editors are comprised of agents that coordinate to solve specific problems. The Index Generator editor, for example, schedules agents to handle the task of induction.

The Case Base Reasoning Editor comprises several modules: the Schema Editor, the Case Editor, Background Information Editor, the Causal Editor and the Index Generator. The Schema and Case Editor allow the user to specify the structure of the case base. The Index Generator uses induction to automatically index cases in the case base. It makes extensive use of any background information provided by the user as entered into the Background Information Editor. The Causal Knowledge Editor allows the developer to represent adversarial reasoning by defining legal arguments.

Our experiments with IKBALS II indicated that dynamic generation of indices suffers from the following problems.

---

7 Vossos, G., Dillon, T., Zeleznikow, J. and Taylor, G., 1991a, 'The Use of Object Oriented Principles to Develop Intelligent Legal Reasoning Systems', 23 *Australian Computer Journal*, pp 2-10

8 Vossos, G., Zeleznikow, J., Dillon, T. and Vossos, V., 1991b, 'An Example of Integrating Legal Case Based Reasoning with Object Oriented Rule-Based Systems - IKBALS II', *Proceedings of the Third International Conference on Artificial Intelligence and Law*, ACM Press, pp 91-101.and Vossos, G., Zeleznikow, J. and Hunter, D., 1993, 'Designing Intelligent Litigation Support Tools - the IKBALS Perspective', Law, Computers and Artificial Intelligence 2(1) pp. 77-93, 1993.

9 Zeleznikow, J., Vossos, G. and Hunter, D., *'The IKBALS Project: Multi-modal reasoning in Legal Knowledge Based Systems'* to appear in AI and Law: An International Journal

10 Vossos, G., Zeleznikow, J., Moore, A. and Hunter, D., 1993b, 'The Credit Act Advisory System (CAAS): Conversion From an Expert System Prototype to a C++ Commercial System', *Proceedings of the Fourth International Conference on Artificial Intelligence and Law*, ACM Press, pp 180-183.

- Such a system displaces control from the knowledge engineer / expert and places it in the hands of end-users. In a sense, the end-user is required to perform the knowledge-modelling. This transfer of responsibility is clearly undesirable;

- Lack of validation. If the system's knowledge-base is left open to modification by the end-user, then it cannot be validated. This causes concerns over the long-term correctness of the system, and, within commercial environments, a lack of acceptance since no one is willing to accept liability for the system behaviour.

- It diminishes performance. Having to process cases sequentially at run-time may have diabolical consequences for the performance of a system comparing cases in very large databases.

Allowing the developer to pre-index cases in the case-base using the method of induction was the preferred approach taken in the IKBALS III project. Unlike other induction algorithms however, IKBALS III uses supplementary background information to guide the generation of the index. Further, the object knowledge-base is used to represent both cases and background information. In order to provide a facility for reasoning with arguments, IKBALS III provides editors to represent and reason with legal argument. Hence, the contents of a legal case in IKBALS III are jointly represented by a factual case description as entered in the Case Editor as well as the arguments for and against the credit provider. These arguments, are indexed separately by IKBALS III and are used by the memorandum generator in producing its explanation.

3. Machine Learning in IKBALS III

There were three main practical reasons why we did not adopt the ID3 algorithm.

- The agents in our system do not themselves contain data-structures. Instead, all data structures used to generate the decision tree are represented in the external knowledge-base. Conventional inductive algorithms execute recursive calls that generate structures internally. Hence such algorithms would have to be re-written to reflect this fact;

- Standard inductive algorithms provide no facility to incorporate background information into the learning process. The ability to access this type of information in a uniform way meant that such algorithms were unsuitable for use in our system;

- ID3 code is not object-oriented and thus would have to be re-written for our purposes.

For these reasons, we designed our customised inductive learning algorithm, INDUCE2.

The INDUCE2 algorithm uses a similar selection criteria to ID3 in subdividing the example set at each step. It uses a top-down, divide and conquer strategy that partitions the given set of examples into smaller and smaller subsets in step with the growth of the decision tree. The order of presentation of the examples is not important. INDUCE2 uses the object knowledge-base to store the structures it processes when constructing the decision tree. The decision tree itself is also represented as an object-hierarchy in the IKBALS III knowledge-base. The nodes of the decision tree correspond to questions, answers and category nodes. The INDUCE2 algorithm represents its acquired knowledge initially as a decision tree in the form described, and then converts the decision tree into rules.

Given a number of facts, generalisation can be performed in many different directions.[11] In order to constrain a generalisation process and extract interesting rules from a case-base, learning should be directed by background knowledge, such as knowledge contained in concept hierarchies. Concept hierarchies can provide valuable information for inductive learning. By organising different levels of concepts into a taxonomy, rules can be restricted to a form which comply to certain syntaxes. This is often referred to as conceptual bias.[12] The net result of rules described by higher level concepts is that the domain is then represented in a simple and explicit form.

Knowledge about concept hierarchies in IKBALS III is directly provided by domain experts. Each concept hierarchy relates to a specific attribute in the domain and is organised from the most general to the specific. The most general point in the concept hierarchy is the name of the attribute itself (the root) with the leaf nodes corresponding to the actual values of the attribute as found in the case-base.

The *UseBackInfo* agent supplements the normal operation of the INDUCE2 learning algorithm by providing additional information concerning the attributes used in the learning task. This handler attempts to prune the branching factor occurring out of any node in the decision tree. It does this by trying to generalise the values of an attribute if the total number of values for an attribute exceeds the threshold. The algorithm traverses the attribute's symbol hierarchy one level at a time, searching for a concept that generalises the values of the attribute to less than or equal to the threshold. The algorithm performs this function by referring to previously stored background information. This background information is defined by the domain expert and encoded by the developer using the Background Editor. The background information is stored in a separate class hierarchy in the knowledge-base.

The strength of the overall approach taken lies in its ability to convert the decision tree into rules and incorporate background information. This approach differ from other inductive algorithms in that it accesses a domain model during the building of the decision tree in an attempt to reduce noise and compact the decision tree. Weaknesses of this approach include the possibilities of generating large decision trees and univariate splits.

4. The Use of Causal Information in IKBALS III

The IKBALS III program assumes that open-textured rule predicates do not have one right answer. Instead, the program assumes that there will be competing reasonable arguments. Its analysis of these arguments involves distinguishing which side particular arguments favour, as well as establishing a mechanism for being able to compare these arguments symbolically.

Legal arguments are represented in IKBALS III by justification heuristics which are in turn represented in the knowledge-base by a causal network. Agents in the Causal Editor allow developers to define justification heuristics along with explanatory and analytical information which is then used by the Report Generator to compare the relative strengths and weaknesses of similar cases. Like all of our reasoning structures, justification heuristics are stored and accessed by agents using the Application Program Interface.

---

11 Dietterich, T. G. & Michalski, R.S., "A comparative review of selected methods for learning from examples", in Michalski, R.S. et al., Eds., *Machine Learning: An Artificial Intelligence Approach*, Vol. 1, Los Altos, CA: Morgan Kaufmann, 1983, pp. 41-82.

12 Genesereth, M. and Nilsson, N., "Logical Foundation of Artificial Intelligence", Los Altos, CA, Morgan Kaufmann, 1987

Justification heuristics serve to represent the various legal arguments used by lawyers to justify a particular outcome. Justification heuristics represent a useful index into cases for the purpose of retrieval. Instead of associating justification heuristics directly to cases at development time, a mechanism was required that would identify potentially relevant arguments given a description of the case which is being examined at run time. Such an approach cross indexes the relevant case instance(s) which meet the requirements of the justification heuristics. These cases are then indexed by the justification heuristics(s) that applied to it. Relationships between cases and arguments are then be investigated by agents using the Application Program Interface.

In order to facilitate for case comparison and analysis, one slot from the list of conditions is selected as the critical slot. This slot is used by the report agent as the basis of comparing the relative strengths and weaknesses of particular arguments. This approach to legal analysis corresponds to Ashley's 'dimensional' approach.[13] If the conditions of a justification heuristic are met, the justification heuristic rule cross indexes the case instance(s) as an instance of that justification heuristic. By late binding justification heuristics at run-time the developer can change the composition of the cases in the case-base without affecting the applicability of justification heuristics.

5. The IKBALS III Run-time System

The IKBALS III run-time system consists of several modules. Most of these modules are concerned with the control of the rule-based reasoning aspect of the system. Two of the modules are dedicated to providing the case-based reasoning support.

The issue of integration is somewhat simplified in IKBALS III , due mainly to the Application Program Interface bridge between the application and knowledge-base. Since the knowledge structures used by both the case-based reasoning module and the rule-based reasoning module coexist in an external object knowledge-base, the agents of both modules gain access to this rich pool of knowledge by communicating via the Application Program Interface.

Agents in the IKBALS III system are autonomous; once activated by a user event, agents execute their specific task without requiring intervention. Control is hard wired directly into those agents which are then responsible for planning, scheduling and executing their tasks. Agents are able to create, modify and delete class, object and rule definitions as well as to control the reasoning process via access to the inference engine.

6. Conclusion

We have described an object-oriented integrated rule-based/case-based reasoning system, which provides for intelligent case indexing, retrieval and reasoning with an induction algorithm. It provides for cooperating agents, making for flexible and simple modification and amendment. Its case-based reasoning agents allow for induction, and indexing on heuristic matching strategies. Though our system operates within a legal domain, we believe that the methodology and architecture is of general application to domains other than law.

---

13 Ashley, K. D., 1991,*Modelling Legal Argument: Reasoning with Cases and Hypotheticals,* MIT Press.

Chapter 7

Knowledge/Software Engineering and Case-Based Reasoning

Model of Problem Solving for the Case-Based Reasoning

Ikram CHEIKHROUHOU

LAFORIA-IBP, Université Paris VI
Tour 46/00, Boîte 169
4, place Jussieu
F- 75252 Paris Cedex 05

e-mail : ikram@laforia.ibp.fr

Abstract This paper proposes a model of problem solving for the case-based reasoning. This modelisation uses the KADS formalism. As other knowledge acquisition methods, KADS offers a representation and structuration framework which try to respect the semantic and the role of the knowledge. The possession of models of problem solving method can facilatate the knowledge acquisition process by specifying the role that the knowledge plays during the problem solving process.

1 Introduction

Founded whether on rules or cases, a knowledge based system uses the domain knowledge to solve a problem. Whatever the choice, we are led to analyse this knowledge and to select the most relevant one for the problem solving. This selection assumes that we know how to organize and to exploit this knowledge. So, a close link emerges between the knowledge acquisition and the problem solving. Indeed, it isn't easy to assess what a knowledge can bring and how to structure it if we don't know what it will be used for.

Often considered as a constructive modelling process, the knowledge acquisition aims to construct conceptual models. These models are important as much as they can point out the problem progress term to its solution by showing, in every stage, the necessary knowledge to elicit. Considered as problem solving methods, these models are generic and so independent of domain. Case-based reasoning can be one of these methods. It may be used, *a priori,* for different kinds of tasks.

The purpose of this paper is to propose an interpretation model based on KADS[1] for this case-based reasoning. This interpretation model is susceptible to help knowledge engineer. It indicates him, notably, what knowledge types ought to be acquired and what roles to attribute to problem solving process.

A short preview of the modelling in knowledge acquisition is presented in Section 2. The Section 3 describes the KADS method. We put stress on the interpretation models assimilated to models of problem solving methods. Section 4 contains our modelisation of the case-based reasoning using the KADS formalism. The last section concludes about the use of the case-based reasoning in the Knowledge-Based system (cooperation & complementarity).

2 Knowledge Acquisition and Modelling

A wide consensus takes shape to the research workers in the knowledge acquisition. It consists in recognizing that the knowledge acquisition passes beyond the frame of expertise transcription to become a modelling process[2]. Indeed, after the introduction of the Newell's knowledge level notion [3], some works have shown the interest of describing the domain knowledge in a conceptual level, which is a level of knowledge interpretation. Many approaches have been proposed. We are interested more particularly in interpretation models of KADS method.

1- KADS, Knowledge Acquisition and Design Structuring, is a methodology for the development of knowledge based systems (ESPRIT I, II project).

The notion of interpretation models defined by the KADS concepters allows to guide the knowledge engineer at the time of the data analysis. The interpretation models contain an abstract description of the characteristics of the different generic tasks. By describing the problem solving process independently of the domain, these models want to be reusable and then quite generic. They produce conceptual models of the futur system when instantiated with domain knowledge.

3 KADS

KADS is a model-driven methodology. It is a "language" of european researchers community in knowledge acquisition. It's based on construction and transformation of models. Each model emphasises some aspects of the system to be built and abstracts from others [6]. KADS separates the knowledge analysis phases from their representation.

Issued from the analysis phase, the construction of the conceptual model is a central activity in the process of knowledge-based-system construction. To construct this model, KADS offers a four layers structure : from the less abstract level describing the domain concepts to the most abstract level representing the system behaviour [1]

The domain layer : this level describes the domain concepts and the relation between them.

The inference layer : this level describes the inference structure, which is a description of inferencing capability based upon knowledge at the domain layer. The inference structure specifies the problem solving competence of the target system. It comprises a diagram consisting of knowledge sources and metaclasses :

- The knowledge source is a representation of a single inference at the inference layer. It consumes one or more metaclasses as input, and produces one metaclass as output. The process by which this transformation is achieved is based on the methods which constitute the internal contents of the knowledge source.

- The metaclass is a representation of the role that may be played by a domain concept during the problem solving.

The task layer : this level describes the decomposition on subtasks whose organize and control these primitive inferences. It represents the reasoning stages (goals) and how to reach them. A task indicates how to follow an inference structure.

The strategy layer : this level allows to specify the choice of several tasks if these coexist.

To facilitate the interpretation of verbal data obtained from the expert, KADS provides a library of interpretation models that consist of a task and an inference layer. An interpretation model is an abstract description of a problem solving method. It gives the knowledge engineer strong indications of what domain knowledge is needed because the metaclasses and knowledge source indicated in this model must have domain specific equivalents.

KADS proposes a classification of generic tasks. For each task, one or more problem solving method are proposed. "Integrate" case-based reasoning model is important as much as this method is often used by the expert to solve a problem. Then, it is more easy to "collect" expert experience in cases form. In addition, cases provide some aspects of the context, which can facilitate the problem solving. Moreover, with several problem methods in the library, the knowledge engineer can select one or several problem methods according to the kind of the problem. These methods can complement each other and/or cooperate.

4 An Interpretation Model for the Case-Based Reasoning

In this section, we present our modelisation of a case-based reasoning, according to the KADS formalism. We give the inference and the task layers, which represent the two essential constituents of the KADS interpretation models.

4.1 The Inference Layer

The Inference Structure.

The following diagram (fig.1.) shows the set of inference steps.

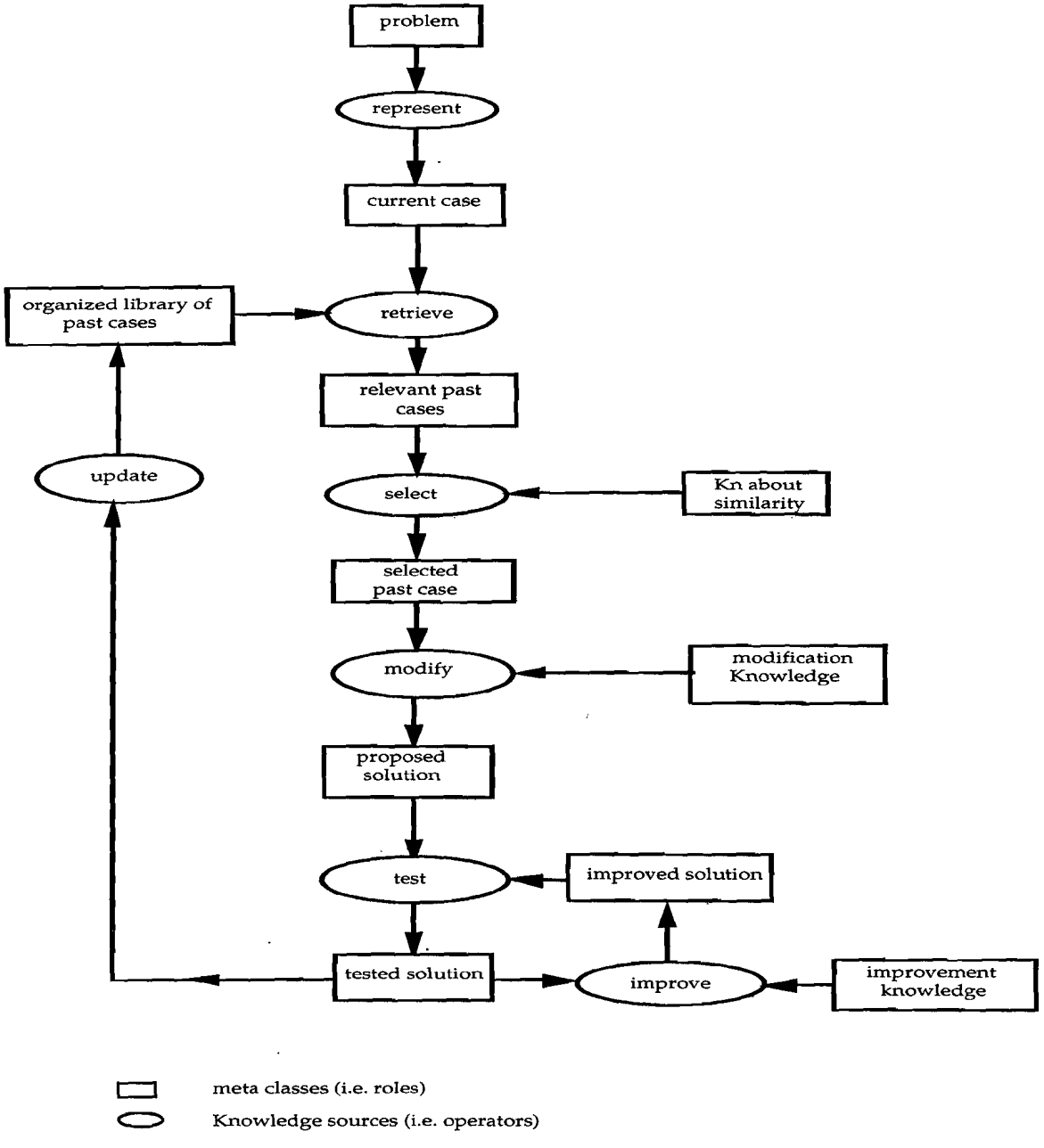


Fig.1. An inference structure for the case-based reasoning

The Knowledge Sources Description.

The knowledge source "*represent*" translates the new problem (i.e. new event) on a case ; so it will be comparable with previously stored cases. This process will be facilitated by the feature indexation of the new event.

The Inputs of the knowledge source "*retrieve*" are the current case and the organised library of past cases. The function of this knowledge source is to research the most relevant cases by using the library of cases.

"*Select*" allows to decide which case is more like the current situation. This process is possible with the help of the knowledge about similarity.

The next step consists in the adaptation (i.e. modification) of the old solution to satisfy the requirements of the new problem. This process is facilatated by using the various features of the new context.

The knowledge source "*test*" evaluates the proposed solution. If this solution complies the new event constraints then it is proposed for the user and put away the library of cases. "*Improve*" allows to explain and to correct the failure of test.

"*Update*" updates the cases library, in consequence of new cases informations.

4.2 The Task Layer

This layer represents an other constituent of the interpretation model. At this layer, the task structure is described. We remind that the task structure is the procedural overlay on the inference structure that shows the sequence of knowledge source invocation [1]. In the following figure, we present the task structure for the inference structure showed in Figure 1.

find (problem -> solution)
represent (problem -> current case)
retrieve (organized library of past cases, current case -> relevant past cases)
select (Knowledge about similarity, relevant past cases -> selected past case)
modify (selected past case, modification Knowledge -> proposed solution)
While (test (proposed solution, improved solution -> tested solution) = bad solution)
(improve (improvement knowledge, tested solution -> improved solution))
EndWhile
update (tested solution -> organized library of past cases)

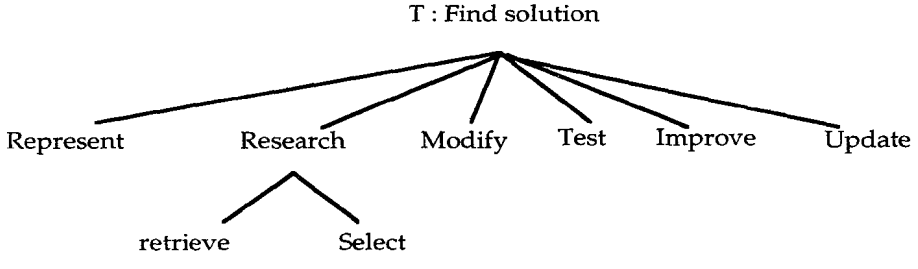


Fig.2. A task strucure for the Case -Based Reasoning

5 Conclusion

Leaning on the case-based reasoning modelisation according to KADS, we have tried to study the different necessary types of knowledge and their interactions.

It is interesting to insert this solving problem method in the library of KADS interpretation models. We stress the importance of using several methods to benefit from the mutual contribution approaches. For example, on the case-based reasoning research phase, an acceptable solution for the current case may not be obtained. In this case, the system can use an other method to find an available solution instead of returning an

echec message.

The model, presented in this paper, shows the different phases that a case-based reasoning system can satisfy. For each phase, the necessary knowledge has been indicated. Moreover the case-based reasoning is used for different kinds of tasks [4] (e.g., diagnosis, conception and planning). A preliminary study of the different techniques used for each case-based reasoning phase showed that an appropriate technique can be found for each specific task. It is perhaps early to give a such conclusion, but a further studies should confirm it. In this case, the appropriate techniques should allow to understand better what kinds of knowledge are really used and hence to be acquired. Moreover, a deep analysis of knowledge allows a more efficient cases representation, which could improve the selection of the most relevant cases and adaptation processes.

References

[1] F.R. HICKMAN, J.L. KILLIN, L. LAND, T. MULHALL, D. PORTER and R.M. TAYLOR: *Analysis for knowledge-Based systems, a practical guide to the Kads methodology.* Ellis Horwood, (1989).

[2] J.P. KRIVINE and J.M. DAVID: *L'acquisition des connaissances vue comme un processus de modélisation : méthodes et outils.* In: Intellectica, numéro spécial : *expertise et sciences cognitives*, n° 12, pp. 101-138, (1992).

[3] A. NEWELL: *The Knowledge Level.* In: Artificial Intelligence 18, pp. 87-127, (1982).

[4] Proceedings of Case-Based Reasoning Workshop. Sposered by DARPA, (1989)

[5] S. SLADE: *Case-Based Reasoning : a research paradigm.* In : AI Magazine 12, n° 1, pp. 42-55, (1991).

[6] B. J. WIELINGA, A. Th. SHREIBER and J.A. BREUKER: *KADS : a modelling Approach to Knowledge Ingineering.* In : Knowledge Acquisition 4, n° 1, pp. 5-53, (1992).

A Software Engineering Model for Co-operative Case Memory Systems

Andrew M. Dearden* and Michael D. Harrison
Department of Computer Science
University of York
Heslington
YO1 5DD
UK
email: andyd@minster.york.ac.uk
mdh@minster.york.ac.uk
tel: (44)-904-432765

August 26, 1993

1 CBR and Case Memory Systems

The process of solving a problem using Case-based Reasoning (CBR) has been described as a sequence of steps as follows:

1. analyse the current situation;
2. generate a set of indices;
3. retrieve and order a set of stored cases;
4. select one or more cases to work with;
5. adapt the chosen case to fit the new situation;
6. test the adapted case and repair any failures.

A number of CBR systems have been constructed in which the first three steps are handled by separable software modules. Commercial products which support only these three steps have achieved notable success (for example [5]). Software systems that capture these first three steps may be described as Case Memory Systems (CMS). The concern of this paper is to develop a model for Case Memory Systems that may be used in the analysis of existing systems and the design of new ones.

*Supported by a CASE studentship from SERC and BTplc

1.1 Why a Model is Needed

For a software engineer attempting to apply a CMS to a new domain the primary sources of information are previous case studies. The engineer is faced with the problem of finding the case study which most closely reflects the important features of the new domain and applying it appropriately.

In order to select between different CMS designs software engineers require a model that will enable them to make an objective assessment of the alternative design possibilities. The model must be capable of expressing important similarities and differences between CMS, and should allow the software engineer to reason about properties of proposed designs as well as completed systems.

1.2 The Advantages of a Formal Model

For example, previous discussions of properties of CMS, e.g. [2], have identified a number of potential indexing terms. However the terms proposed are unreliable because:

- properties are described by reference to its application to a particular domain of knowledge, and *via* a particular knowledge representation, rather than being abstracted to a domain independent level;
- it is not possible from this information alone to examine whether a particular algorithm does or does not satisfy selected properties;
- it is not possible to discover whether any of the properties described are interrelated.

By using a domain independent notation, the software engineering notation Z [7], we have produced a general model of CMS which can be used to explicate important distinctions between CMS reported in the literature. This model:

- is abstracted away from particular domains and tasks;
- can be used to relate different properties;
- can be used to verify or refute claims about the capabilities of different algorithms

We are currently working on an extension to the model to explore distinctions that may arise when considering how a human user might interact with a *Co-operative* CMS (CCMS).

2 The Basic Model

A CMS is modelled using sets in three domains.

A set of *Problem Statements* (PS), representing queries input to the system.

A set of *Descriptions* (D), each description being a set of attribute-value pairs. Descriptions are used to index the case base.

A set of *Reports* (R), which represent the information content of a case.

A Case Base (CB) is characterised as a *function* between Descriptions and Reports. Hence an individual case is a unique Description-Report pair. Querying a CB involves inputing a Problem Statement, translating the Problem Statement to generate a Description, then using the Description, and perhaps other elements of the Problem Statement, to produce an ordering over the set of cases. A CMS is modelled by a translation function T (from Problem Statements to Descriptions) and a higher-order function from CBs to Problem Statements to orderings of cases.

$$
\begin{array}{|l}
\hline
CMS \\
T : P \rightarrow D \\
CMS : (D \rightarrow R) \rightarrow PS \rightarrow (Cases \leftrightarrow Cases) \\
\hline
\forall C : D \rightarrow R; \forall p : PS \bullet \\
CMS(C, p) \in Orderings \\
\hline
\end{array}
$$

The orderings permitted are *lattices of equivalence classes* of cases.

2.1 Comments on the Basic Model

The model allows for many different representations of the case content or for the input to a CMS. The sets of problem statements and reports could be anything at all. Modelling the indexing terms as attribute-value pairs also allows for systems where indexing is based on lists of atomic features. The definition of Orderings as lattices of equivalence classes allows for systems which select one case, systems that use numerical scores, as well as systems such as HYPO [1] which generate complex partial orderings.

To date we have not found any CMS which cannot be related to the basic model. However, the model does assume that the problem statement is static, and does not change, thus it cannot distinguish properties that are important to CCMS. The modelling of such systems is discussed in section 4 below.

3 Distinguishable Properties in the Model

Using the model a number of properties that have been discussed in the literature can be made explicit.

3.1 Directable CMS

Kolodner in [2][p77] argues that if a reasoner is pursuing a particular goal, this goal will be important in similarity assessment. Goals may be used as indexing terms, but in general it is not possible to predict the full set of goals for which a case might be relevant. Instead the reasoner's goal is used to indicate other features that are important to match. We can distinguish this property by considering the role of the translation function T. If a system is able to use part of the input Problem_Statement, which is not used for indexing cases then it must be possible to find two problem statements, which translate to the same

description, but which (for some case bases) result in different case orderings. Formally:

Directable_CMS

$$CMS$$

$$\exists p_1, p_2 : P;\ \exists C : D \rightarrow R \mid T(p_1) = T(p_2) \bullet$$
$$CMS(C, p_1) \neq CMS(C, p_2)$$

As we are interested in a general model we prefer to avoid terms such as Goal-directed which may be specific to planning domains, and prefer the term *directable* to describe systems which might have this property.

3.2 Sensitivity to Context

Ashley [2][p74] argues that similarity assessment should recognise the significance of particular *combinations* of factors, rather than considering each attribute independently.

If we imagine an input Problem Statement, p_1, to a CMS which results in an ordering which prefers a case (d_1, r_1) over (d_2, r_2). If the attributes are considered independently, then changing the problem statement to p_2, where p_2 translates to a superset of p_1, should only change to ordering of (d_1, r_1) and (d_2, r_2) if these cases differ with respect to the newly acquired attributes in $T(p_2) \backslash T(p_1)$. Conversely if we can find a CMS, and cases where this does not hold then the CMS must be modelling some interdependence of attributes, i.e. the significance of individual factors depends on their context.

Formally attribute interdependent CMS satisfy the schema:

Attribute_Interdependent_CMS

$$CMS$$

$$\exists C : D \rightarrow R;\ \exists p_1, p_2 : P;\ \exists As : \mathbb{P}\, Attributes \mid$$
$$\text{dom}\, T(p_1) \subseteq T(p_2) \wedge As = \text{dom}\, T(p_1) \backslash \text{dom}\, T(p_2) \bullet$$
$$(\exists (d_1, r_1), (d_2, r_2) : C \mid As \lhd d_1 = As \lhd d_2 \bullet$$
$$((d_1, r_1), (d_2, r_2)) \in CMS(C, p_1) \wedge \neg((d_1, r_1), (d_2, r_2)) \in CMS(C, p_2))$$

Here the symbol $\lhd$ indicates domain restriction:

$$B \lhd A = \{(x, y) : A \mid x \in B\}$$

3.3 Other Properties

We have also used the model to analyse different mechanisms for taking past experience into account in determining similarity. In particular we have identified three ways in which this might occur:

> **Report Examination** - the use of the content of the individual cases to affect similarity assessment, e.g. by marking particular features of a case as salient.

Domain Content Sensitivity - the use of the distribution of features amongst the indexing Descriptions to indicate the relative significance of attributes, e.g. by weighting in favour of evenly distributed features.

Report History Sensitivity - the use of the content of the set of stored reports to affect similarity assessment, e.g. by identifying features which are highly discriminating amongst the set of reports.

3.4 Relating Properties

Because the Z notation includes a formally defined semantics, the model can support reasoning about the relationships between different properties. For example, it is possible to prove that for a system to be directable it must be attribute interdependent. The proof of this result follows by considering the case of attribute interdependence when the the set As is empty.

3.5 Analysing Algorithms and Systems

A detailed analysis of a single system is beyond the scope of this short paper. However, the Z notation is designed to support such analysis and we have applied the model in analysing properties of a number of systems. The interested reader is referred to [4].

4 Modelling Co-operative CMS

The basic model of CMS assumes that the Problem Statement is static and is completely known in advance. In many domains, such as diagnostic help desks [3] this is not a valid assumption. A Co-operative CMS (CCMS) is a CMS which supports a human user in gradually refining their understanding of the current problem. Examples of CCMS are CBR Express [5] and the case delivery component of KID [6].

Many knowledge based systems (KBS) projects have failed because inadequate attention was paid to human-computer interaction. By constructing a formal model of the option space for CCMS we hope to support software engineers in generating interface designs which are appropriate for the particular task or environment in which the CCMS it to be used.

Below we list and describe (informally) some of the properties we are currently investigating.

Flexibility : A CCMS must provide operations which allow a user to change the value of the current PS. In any given state there will be a set of problem statements to which the system is prepared to move in a single operation. We name this set the *ready* set for the CCMS. The larger the ready set the more flexible the interaction with the CCMS may be.

Advice and Initiative : A CCMS should provide advice to the user on ways in which the problem statement might be refined. The relationship between the advice and the ready set will be important in characterising initiative in the dialogue. If a CCMS is to provide a mixed-initiative dialogue, the ready set must be larger than the advice set.

Presentation and Initiative: The way in which the system presents cases to the user also involves choices about system-, user- and mixed-initiative presentation. Since only a limited number of cases can be presented at any one time, the choice of how much detial about cases is given may lie with the system or the user at various times during a consulation.

Advisory Strategies: Sequences of refinement advice given by a CCMS may be designed to lead the user to *converge* on some case or set of cases which are in some sense the 'most useful' for the user's current problem. Alternatively the advice given may be designed to partition the currently 'most useful' set.

5 Conclusion

To be useful to software engineers a model of CMS or CCMS should be independent of particular knowledge domains or tasks; should support precise reasoning about the properties of individual designs or systems; and should allow investigation of the relationships between different properties. Such a model could be used by sofware engineers in analysing the requirements of a particular application, in locating similar previous designs, and in verifying the properties of completed systems.

We have presented a formal model for the option space of CMS which satisfies these criteria We aim to generate a similar model for the option space of CCMS.

References

[1] Ashley KD, Rissland EL. A case-based approach to modeling legal expertise. IEEE Expert, 3(3):70 - 77, 1988.

[2] R. Bareiss *et al.* . Panel discussion on indexing vocabulary. In DARPA Case based reasoning workshop 1989, pp 66 - 84. Morgan Kaufmann, 1989.

[3] Bridge DG, Dearden AM. Knowledge based systems support for help desk operations: A reference model. Int. J. Systems Research and Information Science, 5:217 - 234, 1992.

[4] Dearden AM. The engineering of co-operative case memory systems. 2nd year DPhil report, Dept. of Computer Science, University of York, 1993.

[5] Klahr P, Vrooman G. Commercialising case based reasoning technology. In Graham IM, Milne RW (eds), Research and Development in Expert Systems VIII, pp 18 - 24. Cambridge University Press, 1991.

[6] Nakakoji K. Increasing shared understanding of a design task between designers and design environments: The role of a specification component. PhD thesis, Dept. of Computer Science, University of Colorado, Boulder, 1993.

[7] Spivey JM. The Z notation: A reference manual. Prentice-Hall International, 1988.

Toward a Task-oriented Methodology in Knowledge Acquisition and System Design in CBR *

Dietmar Janetzko
University of Freiburg
Institute of Computer Science
and Social Research
D–79098 Freiburg, FRG
dietmar@cognition.iig.uni-freiburg.de

Katy Börner
HTWK Leipzig
Department of Informatics
P. O. Box 66
D–04251 Leipzig, FRG
katy@informatik.th-leipzig.de

Carl-Helmut Coulon
Gesellschaft für Mathematik
und Datenverarbeitung
D–53757 Sankt Augustin 1
P.O. Box 1316
coulon@gmd.de

Ludger Hovestadt
University of Karlsruhe
Department of Architecture
Englerstr. 7
D– 76128 Karlsruhe
ludger@ifib.uni-karlsruhe.de

Abstract

In knowledge acquisition and system design there are strong interactions between the tasks the system has to fulfill, the methods chosen, and the knowledge needed. In this paper, we will present an introductory analysis of these interactions. An example is taken from the domain of building design to elucidate the problem of designing supply nets. Additionally, we will present an inference structure that serves as a basis for system design. Both the example and the corresponding inference structure will be used to spell out and exploit the interactions between tasks, methods, and knowledge in knowledge acquisition and system design.

1 Introduction

A major problem in building knowledge-based systems arises if the interaction between knowledge acquisition and system design is neglected. There are a number of pitfalls resulting from that interaction. One of them is knowledge acquisition that does not feed into system design. Another negative effect of that interaction is system design that enforces using a particular kind of knowledge that may or may not be relevant for modeling a task. Although these problems are widely known, there is still a need for a methodological framework that provides guidelines for the application of methods and knowledge once a particular main task ought to be modeled and decomposed. The goal of this paper is to specify the interactions between tasks, methods, and knowledge.

The sections of this abstract may be summarized as follows. First, we present a view on knowledge acquisition and system design that rests upon the notions of tasks, methods, and knowledge. Second, an example is used that permits to analyze the interactions introduced. The example

*This research was supported by the German Ministry for Research and Technology (BMFT) within the joint project FABEL under contract no. 413-4001-01IW104. Project partners in FABEL are German National Research Center of Computer Science (GMD), Sankt Augustin, BSR Consulting GmbH, München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

addresses the design of supply nets, which is a task in building design. Third, we present a computational framework by using MoMo [5] to build an inference structure. This structure allows for a specification of the interactions for conceptual modeling of design and planning tasks taken from the domain of supply nets. Finally, we present a short outlook on the analysis of interactions to be presented in the extended version of this paper.

2 Tasks, Methods, and Knowledge

Tasks, methods, and knowledge form the three basic buildings-blocks involved in knowledge acquisition and system design [4], [3]. In this section, we first introduce each of these three notions and give an outline of interactions between them. In addition, we will point out the relationships between the notions task, methods, knowledge, and the concepts in MoMo, which is the conceptual modeling language we use.

Tasks play an important role in modeling problem solving [6], [1]. Examples of tasks are word processing, job scheduling, and fault diagnosis. Tasks refer to the things to be achieved. To make tasks come true methods and knowledge are needed. If a task cannot be achieved by a single method it has to be decomposed. In MoMo [5], the equivalence of task is called *action* or (MoMo-) task.

Methods are procedures that implement abstract problem solving models. Methods provide the active part in reasoning that carries out problem solving. Methods mediate between tasks and knowledge. Which method is chosen and applied to the task hinges on criteria like availability of knowledge, computational costs, and reliability of the solution. A method which is applied at one level of a task structure may be applied at another level of the same task structure, too. In MoMo, the equivalence of method is referred to as a *generic function.*

Knowledge is used here in a technical sense and refers to the input and output of tasks. In this way, knowledge provides the material methods employ for their operations. In MoMo, knowledge of a defined *type* is represented in *places* that hold the input and output of actions. Whenever two concepts among the building blocks tasks, methods, and knowledge are fixed, there are only a few possibilities left to fix the third one. Thus, if the task to be modeled and the methods that are applied to the task are selected the appropriate type of knowledge can hardly be freely chosen. This focusing effect may be exploited in knowledge acquisition. Having selected design as the top-level task and case-based reasoning as the general problem solving method exercises a pressure to use cases, concept hierarchies to support indexing, and similarity assessment and rules to support modification. Vice versa, it is very well possible to start with selecting the task and the knowledge to be used, thereby constraining the selection of a suitable method.

3 An Example

Our example shows how a task in the design of a supply air network of an office building is modeled by using case-based reasoning as the general problem solving method.The next section is concerned with providing an inference structure that serves as a basis for a computational modeling of problems of that kind.

In Fig. 1 there is a detailed survey of the steps required to plan and design supply nets in buildings. Note, that each step mentioned in Fig. 1 presents an intermediate result which can be depicted visually and may be captured by an inference structure. Both are presented in Fig. 2. The ellipses shown in the left part of Fig. 2 provide information on a sketch level of design. To use ellipses instead of rectangles is a very useful graphical trick: Ellipses overlap only in a few points. Thus, information on different levels of abstraction can be displayed simultaneously[1]. In the left part of Fig. 2 each intermediate result is shown visually. In the right part of Fig. 2

[1] For a detailed introduction into this representation scheme, the reader is referred to [2]

each intermediate result is depicted as a MoMo place. In addition, the actions which are linked to these places are shown. Each figure in the left part of Fig. 2 has an equivalent part in the MoMo inference structure on the right.

4 A Computational Framework

Case-based reasoning has been chosen for the general problem solving method since in this domain the reuse of previous solutions is a common practice. As a consequence, problem solving in this domain is made up of a number of tasks specific to CBR. These tasks, in turn, have to be tackled by specific methods which need a particular kind of knowledge [3]. These tasks would not have been used in this way if a different general problem solving method (like rule-based reasoning) had been employed.

MoMo is used as a language to represent and operationalize a conceptual model of systems. We use it for a model of CBR problem-solving in constructing supply nets. In MoMo, the control flow is defined on the task layer by using tasks. The data flow is specified on the inference layer in terms of *places*, *actions*, and *types*. Actions refer to inferences directed toward a particular task. Places are "containers" holding the input and output of an action. This kind of knowledge has to be specified by assigning it to a particular type. To represent MoMo models graphically, e.g. an inference structure, boxes are used to represent actions and ovals are used to represent places.

The overall problem in a domain like constructing supply nets in building design is divided into a planning and a design phase. Planning refers to the selection and sequential ordering of subgoals that have to be achieved in order to achieve a main goal. That means, planning provides an answer to the question *when* and *which* problem ought to be solved. Design, on the other side, gives information *how* this problem should be solved. Again, decomposing the overall building design task into planning and design tasks calls for specific methods that further confine the kind of appropriate knowledge.

The *planning phase* takes as input a complex `situation-description`, `known subgoals`, and `state-subgoal-pairs`. The output of the planning phase is a distinct problem identification which is worked out in the design phase. In our example, the complex situation may be compared to a snapshot of problem solving at a certain point in time showing that parts of a supply net are already fixed while other parts still have to be planned and designed. The places `known-subgoals` and `state-subgoal-pairs` are occupied by knowledge that is used for goal-setting. The place `state-subgoal-pairs` contains rules that fulfill two purposes. Firstly, they help discerning a particular state among a complex situation description. Secondly, they help pointing out which subgoal ought to be pursued given a particular state. Both the isolated state and the identified subgoal are then combined. In this way part of the situation, i.e. a state, is turned into a problem which can be tackled. The place `known-subgoals`, on the other side, holds those goals that are already achieved. Thus, `known-subgoals` is a means to control which subgoals are already achieved.

The *design phase* uses `pre-select` to retrieve an appropriate case of the `case-base`. This results in a set of candidate cases called `case-set` which seem to be useful to solve the actual problem. `Select`, i.e. a second selection, is required for further confining this subset. This action produces a reduced set of candidate cases referred to as `best-case-subset`. Having completed the selection of appropriate cases the solution found is transferred to the problem-description. The result is an `expanded state`, i.e. the `problem-description` is enlarged by elements found in previous cases. If the solution transferred from another case does not fulfill the requirements of the problem description an adaptation of the transferred solution is necessary. To this purpose, the `expanded state` together with `adaptation-knowledge` enter the action `adapt` which produces an `adapted state`. Subsequently, two actions are performed in order to test the result. `Justify`, the first of the testing actions, is used to find out if the

`situation description:` This place refers to an early stage in design, when the principles of a supply air system are designed.

`known subgoals:` Each object of the situation is connected with a goal to produce and control more detailed informations. The objects with unsatisfied goals are displayed boldly. These are mainly the objects with the most detailed informations.

`problem description:` Given a complex situation, there may be many problems to be solved. This figure shows the problem with the first priority. In this example it is the object for one supply air network. The selected problem can be isolated from the situation and needs only a few other objects (state): the six mid-sized circles represent the rough location of the supply air outlet in the ceiling of one floor in the office building. The small circle represents the vertical duct. The selected problem is to connect the outlets with the vertical duct.

`case base:` The case base shows several prior solutions to the same problem. The ducts we are looking for and which connect the outlets with the vertical duct are represented by the ellipses.

`case set:` The case set consists of a few cases which are most similar to the selected problem.

`best case subset:` This is the case the solution of which needs minimal adaptation to fit the demands of the selected problem.

`expanded state:` This figure shows the best case subset in the background and the state in the foreground, in order to prepare the adaptation.

`adapted state:` In this example the state needs no change, but the horizontal ducts of the best case subset were added and one duct needs shortening.

`justified state:` Some simple checks can be made, e.g. the overall length of the ducts or the number of direction changes are restricted.

`evaluated situation:` Some more checks might be necessary before the selected problem can be merged to the situation, e.g. the spatial coordination of the supply air ducts and the return air ducts.

`situation*:` If no test fails, the state can be merged to the situation, which now contains more detailed information.

`known subgoals*:` The known subgoals are updated as well. In comparison to the illustration of the subgoals at the beginning of this example, one can see that the lower left network is no longer displayed by a bold circle. The three sketches of the supply air ducts we added to the situation formulate new subgoals for the following design steps.

`case base*:` The justified state can be added to the case base which can be seen in the lower left case of this illustration.

Figure 1: CBR-Problem Solving in Building Design

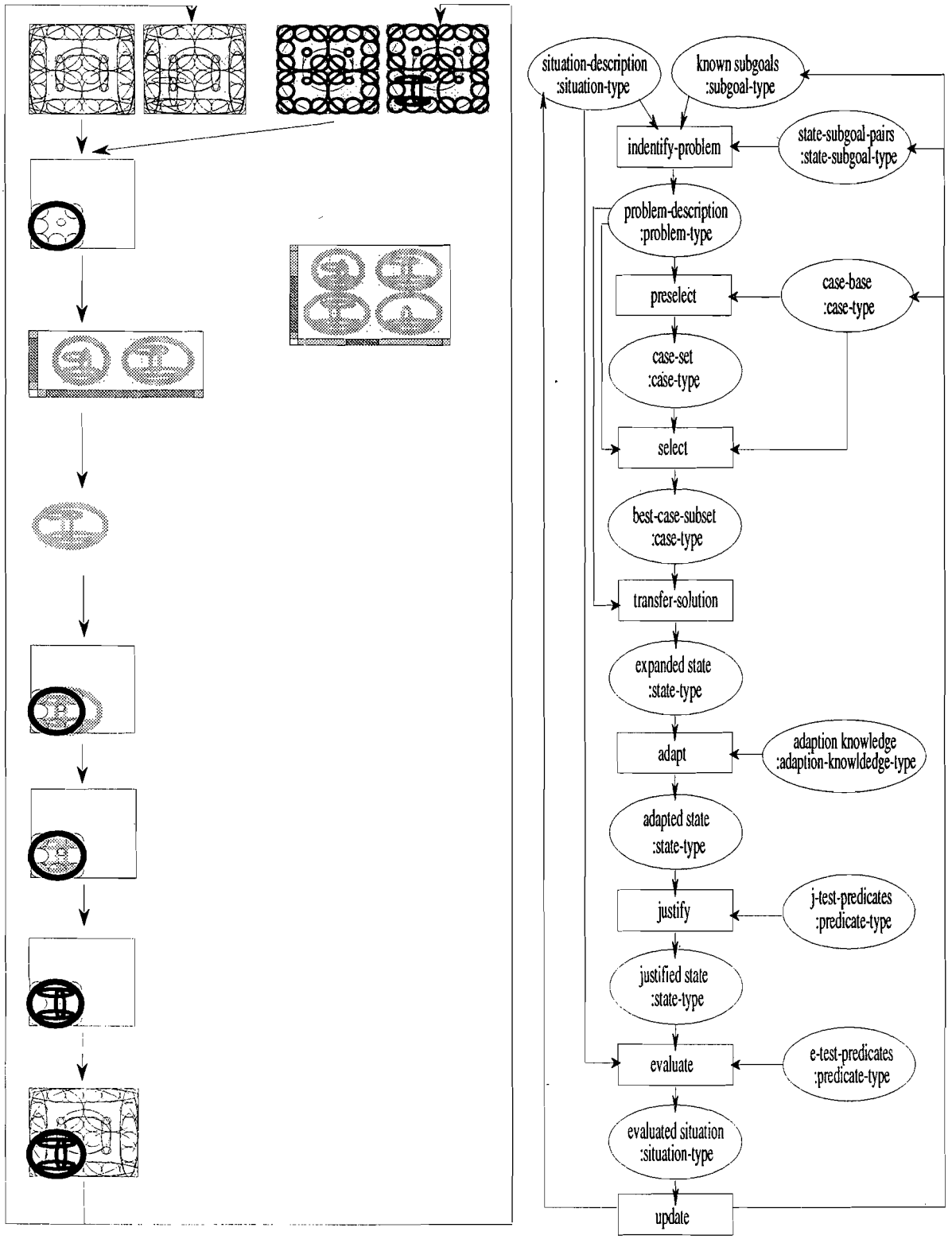


Figure 2: MoMo Inference Structure applied to building design

`adapted state` as such is free from errors. That is, `justify` is taking only the isolated state produced so far into account. `Justify` uses highly specialized `test-predicates` that check locally for common errors. There are test predicates for unconnected or conflicting pipes etc. The output of `justify` is a `justified-state`. This is a problem solving state which is error-free when viewed in isolation. `Evaluate`, the second global testing action, is used to find out if the `justified-state` may still be referred to as error-free once it is embedded into to the more complex `situation-description`. Again, highly specialized `test-predicates` may be employed. Once the `problem-description` together with the state that has been adopted from another case and adapted to the current requirements has passed both testing actions, the action `update` is used to add the new case to the `case-base`.

5 Conclusions

Interactions between tasks, methods, and knowledge may be exploited to improve knowledge acquisition and system design. We investigated these interactions by generating and discussing variants of the inference structure of our implementation. The top level task taken from the domain of building design remained the same across all investigations of interactions. These investigations proceed by systematically replacing lower level tasks and methods by alternative ones which results in different knowledge needs. In short, variants of inference strucutres are generated and the interactions between tasks, methods, and knowledge are investigated. As a result various triples of tasks, methods, and knowledge are obtained that reflect interactions between them. This will be presented in the extended version of this paper. Although these results are experimental in nature they pave the way to a more principled account of knowledge acquisition and system design in CBR-systems.

Acknowledgments

This research has been strongly inspired by work done in the project FABEL the general objective of which is the integration of case-based and model-based approaches in knowledge-based systems. We owe thanks to Gerhard Strube, Oliver Jeschke, Markus Knauff, Matthias Nückles for fruitful discussions about the ideas introduced in this paper. Nonetheless, the paper reflects our personal view to specific questions and tasks involved in knowledge acquisition and system development.

References

[1] B. Chandrasekaran, T. R. Johnson, and J. W. Smith. Task-structure analysis for modeling domain knowledge and problem solving for knowledge system construction. *Paper presented on a Workshop on Problem-Solving Methods, Stanford, July 9-11*, 1992.

[2] L. Hovestadt. Armilla4 - an integrated building model based on visualisation. In *Proc. EuropIA '93, Delft, The Netherlands*, to appear.

[3] D. Janetzko and K. Börner. Tasks-methods-knowledge interdependencies in CBR-systems. FABEL-Report No. 9, Gesellschaft für Mathematik und Datenverarbeitung (GMD). 1993.

[4] L. Steels. Components of expertise. *AI Magazine*, 11(2):28–49, 1990.

[5] J. Walther, A. Voß, M. Linster, Th. Hemmann, H. Voß, and W. Karbach. Momo. Technical report, Gesellschaft für Mathematik und Datenverarbeitung (GMD), 1992.

[6] B. Wielinga, A.T. Schreiber, and J.A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4:5–53, 1992.

Similarity-based Retrieval of Interpretation Models

Frank Maurer
University of Kaiserslautern
AG Prof. Richter
P. O. Box 3049
6750 Kaiserslautern
Germany
e-Mail: maurer@informatik.uni-kl.de

1.0 Introduction

Knowledge acquisition is a bottleneck of expert system development. Thereby, a special problem is the reuse of available methods. To overcome this problem, KADS [6] supports a library of interpretation models. Currently, this library contains only a few models so that humans are able to find an appropriate interpretation model for their task. In the future, expert system developer (or more general: software developers) will use huge libraries. Then the question of how to retrieve a model which is useful will raise. Our hypermedia-based knowledge engineering tool CoMo-Kit [1,2] tries to solve this question with a similarity-based or case-based approach.

To search for an interpretation model CoMo-Kit will use a similarity-based matching of graphs. We extend ideas from the PATDEX system [3, 5] to an object-oriented matching The work described within this paper are first ideas and should not be taken as finished research.

In chapter 2 we define cases and the similarity of cases. Chapter 3 deals with the retrieval of interpretation models.

2.0 Case descriptions

In our approach, a case represents a task with an associated method. The Method is described as a inference structure. Task descriptions which shall be reused are stored in a library and we call "interpretation models", in analogy to KADS. In contrast to KADS, our cases are not generic descriptions of tasks but domain specific structures. Every model of a domain which comes out as the result of a modelling process can be stored in the library.

The structure of a task (e.g. the data and control flow) is represented asa graph. Nodes and links of this graph are associated with types. In figure 1 a task structure is shown. Ellipses represent inferences (functions) whereas rectangles describe concept classes (data structures).

A knowledge engineer is able to build new task structures by using a set of predefined task types. Figure 2 shows the hierarchy of task types which is currently supported by CoMo-Kit.[1]

1. If needed, the set of task types can easily be extended.

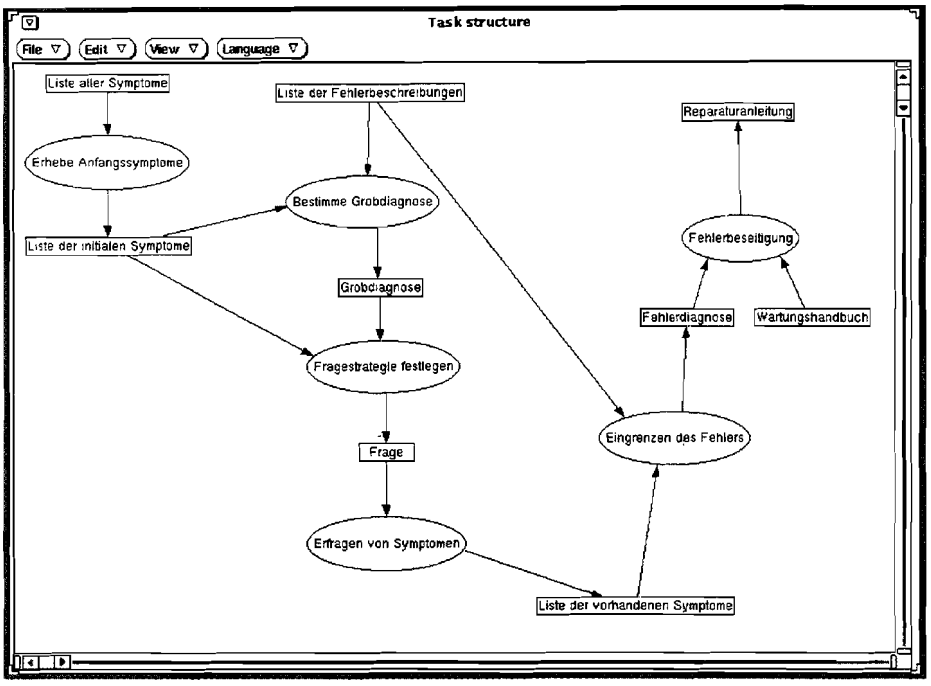


Figure 1: The method of a task

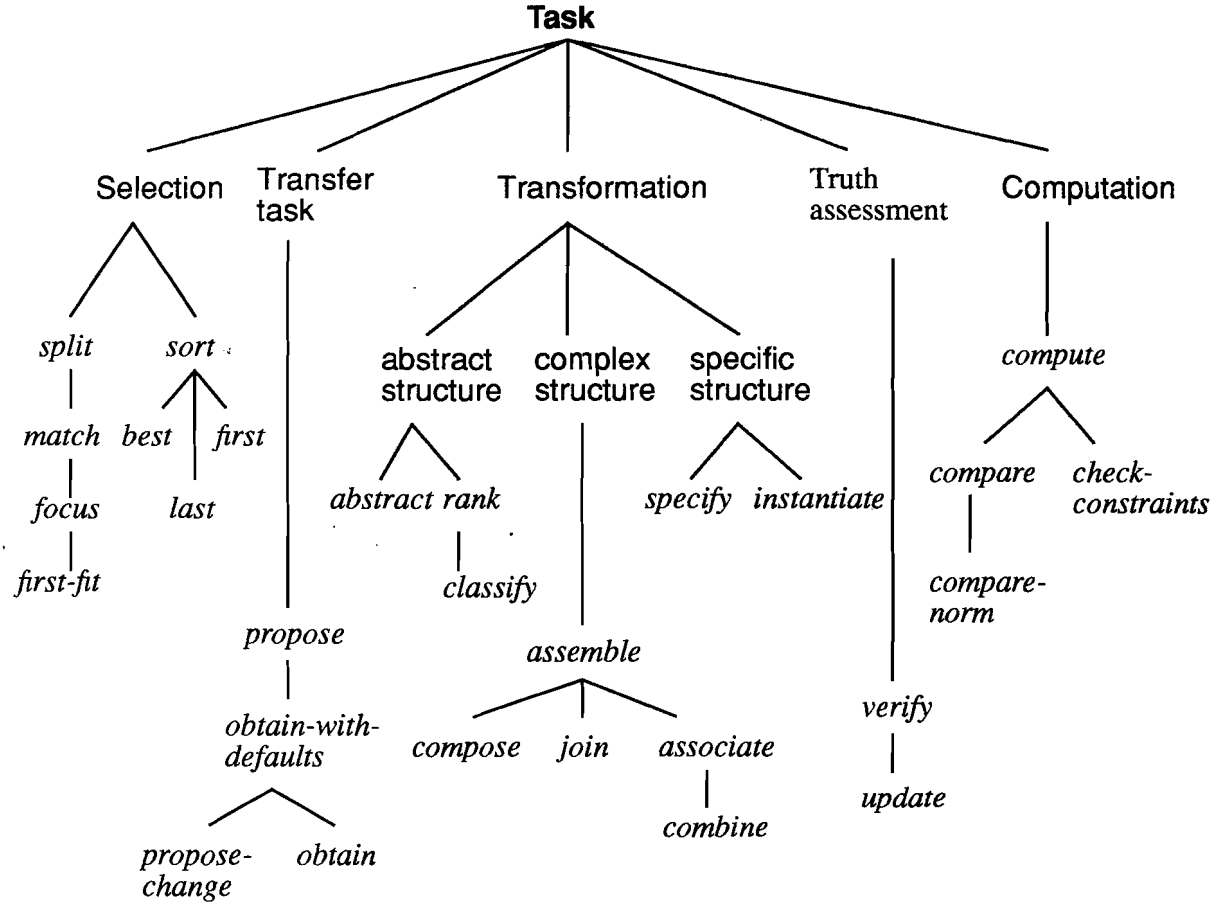


Figure 2: Taxonomy of predefined tasks

In the following, we define nodes and links. Nodes are objects and can be connected via directed links. The similarity of two links depends on the type of the link and the source and destination object.

Definition 1: Node class

A node class defines

- a superclass
- the attributes a_1,.., a_n of instances
- the types t_1,.., t_n of the attributes
- the similarity of nodes of the same class:

$$sim_C(S, T) \in [0,1] \quad \text{(EQ 1)}$$

The class *Object* does not have any attributes and is the root of the class hierarchy. The similarity of instances of Object is always zero:

$$sim_{Object}(S, T) = 0 \quad \text{(EQ 2)}$$

Definition 2: Node instance

An instance associates a value with every attribute a_i. The value must be included in the type of the attribute or it must be "unknown".

$$W_i(a_i) \in t_i \cup \{unknown\} \quad \text{(EQ 3)}$$

Definition 3: Link

A link is an directed arc between a source and a destination object. Links are typed, e.g. every link is an instance of a link class. For every link class a similarity function is defined.

$$sim_L(Link_1(S, T), Link_2(S', T')) \in [0,1] \quad \text{(EQ 4)}$$

Definition 4: Case

A case consists of a set of node and link instances. We *cannot* distinguish between problem description and solution. A problem description is a part of a graph. The solution is the rest.[1]

For every object of a case a weight is defined. Every case has an unique name.

$$Case(Name) \rightarrow \{Node_1, ..., Node_n, Link_1, ..., Link_m\} \quad \text{(EQ 5)}$$

$$\forall O_i \in Case(Name): Weight(O_i) \in [0,1] \quad \text{(EQ 6)}$$

2.1 The similarity of cases

To compute the similarity we see a case as a rule and the new problem description as the working memory. The algorithm computes the similarities for every binding (<Matching>) of the objects of the problem description to the objects of the case. The similarity of a case to the problem description is the weighted sum of the node and link similarities.

1. The impossibility to define which parts of a graph will be used as a problem description and which as the solution is the reason why an inductive approach is not applicable.

$sim_{Case}(C, Graph)$<Matching>=

$$\sum_i Weight(CO_i) \cdot sim_{C/L}(CO_i, Graph(\text{<Matching>})) \qquad \text{(EQ 7)}$$

A matching network is used to speed up the computation of the similarity. The matching network reduces the number of bindings which must be tested. Because of the typing not every node of a case must be matched with every node of the problem description. Only objects of the same class are compared. It is easy to extend this by matching also the instances of subclasses. The speed-up of the matching process is increased if the case-matching is incremental. This is the case for our application: We want to model a part of a task and then search for a similar case. If the retrieved case is not appropriate, the knowledge engineer extends the model and starts the search again.

3.0 Retrieval of Interpretation Models

Interpretation models are stored as graphs in a library, e.g. the matching network. To query the network for a similar case the user draws parts of a task structure (figure 1 shows the graphical interface). The defined node and link instances are propagated to the matching network and the similarity values of the stored cases are updated. The system presents a list of all retrieved interpretation models to the user who is able to choose one for his further work.

We used a similar approach for the retrieval of CAD drawings in architectural design (see [4]).

4.0 State of Realization

The knowledge engineering tool CoMo-Kit is fully implemented and allows to define task hierarchies, inference structures, classes with attributes, and instances. We now work on the implementation of the library. The tools for similarity-based retrieval of interpretation models are, as mentioned in the introduction, are up to now only ideas.

5.0 References

1. F. Maurer: CAKE: Computer-Aided Knowledge Engineering, IJCAI 91 WS "Software Engineering for Knowledge-Based Systems", Sydney 1991, also: SEKI Report SR-91-09
2. F. Maurer, G. Pews: A Distributed Operationalization of Conceptual Models, in: Proc. EKAW-93
3. M.M. Richter, S. Weß: Similarity, Uncertainty and Case-Based Reasoning in PATDEX, in: R.S. Boyer (ed.), Automated Reasoning, Essays in Honor of Woody Bledsoe, Kluwer Academic Press, also: SEKI Report SR-91-01 (SFB), Uni Kaiserslautern
4. A. Rippel: Ähnlichkeitsbasiertes Retrieval in Netzen (Similarity-based retrieval in networks), Diplomarbeit Universität Kaiserslautern, 1993
5. S. Weß: PATDEX/2: Ein System zum adaptiven, fallfokussierenden Lernen in technischen Diagnosesituationen (PATDEX/2: A System for adaptive case-focussed learning in technical diagnosis), SEKI Working Paper SWP-1-91, Uni Kaiserslautern
6. B.J. Wielenga, A.Th. Schreiber, J.A. Breuker: KADS: A Modelling Approach to Knowledge Engineering, Knowledge Acquisition Journal Special Issue on KADS, 1992

Chapter 8

Case-Based Explanation / Case-Based Tutoring

Using Logic to Reason with Cases[1]

Kevin D. Ashley and Vincent Aleven
University of Pittsburgh
Intelligent Systems Program,
School of Law, and Learning Research and Development Center
Pittsburgh, Pennsylvania 15260
ashley@vms.cis.pitt.edu, aleven+@pitt.edu
(412) 648-1495, 624-7039

Extended Abstract

1. Introduction

Before people can decide whether to rely on a Case-Based Reasoning (CBR) system's advice, they must understand the criteria according to which the system asserts that a case is relevant to a problem and more relevant than any other cases. In our application, tutoring students to reason with cases, our intelligent tutoring system, CATO, needs to explain its relevance criteria and illustrate them with examples. Moreover, it needs to deal with a variety of relevance criteria, some of which involve relations among multiple cases. The system designers, and eventually, teachers and students, need to be able to understand, use and modify the program's concepts for assessing case relevance and constructing case-based arguments. A logical representation of the relevance criteria provides the expressiveness and flexibility to make that possible.

At first blush, the choice of a logic representation to support case-based reasoning may seem odd. Case-based reasoning has often been contrasted with logical reasoning. First, logical deduction employs a formal inference mechanism like *modus ponens* to apply general rules to a specific problem. By contrast, a case-based reasoner draws inferences by comparing the problem to specific past cases and may use a variety of comparison methods [Ashley, 1993]. Second, CBR is nonmonotonic. For instance, a reasoner that could apply rules and cases might find that a problem matched not only a rule's antecedents but also an exception to the rule, leading it to abandon the rule's conclusion [Golding and Rosenbloom, 1991]. Third, in our domain of legal reasoning, logical representations are not ideal for representing statutory and court-made rules where concepts are open textured and usually there is no one right answer [Sergot *et al.*, 1986, Gardner, 1987]. Case-based reasoners have employed cases to represent the meanings of such terms and to generate competing reasonable arguments comparing the problem to conflicting cases [Rissland and Skalak1991, Branting, 1991].

On reflection, however, these valid objections do not imply that logic may have no role in implementing CBR systems. At an operational level, a CBR program needs to compute the relevance of cases. Although in most CBR programs to date, relevance concepts have been operationalized by structuring a program's memory (e.g., as a discrimination net) and building procedures to sort, select and filter cases (e.g., see [Koton, 1988, Sycara, 1987]), such concepts can be expressed in first-order logic and implemented by a deductive pattern-matcher. In developing CATO, we have employed the knowledge representation system Loom [MacGregor, 1991] to represent relevance and argument concepts declaratively in logic expressions. As long as the computational efficiency of the logically implemented relevance concepts is comparable to that of procedural representations, the advantages of a logic representation may be considerable.

The declarative logical representation offers important advantages for our CBR tutoring application, allowing us to: (1) Specify relevance criteria in terms of relationships among multiple cases, (2) Deal with multiple relevance criteria, (3) Communicate relevance criteria and illustrate them with examples, and (4) Support user queries of the case base. The primary significance of this work, however, has to do with explanation in CBR systems. We are beginning to address the question: how can case-based reasoners explain their reasoning and convince users of the plausibility of the system's conclusions?

In this paper we discuss these advantages and report on an experiment that we undertook to evaluate the computational efficiency of case retrieval with declaratively-defined relevance concepts.

2. Our Application: Tutoring with Cases

We are studying how to instruct law students to reason with cases, in particular, to evaluate problems by comparing them to past cases and to justify their legal conclusions by drawing analogies to selected precedents. The skills we instruct are important not only for attorneys in the American and English legal systems, where arguing by analogy to precedents is standard, but possibly also for reasoners in other disciplines, such as practical ethics, business and political science, where experts also reason with cases.

[1]This work is supported by an NSF Presidential Young Investigator Award and grants from the National Center for Automated Information Research and Digital Equipment Corporation.

Criteria for defining relevance and comparing cases may vary among and even within domains. For our domain, trade secrets law, we have defined relevant similarities and differences in terms of *factors*. Trade secrets law involves disputes in which a corporation (the plaintiff) complains in a law suit that a competitor or a former employee (the defendant) has gained an unfair competitive advantage by obtaining the corporation's confidential product development information, its trade secrets. A factor is a collection of facts that typically tends to strengthen or weaken the strength of the plaintiff's argument. Experts can list factors that typically strengthen a plaintiff's argument that the defendant misappropriated plaintiff's confidential information, as well as other factors that strengthen a defendant's argument.

An important class of legal arguments are about the importance of factors in particular circumstances: to what extent should certain factors determine the outcome of a problem. The attorney needs a method for resolving the conflict among the factors. In law, however, there are no authoritative weights of factors with which to resolve such conflicts [Ashley and Rissland, 1988, Ashley, 1990]. Instead, attorneys use certain rhetorical tools – we call them *Dialectical Examples* – to convince people that certain factors are more important. We have identified five standard ways of arguing with cases. Each Dialectical Example enables one to support or attack an assertion that a particular set of factors justifies a decision [Ashley and Aleven, 1992].

We introduce some basic argument building blocks and their associated relevance and argument concepts in Figure 1, a brief, annotated legal argument. The plaintiff (π) in the *Structural Dynamics* fact situation argues by analogy to a representative example, the *Analogic* case, which has no trumping counterexample (defined below). The defendant (δ) responds with a *ceteris paribus* comparison. Experienced arguers understand these concepts and have developed skill in applying them. Among other things, they know how to: identify factors, draw analogies to cases in terms of factors, point out relevant differences, cite counterexamples to a case, avoid picking a case to cite which is irrelevant or was won by the other side, prefer to cite more on point representative examples for which there are no trumping counterexamples, cover the opponent's bases and make *ceteris paribus* comparisons. We will say more about the building blocks in the final paper. For purposes of this extended abstract, let us focus on the defendant's use of a *ceteris paribus* comparison. A *ceteris paribus* comparison requires two cases with different outcomes that differ from each other only by a single factor, present in the problem. The single factor should be such that its presence can explain the difference in outcome between the two cases. As a rhetorical tool, a *ceteris paribus* comparison can be useful in justifying an assertion that a particular factor is important enough to justify a particular outcome in the problem. In Figure 1, the defendant uses the *ceteris paribus* comparison to argue that one factor, the employee's being a sole developer of the the product, is important.

To instruct law students in these argumentation skills, we use small, carefully selected combinations of cases, called *Argument Contexts*. Argument Contexts illustrate the Dialectical Examples and present concrete circumstances in which to practice and develop the skills. For instance, a law instructor could use two cases, related in just the right way, to illustrate the concept of a *ceteris paribus* comparison. Also, the Argument Context shown in Figure 2 can be used to instruct students about the kind of representative example to choose to cite for the plaintiff in the argument shown in Figure 1. It is in the form of a Claim Lattice, a knowledge structure developed for the HYPO program [Ashley, 1990, pp.55-57]. The root node represents the *Structural Dynamics* case and the set of all of its applicable factors. Each case in the body of the Claim Lattice shares some subset of that set of factors; the nodes are ordered in terms of the inclusiveness of that set. That is, *Eastern Marble* has a subset of *Amoco's* set and therefore is less on point than *Amoco*. The same is true of *Schulenburg* and *Analogic*. With this Argument Context, and instructor could ask the student, "Which case should the plaintiff cite?" Three of the cases were won by plaintiffs, so there are three possibilities. If the plaintiff cites *Eastern Marble*, however, the defendant could respond by citing *Amoco* as a trumping counterexample. It was won by defendant, shares everything with the cfs that *Eastern Marble* does, but also shares an additional factor, F3. *Analogic* is the best case to cite. It is more on point than *Schulenburg* and cannot be trumped. In a preliminary experiment, we employed program-generated Argument Contexts manually to teach basic argument skills to first year law students with good results [Aleven and Ashley, 1992]. The program that generated the Argument Contexts is one module of a future case argument tutoring system, CATO.

3. A Declarative Representation for CBR

As the above example suggests, we teach students how to use comparisons of a problem to past cases as warrants in arguments justifying assertions about the problem. In our domain and task the relevance criteria are part of the warrant. In order to perceive the force of an argument comparing a problem to a precedent, one needs to understand the sense in which an arguer regards the case as relevant. That is why we need an express, explainable representation for the relevance criteria.

Our program, CATO, uses an explicit representation of relevance concepts to support its case retrieval functions: To find cases or combinations of cases that an arguer can use in an argument or that a law teacher can use as examples to illustrate lessons about argumentation (Argument Contexts). Currently, CATO's case base contains 31 legal cases which will soon be increased to one hundred cases. Individual

cases are represented as lists of factors. This has proven adequate for selecting most types of Argument Contexts.

CATO's relevance concepts are defined in a knowledge base implemented in Loom [MacGregor, 1991], a KL-ONE style knowledge representation system. As is typical for systems of this family, Loom offers a terminological language for defining structured concepts and relations, as well as reasoning facilities including automatic classification of concepts and automatic recognition of the instances of concepts. In Loom, one can also express definitions for concepts or relations in Loom's first-order logic query language. CATO uses all these reasoning modes, but relies mostly on the facility to state and apply definitions expressed in first-order logic.

Some of the concepts and relations in CATO's knowledge base are listed in Figure 3. Figures 4 and 5 show examples of relevance criteria expressed in Loom's query language. All terms (predicates) that are referred by these definitions are also defined in the knowledge base. In the final paper, we will illustrate other concepts and relations and their definitions. Suffice it to point out that that the logical expressions are a natural way for describing the crucial relationships among cases and factors.

Case retrieval is a matter of finding cases that instantiate a given relevance criterion. Once a relevance criterion has been expressed declaratively, as a Loom definition, Loom's query interpreter takes care of the rest. It searches the case base for cases (or combinations of cases) that satisfy the definition. No additional coding is necessary to implement case retrieval[2]. For example, Loom can apply the definitions shown in Figures 4 and 5 to find best untrumped cases or *ceteris paribus* comparisons. One interesting aspect of the representation is this: In queries, each argument of the relation can be, but does not have to be, instantiated. Therefore, retrieving *ceteris paribus* comparisons relevant to a given factor is just as easy as retrieving *all* such comparisons that can be found in the case base. For a student developing an argument, the former query is more useful, for a law teacher looking for training examples, the latter.

A teacher can also employ more complex queries for Argument Contexts to use as training examples. The query used to retrieve the five-case Argument Context of Figure 2 is shown in Figure 6. We have implemented an Argument Context generation program, which was mentioned in the previous section. For certain very useful queries, like this one, this program presents menus of parameters for the user to fill out, generates a version of the query based on the parameter values, retrieves cases and then enables the user to filter and rank cases.

Ultimately, our goal is to develop a tutoring system that teaches law students the argumentation skills described in the previous section. Currently, CATO has no pedagogical capabilities other than the generation of Argument Contexts. It cannot produce natural language explanations of relevance concepts or give a student feedback and advice in developing an argument. However, we believe that the express representation of relevance concepts makes it easier to develop these functions.

4. Merits of Our Declarative Representation

The declarative logical representation offers important advantages for our CBR tutoring application. Our goal is to develop a program that teaches students to use cases in arguments to justify legal conclusions, based on a model of reasoning with Dialectical Examples. We need to deal with relevance criteria in ways that have been relatively unusual in CBR work so far, but which we anticipate will become increasingly useful.

Specifying relevance criteria in terms of relationships among multiple cases. Many of the Dialectical Examples, such as the *ceteris paribus* comparison, involve comparisons of multiple cases. The relationships among the cases effect the nature and quality of the warrant. A *ceteris paribus* comparison is more convincing if the factor of interest is the only difference between two cases with opposite results. A case is the best untrumped case to cite only if it has no trumping counterexamples. Relevance criteria, therefore, are naturally expressed in terms of the relationships among multiple cases and factors. These relationships among multiple cases can be conveniently expressed in first-order logic. For example, the requirement that the two cases in a *ceteris paribus* comparison must have each other's pro-winner factors can be concisely stated (see above). One can also quite naturally specify a relationship that should *not* be present among any cases in the database. For instance, the definition of an untrumped best case in Figure 4 states that for every case in the database it shall not be a counterexample to the best case. Complex conditions that involve any number of cases interrelated by multiple relationships can be expressed easily.

Dealing with multiple relevance criteria. The Dialectical Examples illustrate just a handful of the many different ways a comparison with past cases can be used to justify a conclusion about a problem. Each comparison uses cases related in ways specific to that comparison. Therefore, CATO needs to deal

[2]Loom evaluates a query by translating it into Lisp code that implements an exhaustive search. It thenexecutes this code to find all values for the query variable(s) that satisfy the query constraints.

with multiple criteria for case relevance. Also, as we identify additional ways of arguing with cases, new relevance criteria need to be defined. The declarative representation facilitates the implementation of multiple relevance criteria and the prototyping of new relevance criteria, because relevance criteria can be expressed concisely and conveniently in first-order logic and can be modified easily.

It should also be easy to modify a relevance criterion to adapt to particular circumstances. Given certain problems and certain case databases, a relevance criterion may be too strict or too loose. If too loose, too many cases may satisfy the criterion. If too strict, no case may satisfy the constraints. Yet, a suitably relaxed relevance criterion might retrieve cases which are nearly as useful. A declarative logic representation is very flexible: It allows one easily to formulate looser or stricter queries by removing, adding or modifying conditions. For instance, the initial versions of the *ceteris paribus* criterion were too strict, so we relaxed the constraints to allow the cases to differ by more than one factor.

Communicating relevance criteria and illustrating them with examples. Our system needs to explain relevance criteria because students have to learn them. Since relevance criteria serve as components of warrants they need to be defined in symbolic terms, not numerically. Since an arguer may be compelled to defend his assertion that a case is relevant, the significance of the relevant similarities and differences had better not have been converted into numbers. Defining the terms symbolically in terms of a declarative logical representation makes them easier to communicate and explain to the user. In part, this is because logically-defined definitions are easier for the user to understand. In addition, however, we have found that they are easier to illustrate with examples.

We illustrate relevance criteria with various types of examples. In order to illustrate a relevance criterion, we can search for collections of a problem and cases that instantiate the relevance criterion. This means that CATO can retrieve *all* instances of best untrumped cases to cite, of *ceteris paribus* comparisons, or of cover-the-opponent's-bases situations involving *any* case in the case base as problem situation. For most CBR systems, this kind of retrieval would not be possible. We use these as examples in exercises (these examples are Argument Contexts) to illustrate the relevance criterion and instruct students by example how to employ them.

Supporting user queries of case base. We intend the student users to query the case database. Using CATO, we want to teach them to formulate better queries for relevant cases (a skill we expect will transfer to other legal information retrieval systems like Lexis, Westlaw and West's natural language query system, WIN). In addition, a student's queries serve as a test of his/her understanding of the relevance criteria. It follows that queries need to be easy to understand, express, modify, and execute. In addition, our system needs to be able to deal with a range of queries that cannot be anticipated completely in advance. We believe a declarative logical representation is the most likely of all the available alternatives to be manipulable by student users. We plan to design a simplified and specialized query language for student users. This language can be implemented by translating the student queries into first-order logic queries.

Explanation in CBR systems. We think the advantages of a declarative logical representation have significance for CBR beyond our tutoring application. The CBR community has not adequately addressed the question: how can case-based reasoners explain their reasoning and convince users of the plausibility of the system's conclusions? (See [Ashley, 1993].) There are at least five ways:

1. Show the user a similar precedent. Such an explanation may involve mapping and adapting an explanation from the precedent to the problem as in SWALE [Kass *et al.*, 1986], CASEY [Koton, 1988], GREBE [Branting, 1991], or integrating the precedent into a rule-based explanation as in CABARET [Rissland and Skalak, 1991]. The precedent, however, is only part of the warrant represented by the case comparison.
2. Some CBR programs, like CASEY, can justify why the precedent matches the original.
3. In addition, a program could explain why the particular precedent is a better match then other candidates (HYPO);
4. In addition, the program could explain its criterion for justifying the match or for considering one case to be better than another.
5. In addition, the program could explain why the criterion matters in terms of the theory of the domain and task.

This work on CATO focuses on the third and fourth methods. By representing relevance criteria declaratively, we have made some progress in enabling a program to explain aspects of its relevance criteria by example. The approach can be related to another CBR program. CASEY justifies a match between a new case and a past case on the basis of its "evidence principles", domain-independent rules for adapting causal explanations. Would it be useful for CASEY to illustrate a given evidence principle by retrieving two cases which this evidence principle justifies calling similar? For purposes of tutoring or explanation, we believe so. For instance, a user might say: "I don't understand your explanation. Why is this past case a good match?" A simple example of a match justified by the evidence principle would help make a good

5. Empirical Efficiency Analysis

We conducted an experiment to evaluate the efficiency of case-based reasoning with declaratively represented relevance concepts. We collected timing information for various queries using (synthetic, computer-generated) case bases ranging in size from 26 to 250 cases. The queries that we used represent the whole spectrum that we have described in this paper and include queries for best untrumped cases, *ceteris paribus* comparisons, and 5-case Argument Contexts. (The queries were very similar—though not identical—to the ones shown in Figures 4, 5, and 6.) In this experiment, we used two techniques to to speed up the query for 5-case Argument Contexts. We added constraints to the query in order to reduce the amount of search ("query reformulation"). Also, we precomputed certain often-referenced information and stored it in tables, thus trading space for time ("tables").

The run times that we measured are shown in Figure 7. (The results were obtained running Loom 1.4.1. on a DECstation 5000/240.) When Loom evaluates a query, it searches for *all* cases (or combinations of cases) that satisfy the query. In other words, it performs an *exhaustive search.* The timing results should be interpreted with this in mind. Not suprisingly, the observed run times correspond to the asymptotical time complexity of the queries, which is $O(N^2)$ for best untrumped cases, $O(N^3)$ for *ceteris paribus* comparisons, and $O(N^5)$ for 5-case Argument Contexts, where N is the number of cases in the case base. The standard CBR operation of retrieving the cases that are most relevant to a given problem (best untrumped cases) took less than 20 seconds with a 250-case database. While some of the other retrieval times times may seem rather long, it should be noted that the queries for *ceteris paribus* comparisons and 5-case Argument Contexts, when run with the larger case bases, retrieved thousands of case combinations. This is more than a law instructor looking for examples could possibly need. Clearly, exhaustive search is not necessary for these queries. Therefore, we conclude that case retrieval based on declaratively represented relevance concepts is not prohibitively expensive. In the final paper, we will discuss the experiment in greater depth.

6. Conclusion

Our application, tutoring students to reason with cases, necessitated adopting a declarative representation of case-based relevance concepts. Using Loom's first-order logic query language, relevance concepts can be conveniently expressed in terms of relations among multiple cases. Loom's query interpreter is used to do case retrieval. This has turned out not to be prohibitively expensive computationally.

Representing relevance criteria in first-order logic has considerable advantages. First, the declarative representation facilitates the use of multiple, changing relevance criteria, since it allows the criteria to be expressed and/or modified with great ease. Second, students using CATO will eventually express their own queries for CATO to interpret. We expect that the development of a simplified and specialized query language for this purpose will be greatly facilitated by a declarative representation of the underlying relevance concepts.

Finally, by representing relevance criteria declaratively, we have made some progress in enabling a program to explain aspects of its relevance criteria by example. Our work on CATO focuses on enabling a program to explain why the particular precedent is a better match then other candidates and to explain its criteria for justifying the match or for considering one case to be better than another. We believe that as CBR system designers confront the problem of building programs that can explain their results, a logical representation of relevance concepts will be useful.

References

Aleven, Vincent and Ashley, Kevin D. 1992. Automated Generation of Examples for a Tutorial in Case-Based Argumentation. In C. Frasson, G. Gauthier, and G.I. and McCalla (eds.), *Proceedings of the Second International Conference on Intelligent Tutoring Systems*, 576–584. Berlin, Springer-Verlag.

Ashley, Kevin D. and Aleven, Vincent 1992. Generating Dialectical Examples Automatically. In *Proceedings AAAI-92*, 654–660. Menlo Park, CA: AAAI Press.

Ashley, Kevin D. and Rissland, Edwina L. 1988. Waiting on Weighting: A Symbolic Least Commitment Approach. In *Proceedings AAAI-88*, 239–244. Distributed by Morgan Kaufmann, San Mateo, CA.

Ashley, Kevin D. 1990. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals.* MIT Press, Cambridge.

Ashley, Kevin D. 1993. Case-Based Reasoning and its Implications for Legal Expert Systems. *Artificial Intelligence and Law* 1(2).

Branting, L. Karl 1991. Building Explanations from Rules and Structured Cases. *International Journal of Man-Machine Studies* 34(6):797–837.

Gardner, A. vdL. 1987. *An Artificial Intelligence Approach to Legal Reasoning.* MIT Press, Cambridge.

Golding, Andrew R. and Rosenbloom, Paul S. 1991. Improving Rule-Based Systems through Case-Based Reasoning. In *Proceedings AAAI-91*, 22–27. Menlo Park, CA: AAAI Press.

Kass, A. M.; Leake, D.; and Owens, C. C. 1986. Swale: A Program that Explains. In Roger C. Schanck (ed.), *Explanation Patterns: Understanding Mechanically and Creatively.* Hillsdale, NJ: Lawrence Erlbaum.

Koton, Phyllis 1988. *Using Experience in Learning and Problem Solving.* Ph.D. Dissertation, MIT.

MacGregor, Robert 1991. The Evolving Technology of Classification-Based Knowledge Representation Systems. In John F. Sowa (ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, 385–400. San Mateo, CA: Morgan Kaufmann.

Rissland, Edwina L. and Skalak, David B. 1991. CABARET: Statutory Interpretation in a Hybrid Architecture. *International Journal of Man-Machine Studies* 34(6):839–887.

Sergot, M. J.; Sadri, F.; Kowalski, R. A.; Kriwaczek, F.; Hammond, P.; and Cory, H. T. 1986. The British Nationality Act as a Logic Program. *Communications of the ACM* 29(5):370–386.

Sycara, Katia 1987. *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods.* Ph.D. Dissertation, Georgia Institute of Technology. School of Information and Computer Science, Technical Report No. 87-26.

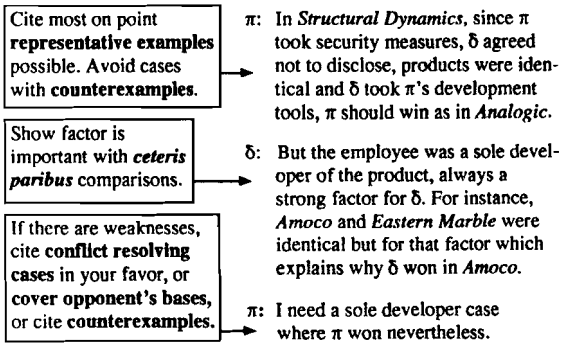


Figure 1: Dialectical Examples as Building Blocks

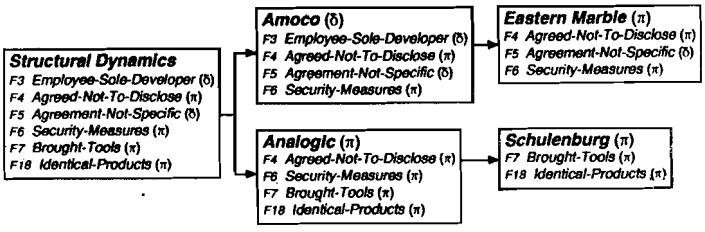


Figure 2: Argument Context for "Select Best Case" task.

```
(defrelation untrumped-best-case
  :domains (Precedent Case)
  :range Side
  :is (:satisfies (?c ?cfs ?s)
        (:and (Case ?c)
              (Case ?cfs)
              (Side ?s)
              (best-case-to-cite ?c ?cfs ?s)
              (:for-all ?cex
                (:implies
                  (Case ?cex)
                  (:not (trumping-cex
                          ?cex ?c ?cfs))))))
  :attributes :multiple-valued)
```

"Case `?c` is an untrumped best case for side `?s`, with respect to problem situation `?cfs`, if `?c` is a best case to cite for `?s`, and if for every case `?cex`, `?cex` is not a trumping counterexample for `?c`"

Figure 4: Definition of Untrumped Best Case.

```
(defrelation ceteris-paribus
  :is (:satisfies (?c1 ?c2 ?f)
        (:and (Case ?c1)
              (Case ?c2)
              (Factor ?f)
              (opposite (outcome ?c1)
                        (outcome ?c2))
              (pro-winner-factor ?c1 ?f)
              (:not (applicable-factor ?c2 ?f))
              (:for-all ?f1
                (:implies
                  (:and (Factor ?f1)
                        (pro-winner-factor ?c1 ?f1)
                        (neq ?f1 ?f))
                  (applicable-factor ?c2 ?f1)))
              (:for-all ?f1
                (:implies
                  (:and (Factor ?f1)
                        (pro-winner-factor ?c2 ?f1))
                  (applicable-factor ?c1 ?f1))))))
```

"Cases `?c1` and `?c2` make a *ceteris paribus* comparison for factor `?f`, if: `?c1` and `?c2` have opposite outcomes; `?f` is a pro-winner factor of `?c1` but does not apply in `?c2`; all pro-winner factors of `?c1`, except `?f`, also apply in `?c2`; vice versa."

Figure 5: A definition of *ceteris paribus* comparison.

```
(retrieve (?cfs ?c1 ?c2 ?c3 ?c4)
  (:and
    (Pro-P-Case ?c1)
    (Pro-P-Case ?c2)
    (Pro-P-Case ?c3)
    (Pro-D-Case ?c4)
    (Case ?cfs)
    (citable ?c1 ?cfs plaintiff)
    (citable ?c2 ?cfs plaintiff)
    (citable ?c3 ?cfs plaintiff)
    (more-on-point ?c2 ?c3 ?cfs)
    (trumping-cex ?c4 ?c1 ?cfs)
    (unordered ?c1 ?c2 ?cfs)
    (unordered ?c3 ?c4 ?cfs)))
```

"Retrieve cases `?cfs`, `?c1`, `?c2`, `?c3`, `?c4` such that: `?c1`, `?c2`, and `?c3` are citable for the plaintiff; `?c2` is more on point than `?c3`; `?c4` is a trumping counterexample for `?c1`; cases `?c2` and `?c3` are in a different branch than `?c1` and `?c4`."

Figure 6: Query to retrieve 5-case Argument Contexts.

Concepts and Relations		
Primitive Concepts	**Relevance Concepts**	**Pedagogical Concepts**
Case Factor Side (π or δ) Outcome Applicable Factor Favors	Relevantly Similar Citable Best Case to Cite Untrumped Best Case Trumping Counterexample *Ceteris Paribus* Cover the Bases *etc.*	Vanilla Case Packed Case Conflicting Factors Case Unordered *etc.*
Facts		
For 20 factors: – name – side that it favors	For 31 cases: – name, cite – applicable factors – outcome (π or δ)	

Figure 3: Knowledge Base for Case-Based Argumentation

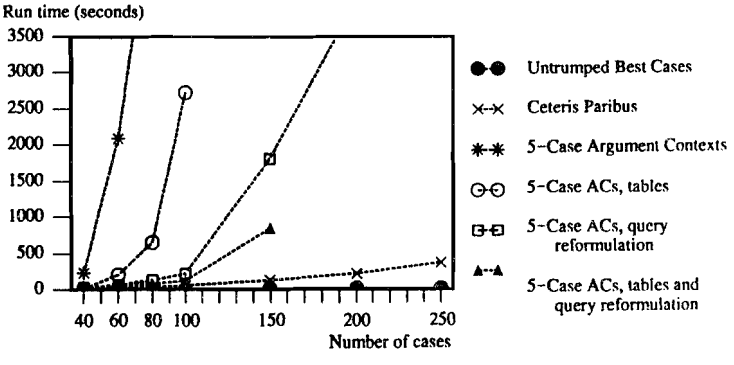


Figure 7: Efficiency of various queries as a function of case base size

Multiple Explanation Patterns

Uri J. Schild, Yaakov Kerner[1]

Department of Mathematics and Computer Science
Bar Ilan University, Ramat-Gan 52900, Israel
schild@bimacs.biu.ac.il

Abstract

In the Case-Based Reasoning paradigm cases are often given initially in natural language in the form of a "story". While this textual form is appropriate for humans, it is often not suitable for direct application by a computer. Our paper uses the legal domain of sentencing for criminal offences to illustrate an approach to indexing, knowledge representation of stories and their application in quantitative reasoning. This approach extends the well-known concept of Explanation Patterns.

Keywords: CBR, Explanation Patterns, Legal Applications.

1. Introduction.

Our object of interest is the domain of 'stories' (see, e.g., [Schank90] for a detailed discussion of this concept). When a human reasons about a situation in the present, he is often reminded of stories that he has heard or actually experienced himself in the past. He may then attempt to understand and explain the case at hand based on those stories.

Our basic idea is to apply this *explanatory* approach to a domain where explanations of previous cases lead to a quantitative result. Such a domain is the area of criminal sentencing. Judges are often reminded of previous cases with similar features when passing sentence. Indeed, the sentences of some previous cases (precedents) are even of binding importance. The cases are of course our 'stories', and the sentence itself is the quantitative result attached to the story.

We are motivated by our desire to build a computer system that may support a judge in deciding which sentence to hand down in a new case. The areas we have considered are Robbery and Rape (and some other sexual offences). Both are areas with maximum sentences (according to the Israeli law) of twenty years.

Such a system should not supply its user with a single, definite answer (i.e., a proposed sentence). No judge would appreciate that. We propose an intelligent decision support system, where several approaches and ways of reasoning will be produced for the user, but the final choice will be his only.

The questions we shall deal with are: How should such stories be represented in a computer, and how may they be retrieved by a case-based reasoner in order to obtain such a quantitative result also for a new story? The actual, quantitative application of the retrieved stories will not be dealt with in this paper.

The layout of the paper is as follows. In section two we shall consider previous work relating to the representation and use of stories. In section three we shall propose a generalization of one such approach and its adaption to our domain. This will establish the theoretical background relating to the knowledge representation and retrieval in a practical system which is at present under implementation by us. Section four will summarize the paper and suggest future developments.

2. Background.

2.1 An Example.

Consider the following (true) story:

A couple is standing on a nice summer-day on the beach in Natanya (an Israeli seaside resort). They are in

[1] This work is in partial fullfilment of the requirements towards the degree of Ph.D. by the second author under the supervision of the first author.

bathing suits, and the woman is wearing a gold-chain around her neck. The chain is rather thin and quite inexpensive. A youngster rides up to them on a horse (!), bends down, takes hold of the chain, tears it off the woman and gallops away. He is eventually apprehended and found guilty of robbery (crime never really does pay). His sentence is relatively heavy: Two and a half years in prison. The judge explains that this kind of robbery is usually not considered a serious crime. The main reason for the sentence is that the sheer audacity of the young robber is taken to be as an aggravating circumstance: To snatch the chain in broad daylight on a public beach must be considered the height of insolence.

For the purpose of this research we have been conducting interviews with judges from the District and Appeals Court in Tel-Aviv. We usually prompt a judge by asking him to tell us any story that comes into his mind relating to robbery or rape (and sentencing). In this particular case the judge reacted to our prompt by saying: "Oh, I shall tell you about the impudent youngster on the horse". In almost all our interviews the judges have *automatically* given titles to their stories at some stage of the story-telling. We have taken such a title to indicate the *index* for retrieval of the story, as indeed it appears to be in the case just described: The insolence of the robber was considered the main feature in the judge's reasoning leading to the determination of the sentence.

In this case the index indicates aggravating circumstances, while other stories and their title may indicate mitigating circumstances. Our assumption is that when the judge mentioned above (and perhaps also other judges familiar with the story) encounters another case involving an audacious crime, he will be reminded of this story. He will also remember the severity of the sentence - or rather the *reasoning* behind that sentence as a factor (possibly among others), that may contribute to the decision in the case at hand.

In other words, in the present case he will choose such a sentence that it may be explained on the basis of the previous story or stories. This is explanation-based retrieval and reasoning. We shall supply further justification for this approach below, in section 3.1.

This assumes, of course, that judges are consistent in their sentence-passing. While some undoubtly are, others may not be so. The general public feels that judgments and sentencing should be consistent and uniform, and it is our suggestion that a computer system of the kind described here may contribute to attain such uniformity. We do not attempt to build psychological models, or perform cognitive simulation. However, we believe that the actual use of such a system will ensure that a judge is in possession of relevant background information (precedents).

2.2 Related Work

A 'story' is often considered as consisting of a sequence of *episodes*, i.e. events, actions, situations, etc, and the relationships among such episodes. It is dynamic with little or no hierarchical structure. Classical knowledge structures like semantic networks cannot in themselves suffice for representing stories. Such basic structures are appropriate for representing certain static aspects of the stories, but cannot cover the overall picture.

A *script* ([Schank77]) may actually be an appropriate form for knowledge representation for the legal process itself, as it may be used to describe ordinary and routine activities. However, scripts are not appropriate for describing the reasoning leading to the sentence imposed by a judge.

Also the Memory Organization Packet (MOP) (see [Schank82], [Kolodner83])), which generalizes the script describes stereotypical events and does not enable the kind of explanations we seek.

Narrative understanding systems, e.g., CYRUS ([Kolodner81]), BORIS ([Lehnert83], [Dyer83]), and MEDIATOR ([Kolodner85]) are not applicable here, as our object is not natural-laguage understanding or story-understanding, but the application of understood stories to a new story, which needs to be explained.

One could possibly use various kinds of logics, e.g. Episodic Logic ([Schubert89]). However, as the originators of such logics usually acknowledge themselves, much work remains to be done on these logics before they become applicable in practical systems.

JUDGE is a case-based system in the legal domain, which attempts to model the behaviour of judges, when passing sentence ([Bain89]). As such, it is of course very relevant to our work. All the conclusions drawn by Bain concerning the behaviour of judges are supported by our own experience. However, the aims of our work differs from his.

Bain's system computes sentences by essentially mapping a partial ordering of crime heinousness onto a partial ordering of sentence type and durations ([Bain89], p.113). It operates in the area of murder, manslaughter and assault. The method is to infer the motives of the actors of the crime and decide on the degree to which each was justified in acting. No other parameters besides heinousness are taken into account, while our object is to address the general problem.

JUDGE uses a single precedent. If the new case is similar to the old one in all the predefined aspects - the same sentence is decided upon. If not, the system decides whether the new case has aggravating or mitigating features with respect to the old case. Such features can be: Unprovoked Violence, Self Defence, etc. The system then modifies the old sentence accordingly, e.g., it may add or subtract 25% of the old sentence.

Our approach is different. Firstly, our indexing scheme is explanation based. Secondly, we propose that the system should attempt to reason with several related cases, and consider several parameters. It should not deliver one final answer, but rather present the user with several arguments, that may even be conflicting. As we have already stressed, the purpose is to construct a sentencing support system, and not a sentencing system.

2.3 Swale and Abe.

Swale ([Kass86a], [Kass86b]) is a computer system which produces creative explanations for non-standard stories. Abe ([Kass89]) is both a simplified and extended version of Swale. The system defines the concept of an 'explanation pattern' (XP) for a story. It uses the explanation patterns for stories in the database to explain a 'gap' in the explanation of a new story. If these explanations cannot be applied in a straightforward manner, the system has a number of adaptation strategies.

As a concrete example, assume that the database contains the following two stories (originally given by Kass, and here considerably shortened) and their XPs:

(1) A famous sportsman suddenly collapsed and died. The XP is: Unknown to everybody he had a weak heart.
(2) A otherwise healthy lady suddenly died. The XP is: Her husband killed her in order to obtain the insurance money.

We now consider a new story, based on an actual case: A famous racehorse (called Swale) suddenly collapsed and died. The 'gap' here is why the horse died suddenly. The system will adapt and apply the two previous XPs and suggest two possible explanations for the gap:

(1) Unknown to the owner and trainer the horse had a weak heart.
(2) The owner killed the horse in order to obtain the insurance money.

The use of an XP in connection with a gap can actually work two ways. Given a story with a gap one can look for stories with an XP to explain the gap. Conversely, given a gap and its explanation one can look for an appropriate XP in order to *justify* the explanation.

A similar situation also occurs in the legal domain. Sometimes a judge will decide on a sentence after considering the old stories. Sometimes he will decide on a sentence, and then see how to justify it (both to himself and to the world), by finding the appropriate precedents. This latter possibility is well-known and acknowledged by the judges themselves (they sometimes say they have a 'gut-feeling' of what the sentence should be).

3. Our Knowledge Structures.

3.1 Indexing by Explanations.

One may suppose that *predictive explanations* ([Schank86], p.32) would be appropriate in the sentencing domain. This, however, would necessitate extensive knowledge about the judges themselves, their outlook, behavioural patterns, etc. Indeed, this approach seems to be the one taken in JUDGE (see section 2.3). We do not believe such information to be readily available or dependable, and have chosen a different approach.

Intent explanations ([Schank86], p.32) are given when one has to interpret the behaviour of agents according to their motives. This is the approach we have chosen. We must therefore decide how to index our stories accordingly.

In section 2.1 we explained how the indexing was actually supplied by the judges themselves, when they chose names for their stories. Thus "The impudent youngster on the horse" yields an index called aggravation-through-insolent-behaviour (of offender). Most of those indices were of a general type, but some were unique for the type of crime (robbery, rape, etc.)

We were not, however, satisfied with this approach alone. Independently we elicited knowledge about the sentencing process from a judge not involved in the story-telling. This knowledge was used in creating a discrimination tree. It then became apparent that the nodes in the discrimination tree and the indices derived from the story titles were identical. A further correspondence was also obtained by considering results obtained from expert criminologists and jurists. They have supplied us with what they call 'sentencing parameters', which are essentially equivalent to our explanatory indices.

A final test of the indices has been planned, but not yet carried out. We intend to use statistical data about criminal offenders and their sentences over the last ten years, as compiled by the Israeli police. Our hypothesis is that there is a strong correlation between sentences and our indices.

3.2 Judicial XPs

An explanation pattern includes the following aspects: (1) facts, (2) beliefs, (3) purpose, (4) plan, (5) action (behaviour). The XPs developed by Kass et al. (see section 2.4) have this structure, which we shall also adopt:

(1) Facts: These are the indices, as explained in the previous section.
(2) Beliefs: Additional relevant knowledge.
(3) Purpose: We here consider the criminological approach: Retribution, Prevention, Deterrence or Rehabilitation as the purpose behind a given sentence.
(4) Plan: We aim at using the explanations of aggravating or mitigating factors according to facts and beliefs.
(5) Action: The sentence itself, or rather its deviation from the standard (tariff).

An example will here be in order:

A man was arrested and found guilty of indecent exposure (paragraph 349 aleph of the Israeli criminal law). He had no previous offences, and received a suspended sentence of three months. This is an exceptionally light sentence relative to the average sentence for infraction of paragraph 349 aleph.

MOP:	XP: 'The First-Timer'
accused according to 349 aleph	facts: first offence
found guilty	beliefs: not dangerous to public
standard sentence: 1 month	purpose: retribution
maximal sentence: 1 year	plan: light sentence
	action: reduce standard sentence

Decision: 3 months (suspended)

When a (decided) case is entered into the case-base, its XP is determined either by the justification always given for written precedents, or by the explanations supplied by the judge, who told the 'story'. For a new case the facts and beliefs are supplied by the judge about to pass sentence in the case. He also supplies the purpose, but would presumably want to experiment interactively with different criminological approaches. The plan-slot is then filled out on a temporary basis: mitigation or aggravation. Only the action is left to be decided.

This approach is somewhat naive. It appears that a single XP cannot represent all the different facets and intricacies of a case, and we shall see in the next section that the retrieval is actually carried out according to a more detailed structure than the XP.

3.3 Judicial MXPs.

The concept of an XP appears to be insufficient for the kind of explanations we aim at creating. It cannot cope with the detail and complexity of most legal cases, as illustrated by the following (true) 'story':

A young woman met a young man one evening in Tel-Aviv, and they decided to have fun together. After some dancing in a nightclub they ended up in a hotel room, where they spent the rest of the night in activities, which apparently were mutually enjoyable. The next evening they met again, and after some preliminary dancing they went down to the beach. Despite the girl's protests the boy repeated the performance of the previous night, with the result that the girl accused him of rape the next morning. When the girl told her story in court, the judge asked her why she complained to the police after having agreed to sleep with the boy the first night. "Why that is obvious", said the girl, "I do not mind sleeping with him in a fancy hotel, but not on the beach!". The boy eventually got off with a light prison sentence: 4 months.

Obviously there are many elements in this kind of story, and we have therefore constructed the following multi-structure, called an MXP, which is made up of several XPs. We shall first show it for the above story.

MXP: 'Not on the beach'

MOP:

accused according to 345
found guilty
standard sentence: 6 years
maximal sentence: 16 years

XP-1:

facts: first offence
beliefs: not dangerous to public
purpose: punishment, prevention,
should be given a second chance
plan: extreme mitigation
action: reduce standard sentence

XP-2:

facts: confessed
beliefs: seems trustworthy
purpose: retribution, no overload of prisons
plan: mitigation
action: reduce standard sentence

XP-3:

facts: victim agreed on previous occasion
beliefs: not as serious as standard rape-situation
not dangerous to public
purpose: retribution
plan: extreme mitigation
action: reduce standard sentence

An MXP consists of three parts: (i) A MOP, which contains general information about the story described by the MXP. (ii) A set of XPs. Each of these XPs describes one aspect of the given case relative to one of the indices found in the story. It has the layout shown above, with the index, given in the facts-slot. (iii) A quantitative conclusion from the individual XPs, i.e., the actual sentence (not shown above).

The case-base actually stores MXPs (and not XPs) for each member (case). The sentence in an old case is derived by the judge through consideration of the individual XPs of its MXP. Obviously this is not done using some kind of mathematical formula, so while the sentence is known and supplied in (iii), its derivation is only *indicated* in the plan- and action-slots.

An MXP is thus a structure, which gives an interpretation and explanation of a 'story' (legal case) according to the relevant legal aspects. Each such aspect is represented by an XP in part (ii), and general knowledge of importance is stored in parts (i) and (iii). The MXP does not represent the story from a narrative point of view - e.g., there is no time sequence of events. It represents the story as seen from the various legal viewpoints.

Based on the facts and beliefs of the new case it is possible to construct its MXP. The user of the system must supply the value of the purpose-slot, and the plan-slots are automatically filled according to the index of the individual XP, depending whether it is a mitigating or aggravating feature.

The retrieval is then of all MXPs which have any index in common with the new case. These MXPs may now be arranged in a lattice, according to the number of indices common with the indices of the new case. This is similar to the so-called claim-lattice of HYPO (see [Ashley90], p.40-42).

At this stage the XPs belonging to the 'most-on-point' MXPs are selected. The 'most-on-point' MXPs are those MXPs in the lattice that have greatest overlap of common indices with the new case.

One could conceivably argue that those XPs belonging to retrieved MXPs, which do not have an index in common with the case at hand, ought to be discarded. However, those XPs undoubtly contribute and influence the final outcome (sentence) of the case where they appear. We therefore use this dissimilarity to impose a fine structure on the 'most-on-point' MXPs. The more non-relevant XPs a given MXP has, the less relevant it is judged to be to the given case (see [Tversky77]).

We may thus finally define the set of 'most-most-on-point' MXPs as those with the largest number of indices common with the new case, and the smallest number of indices different from the indices of the new case. These MXPs are retrieved, and support the judge in his decision in the new case.

A qualitative approach to weighing of the MXPs and the individual XPs, including a cut-off threshold for controlling the number of retrieved MXPs is under development, and will not be considered here.

4. Summary.

The problems we have discussed in this paper relates to quantitative use of 'stories' .Our concern has been to choose and adapt an appropriate knowledge representation and retrieval method. We decided to adapt and generalize a knowledge representation structure: XP, which is convenient for giving explanations for stories. In our case these explanations are not related to episodic events, but to the deliberations and decisions of the judiciary.

The reason we adopted this approach is our wish to build a decision support system for sentencing. Such a system should not propose just a single sentence, but supply several possible ways of passing sentence in such a way that the justification of the sentence is evident. Our solution uses the MXP knowledge-structure discussed in the previous paragraph.

5. References.

[Ashley90] Ashley K.D. Modeling Legal Argument The MIT Press, Cambridge, Mass, 1990.

[Bain89] Bain W.M. JUDGE in: Riesbeck C.K., Schank R.C Inside Case-Based Reasoning Lawrence Erlbaum Assoc., Hillsdale, NJ, 1989, p.93-163.

[Dyer83] Dyer M.G. In-Depth Understanding MIT Press, Cambridge, Mass., 1983

[Kass86a] Kass A.M., Leake D.B., Owens C.C. Swale: A Program that Explains in: Schank R.C. Explanation Patterns: Understanding Mechanically and Creatively Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986, p.232-254.

[Kass86b] Kass A.M. Modifying Explanations to Understand Stories Proc.Eighth Annual Conference of the Cognitive Science Society Amherst, MA, 1986.

[Kass89] Kass A.M. Adaption-Based Explanation: Extending Script/Frame Theory to handle Novel Input IJCAI-89, 1989, p.141-147.

[Kolodner81] Kolodner J.L Organization and Retrieval in a Conceptual Memory for Events IJCAI-81, 1981, p.227-233.

[Kolodner83] Kolodner J.L. Maintaining Organization in a Conceptual Memory for Events Cognitive Science, 7, 1983, p.281-328.

[Kolodner85] Kolodner J.L., Simpson R.L., Sycara-Cyranski K. A Computer Model of Case-Based Reasoning in Problem Solving IJCAI-85, 1985, p.284-290.

[Leake92] Leake D. Evaluating Explanations: A Content Theory Lawrence Erlbaum Assoc., Hillsdale, NJ, 1992.

[Lehnert83] Lehnert W., Dyer M.G., Johnson P., Yang C., Harley S. BORIS - An Experiment in In-Depth Understanding of Narratives Artificial Intelligence, 20, 1983, p.15-62.

[Schank77] Schank R.C., Abelson R.P. Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures Lawrence Erlbaum Assoc., Hillsdale, NJ, 1977.

[Schank80] Schank R.C. Language and Memory Cognitive Science, 4 (3), 1980, p.243-284.

[Schank82] Schank R.C. Dynamic Memory Cambridge University Press, 1982.

[Schank89] Schank R,C., Leake D. Creativity and Learning in a Case-Based Explainer Artificial Intelligence, 40, 1989.

[Schank90] Schank R.C. Tell me a Story - A New Look at Real and Artificial Memory Charles Scribner's Sons, Macmillan Publ. Co., New York, 1990.

[Schubert89] Schubert L.K., Hwang C.E. An Episodic Knowledge Representation for Narrative Texts First Int. Conf. on Principles of Knowledge Rep. and Reasoning Toronto, Canada, 1989, p.444-458.

[Tversky77] Tversky A. Features of Similarity Psychological Review, 84, 4, 1977, p.327-352.

Making Case-Based Tutoring More Effective

Thomas J. Schult & Peter Reimann
Dept. of Psychology, University of Freiburg, 79085 Freiburg,
e-mail: schult@psychologie.uni-freiburg.de

Abstract: Starting from cognitive psychology findings concerning interindividual differences in learning from cases, we suggest strategies to optimize the learners' case processing capabilities by an appropriate design of the tutoring system. We introduce the systems CABAT (giving remindings in simulation environments), AXE (modeling effective case processing strategies) and CACHET (teaching effective processing strategies for cases of different origin).

It is well known that the acquisition of a problem solving skill (e.g., learning to program) or the understanding of an abstract principle (e.g., the concept of force in physics) is at least in its initial stages highly dependent on information about concrete, illustrative examples. These specific examples help learners to perform their first steps in using new procedures and applying new principles. Thus, they can be the basis for the development of more abstracted and generalized representations of skill and principles.

Not surprisingly, then, teaching strategies involving cases have been an area of active research, resulting in a number of promising tutoring systems covering domains such as instructional planning (Kolodner, 1991; Du & McCalla, 1991), natural sciences (Murray et al. 1990; Edelson 1991), business (Ferguson et al., 1991), and many more.

However, it was observed that just solving tutorial cases does not necessarily foster competence (Gräsel, Prenzel & Mandl, 1993). Crucial for successful later retrieval is a rich mental indexing structure of the cases that counteracts against the consequences of biasing and failures of retention. The more the learner is able to connect a new case to existing knowledge, the more he profits for later problem solving. In instructional settings, maximizing this connection should be supported by a tutorial module. We attempt to give this support in two ways that are explained in the following: one is to remind the learner of a previous case that is similar to the current one, and the other is to teach him effective case processing strategies.

Our suggestions are based on empirical work concerning interindividual differences in learning from examples. For instance, Chi, Bassok, Lewis, Reimann, and Glaser (1989) analyzed how students acquire problem solving knowledge concerning mechanics by studying worked-out examples. The study revealed important differences between successful and less successful students, success measured in terms of correct solutions to problems. Successful students mentioned more often that they didn't understand a certain part of the worked-out example. Besides this difference in monitoring understanding of the example text, successful students also engaged in a series of activities to overcome their problems: They elaborated on the relations between a particular step in the example and the goals behind that step. They further attempted to come up with a specification of conditions that could explain why the operator under question was applied. Finally, they elaborated on the effects the application of an operator had beyond those mentioned in the example. The less successful students displayed either none or considerably less of these elaborative inferences.

This so-called "self-explanation effect" has been reconstructed several times both in physics domains (VanLehn, Jones & Chi, 1992, Reimann, Wichmann & Schult, 1993) and programming (Pirolli & Recker, 1991). Besides its psychological relevance, we want to stress the importance of these empirical findings for research on intelligent tutoring systems. Building on these studies and the respective cognitive models, we try to foster the effectiveness of case-based teaching in two respects: Using remindings instead of predefined cases, and teaching case elaboration strategies.

1. Remindings in Simulation Environments

Simulation based learning environments have become an established tutoring system architecture (see de Jong, 1991, for an overview). Communicating with reactive systems of this kind leaves the control of the interaction to the learner, which is a part of the "discovery learning" philosophy usually underlying these systems. Often it was mentioned that this form of teaching requires some guidance to prevent the student from

getting lost in the space of possible situations (Elsom-Cook, 1990, Bredeweg & Winkels, 1991). Helping the student in organizing the interaction is one way to supply this guidance, but it is seldom provided. This help can also optimize the connections between cases and background knowledge, a requirement for effective indexing, as mentioned above.

Simulation environments (and tutoring systems in general) provide the means to work on a single problem at a time, but mostly they do not support comparing two or more problems. If they do, they provide help in analyzing a series of events, e.g. with a spreadsheet-like tool for recording simulation states and cognitive activities of the learner (Reimann, 1992). Still it is up to student to decide which events to consider and how to perform the comparison. This task can be supported by a case-based teaching component which serves as an external memory assistant for the student.

How can this support be given? Up to now, case-based tutors always use predefined cases for teaching. But in simulation environments, specifying cases in advance may not fit to the discovery learning philosophy. Presenting cases without restricting the student unnecessarily can be achieved by using the interaction itself as a case repository that is exploited by a memory assistant that reminds the learner of previous problems similar to the current one. This approach allows the learner to interact with the simulation environment as usual, but provides the opportunity for her to transfer from a prior to the current situation and to generalize across the two cases. Furthermore, we suppose that remindings activate elaborations produced when the reminded event first took place, and that the current case can be elaborated more thoroughly by relating it with the problem solving episode the system reminds of. In the light of the findings of the Chi et al. (1989) study, there should be more effective learning from the cases dealt with.

The system CABAT (Schult, 1993) puts forward a method of generating such remindings. CABAT was used as a component of a microworld learning environment in which students can design and run simulated experiments concerning elastic impacts, and it seems to be applicable to other domains that are formula-based. The central idea of CABAT is to store all experiments performed by the learner as cases. This episodic knowledge is put to use whenever the student encounters a situation that is similar to a previous one: Then the system reminds her of this previous case and explains the particular kind of similarity, so that at least parts of the prior solution can be transferred and adapted to the present case. In order to define similarity appropriately for this task, CABAT integrates an algebraic analysis of the domain formulas with a domain-independent theory of

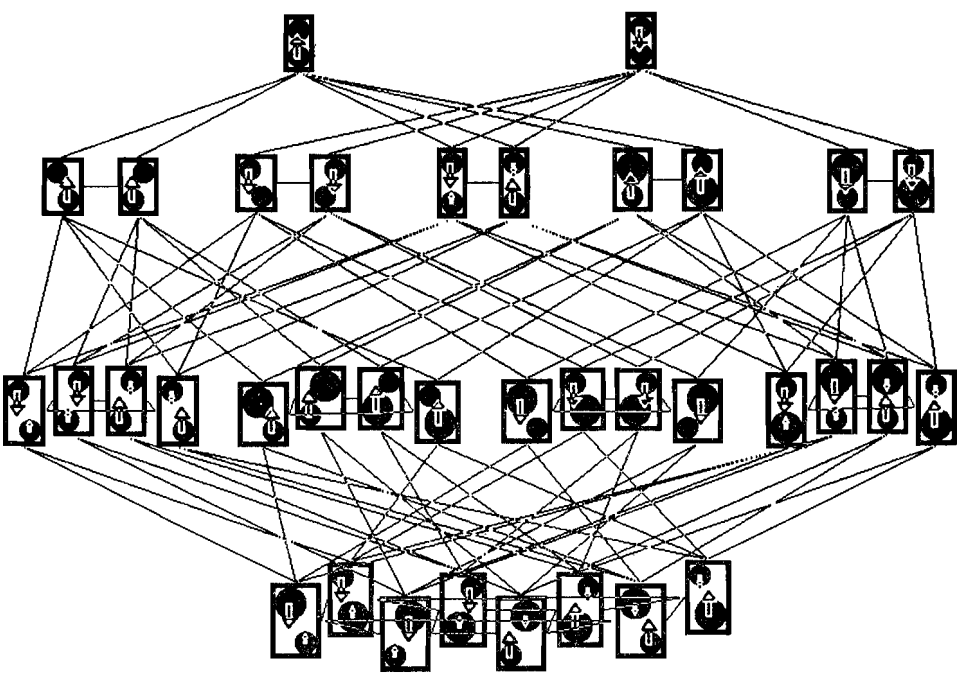


Fig. 1. A part of the graphical representation of a concept of similarity gained by CABAT in the domain of elastic impacts of disks

different types of similarity. This theory classifies cases by structural and superficial features, preferring the former.

2. Modeling Case Elaboration Strategies

The long term goal of the AXE project ("Active Example Elaborations", Reimann & Schult, 1991; Reimann, Schult & Wichmann, i. pr.,) is to develop a computer program that helps students with learning from cases that have the form of worked-out examples. The program will receive the same example as a human student and will try to comprehend it, aided by domain-specific knowledge and an explicit strategy for example elaboration. The process and the outcome of the example understanding attempts should form the basis for an interaction between the machine and the human student, where the system tries to help the student in elaborating effectively. Both to generate output that is potentially meaningful for the student and to enable the program to follow the student through her elaboration activities, it needs information about example comprehension strategies as put to use by human learners. Therefore, a prerequisite for the above tutorial scenario are computational models of elaboration strategies and the knowledge they process. In the AXE architecture, we incorporated elaboration processes found in the subjects of the Chi et al. (1989) study when reading the mechanics examples, among others.

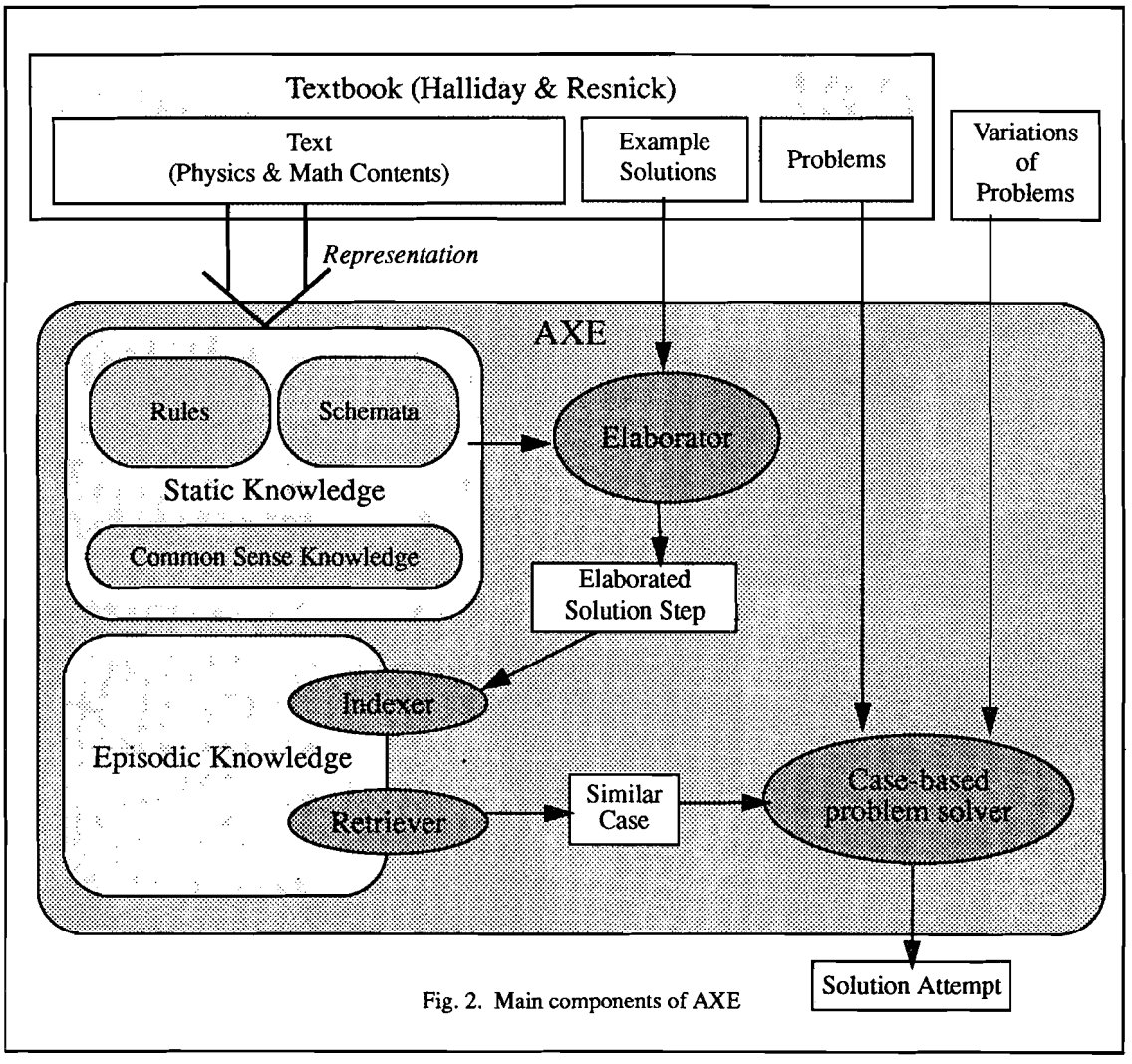


Fig. 2. Main components of AXE

AXE has two components: An example understander that takes as input a worked-out example text (not in natural language form) and produces as output an enriched case representation of the information contained in the example, and a problem solver that takes as input a problem description and produces a sequence of solution steps, relying mainly on the knowledge gained by the example understander (i.e., cases). This design allows us to

evaluate the quality of different learning methods by observing their impact on problem solving. Within an instructional application, this will allow us to demonstrate to students the relative merits of different learning from examples methods in terms of problem solving gain. Since AXE is geared towards modeling (and later supporting) the initial stages of knowledge acquisition from examples for learners with little domain specific knowledge, it is equipped with example elaboration strategies that can be employed by agents with varying degrees of domain knowledge. These elaboration strategies (basically, methods for solution enhancement and plan recognition) are not domain-dependent, but belong to the standard repertoire of normal adult learners. Depending on whether AXE is to be used for descriptive modeling (e.g., to capture characteristics of a poor learner) or for prescriptive purposes (e.g., to serve as an example learning model that should be imitated) it can be equipped with more advanced example study methods one finds only in more active human learners and readers. Running AXE in various configurations, we found clear relations between example study performance and later (case-based) problem solving.

3. An Integrated Case-Based Teaching Environment

Building on the insights gained from the AXE project, we are now using these results for instructional purposes: To demonstrate to students the value of knowledge-based elaborations of worked-out examples and to give them a first model of how to elaborate on these cases in an active, expectation-driven way. This case-based strategy tutor should help students in the initial phase of learning, a phase in which many students experience more frustration than insight.

Thus, we are concerned not just with communicating domain knowledge, but also with teaching effective case processing methods. Beside this focus on strategy training, we want to integrate cases of different origin: predefined cases (as in common case-based tutoring systems), reminded cases (as in CABAT) and on-line generated cases. As the student should not be overwhelmed with cases, this requires a selective dynamic scheduling process that determines in each situation an appropriate case to be presented (or none at all).

The domain chosen for the first prototype is chess endgames, an area which is usually taught by cases. Eventually, the system CACHET ("Case-Based Chess Endgame Tutor") will support the learner with various methods:

During the example study phase, it should

- Provide an interface so that student-generated elaborations can become part of the case representation
- Point out (remind) similiarities between examples (parts of examples)
- Demonstrate a good example processing strategy and the elaboration methods involved in a way that can be understood/copied by human learners
- Give direct advice on example processing procedures

During (case-based) problems solving, it should

- Remind the learner of similar examples or problems solved already (problems the student solved successfully become example cases, unsuccessful solutions can serve as counterexamples; both are treated as cases);
- Provide additional examples that were not produced by the student
- Support case modification
- Give hints and advice.

Our first goal with CACHET is to implement a tutor that supports all phases of case-based reasoning and learning with worked-out examples: encoding of examples, indexing, retrieval, modification, and learning (of new indices). The first phase (encoding) is an issue often ignored in research on case-based reasoning where the assumptions is that cases "are there", whereas we stress that cases need to be produced. The second goal is to combine several instructional strategies, ranging from non-directive forms (such as pointing out a similarity or difference to the student without further comments) over semi-directive (modeling good behavior) to rather directive forms (giving concrete advice). How and when these strategies are to be used depends on the general pedagogical strategy (e.g., stressing exploration vs. stressing immediate feedback) and on characteristics of the student, both general ones (such as domain knowledge already acquired or a preference to first work on ones

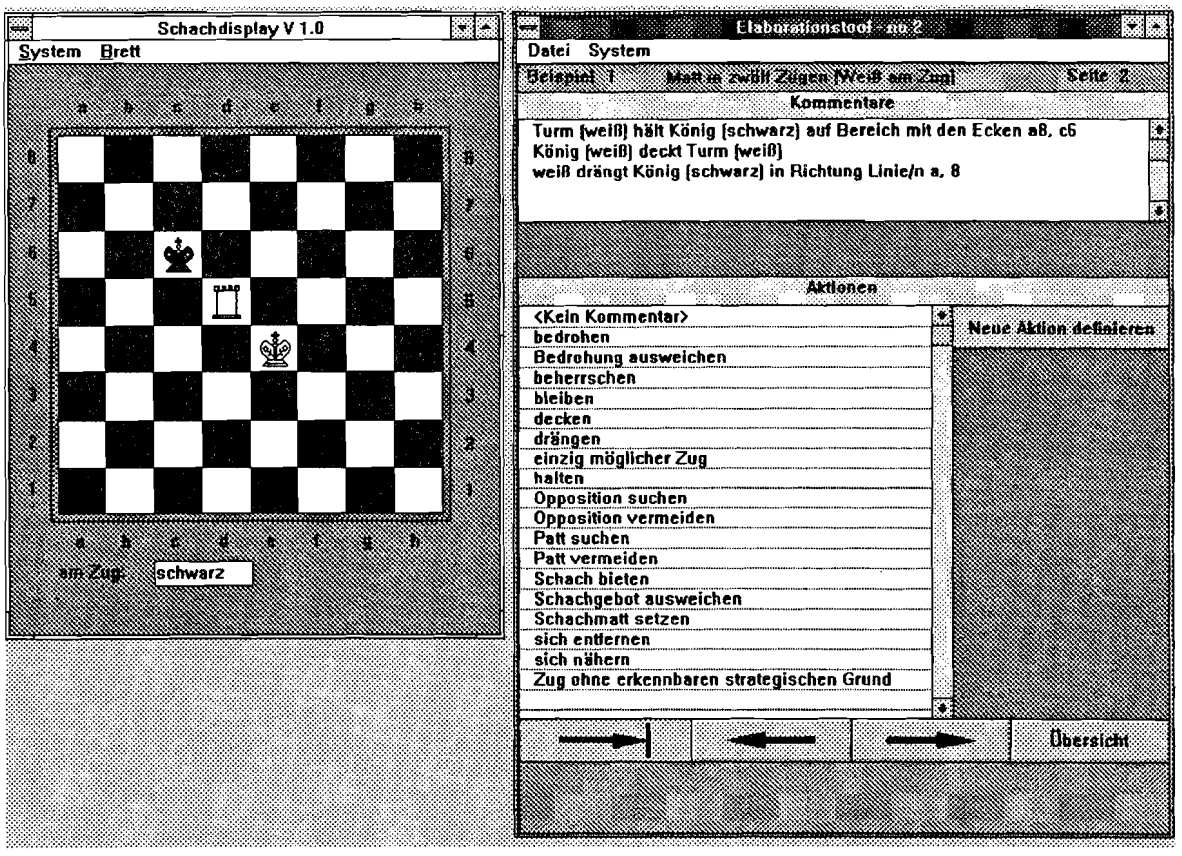


Fig. 3. Enriching a presented case step by elaborating it

own) and specific ones (e.g., the state of knowledge after three examples). We intend to use methods of opportunistic planning to account for the various factors that will influence the details of the tutorial strategy.

A prerequisite for all these forms of tutorial interaction is that the tutorial program and the student have a shared language to denote objects, relations and phenomena in the domain under study. In our case, the tutorial discourse pertains to two levels: the object level examples and problem solutions (together: cases) and a meta-level, i. e., methods of elaborating example solutions and using them for problem solving.

Over the last months, we have been developing a first version of an interface that provides student and system with a common language for the object level moves and reasons for moves in simple chess endgames. One should note that such an interface already constitutes an instructional manipulation: The student is provided with a case description language, and using it to describe chess endgame positions, moves and plans may already result in a deeper processing of examples (and counter-examples).

The case description language was developed mainly by an analysis of available textbooks on endgames. At each step of the presented example, the learner is offered a set of domain specific elaborations ranging from paraphrazing comments as "check" to more higher level elaborations as "avoid opposition". Currently we are evaluating the effects of such a elaboration environment on learning. The next step will be to add a retrieval component and a case memory indexed by the elaborations, so that elaborating is not only a prerequisite for the learner's remembering of important situations, but also enables the system to remind the learner, if appropriate.

To summarize: Psychological research provides us with an increasing number of observations that show how humans make use of specific instances and cases, both for problem solving and learning. In order to develop forms of computer-based instruction that take these findings into account, research on case-based reasoning seems to provide the right sort of modeling techniques. We illustrated with three examples how these techniques can be put to use for instructional purposes.

References:

Bredeweg, B. & Winkels, R. (1991). Teaching according to GARP. In L. Birnbaum (Ed.), *The international conference on the learning sciences.* Charlottesville, VA: AACE.

Chi, M.T.H., Bassok, M., Lewis, M., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science, 13, 145-182.

de Jong, T. (Ed.) (1991). Computer simulations in an instructional context. *Education and Computing* 6. Amsterdam: Elsevier.

Du, Z. & McCalla, G. (1991). CBMIP - A case-based mathematics instructional planner. In L. Birnbaum (Ed.), *The international conference on the learning sciences.* Charlottesville, VA: AACE.

Edelson, D.C. (1991). Why do cheetahs run fast? Responsive questioning in a case-based teaching system. In L. Birnbaum (Ed.), *The international conference on the learning sciences.* Charlottesville, VA: AACE.

Elsom-Cook (Ed.) (1990). *Guided discovery tutoring.* London: Paul Chapman.

Ferguson, W., Bareiss, R., Osgood, R. & Birnbaum, L. (1991). ASK systems: An approach to story-based teaching. In L. Birnbaum (Ed.), *The international conference on the learning sciences.* Charlottesville, VA: AACE.

Gräsel, C., Prenzel, M. & Mandl, H. (1993). Konstruktionsprozesse beim Bearbeiten eines fallbasierten Computerlernprogramms (Research Reports, No. 13). München: Ludwig-Maximilians-Universität, Institut für Empirische Pädagogik und Pädagogische Psychologie.

Kolodner, J. (1991). Helping teachers teach science better: Case-based decision aiding for science education. In L. Birnbaum (Ed.), *The international conference on the learning sciences.* Charlottesville, VA: AACE.

Murray, T., Schultz, K., Brown, D. & Clement, J. (1990). An analogy-based computer tutor for remediating physics misconceptions. *Interactive Learning Environments* 1, 2, 79-101.

Pirolli, P., & Recker, M. (1991). The role of examples, self-explanation, practice, and reflection. Berkeley: University of California at Berkeley.

Reimann, P. (1992). Elicitating hypothesis-driven learning in a computer-based discovery environment. In A. Tiberghien & H. Mandl (Eds.), Knowledge acquisition in the domain of physics and intelligent learning environments. Berlin: Springer.

Reimann, P. & Schult, T.J. (1991). Modeling example elaboration strategies. In L. Birnbaum (Hrsg.), *The International Conference on the Learning Sciences.* Charlottesville, VA: Assoc. for the Advancement of Computing in Education.

Reimann, P., Schult, T.J. & Wichmann, S. (i. pr.), Understanding and Using Worked-Out Examples. In G. Strube & K.F. Wender (Hrsg.), The cognitive psychology of knowledge. Amsterdam: Elsevier.

Reimann, P., Wichmann, S. & Schult, T. J. (1993). A learning strategy model for worked-out examples. In Proceedings of the World Conference on Artificial Intelligence in Education 1993. Charlottesville, VA: Assoc. for the Advancement of Computing in Education.

Schult, T.J. (1993), Tutorial remindings in a physics simulation environment. In Proceedings of the World Conference on Artificial Intelligence in Education 1993. Charlottesville, VA: Assoc. for the Advancement of Computing in Education.

VanLehn, K., Jones, R.M., & Chi, M.T.H. (1992). A model of the self-explanation effect. Journal of the Learning Sciences, 2, 1-59.

ELM: Case-based Diagnosis of Program Code in a Knowledge-based Help System

Gerhard Weber

FB I - Psychology, University of Trier
54286 Trier, FRG

Abstract. ELM is a case-based learning system that interprets, stores, and reuses solutions to programming tasks. Information from an episodic learner model can be used to reduce the effort creating explanations of how and why a solution to a programming problem produced by a programmer is buggy or suboptimal. Program recognition adapted to a particular programmer can be used by tutoring and help systems to individualize help and to improve tutorial activities adapted to the users needs.

1 Introduction

In intelligent tutoring systems or knowledge-based help systems, simulating how novices understand and code programs can be useful to build a valid, cognitive student model. Up to now, approaches to program recognition and program debugging based on cognitive models (e.g., the CMU-LISP-tutor [1]) do not adapt to the programmer, that is, they do not learn how a particular programmer typically solves programming problems. Information resulting from program diagnosis can be used in many systems to build a student model, but information from such a model is used only for tutorial and remedial purposes and not for triggering or improving the diagnostic process itself. So, each diagnosis begins from the scratch.

From a CBR point of view, this seems not to be optimal. Programmers use examples and previous problem solutions when solving new programming problems. They search for solutions to analog problems and alter existing code [11, 13]. Therefore, results from previous analyses of program code demonstrated in examples or coded by the same programmer can effectively be reused when diagnosing the current program code. Considering these findings, we have developed the ELM system [11]. Similar to the model-tracing approach in the CMU-LISP-tutor [2], ELM tries to automatically generate the same code the novice programmer has produced as a solution to a problem solving task. When generating the code, the system can reuse solutions to subgoals and corresponding plans stored in the episodic learner model (ELM) according to case-based reasoning.

2 Description of ELM

The CBR-approach employed in ELM differs from many other CBR systems in some important aspects. ELM relies on a rule-based problem solving system being able to analyze program code by its own without considering pre-stored cases. As it is a case-based learning system, ELM learns about an individual learner from creating cases from explanation structures that result from analyzing examples and problem solutions. With a growing case-base, information from these cases can be used to shorten the problem solving process of the diagnostic component by reusing previous solutions and by avoiding dead ends during search. As the system does not start with pre-stored cases, the rule-based diagnostic process will be described first. Second, we will introduce and demonstrate how cases are created, stored, indexed, and used.

2.1 The Diagnostic Process

Diagnosing program code in ELM works as follows. Novices programming in the ELM-programming environment [12] code function definitions in a structured LISP-editor [5], so their function code is at least syntactically correct. The cognitive analysis of the program code employs an explanation-based generalization (EBG) method [7]. It starts with a task description related to higher concepts (general and LISP-specific programming concepts, plans, and schemata) in the knowledge base. Every concept comprises plan transformations and rules describing different ways to solve the goal given by the current plan. Applying a rule results in comparing the plan description to the corresponding part of the student's code. In the plan description, further concepts may be addressed. The cognitive diagnosis is called recursively. It terminates when a function name, a parameter, or a constant are matched. The cognitive diagnosis results in a derivation tree built from all

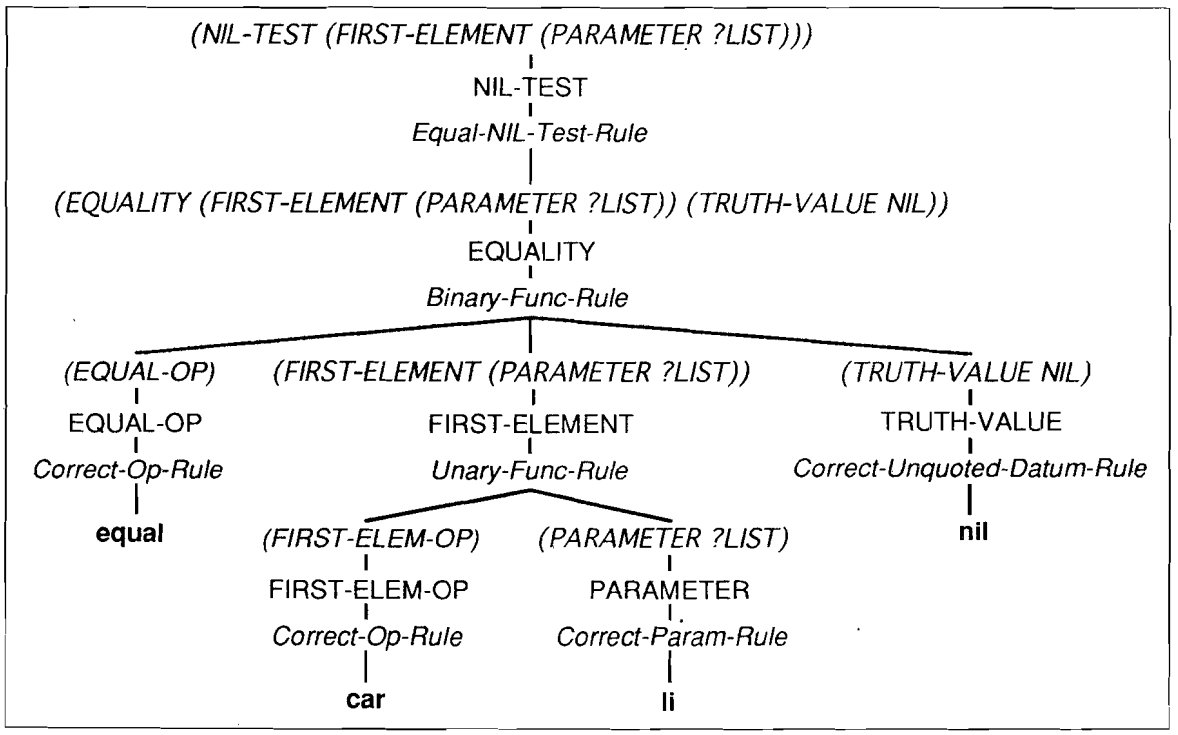


Fig. 1. Partial derivation tree explaining the code `(equal (car li) nil)` for the plan *(NIL-TEST (FIRST-ELEMENT (PARAMETER ?LIST)))*.
Legend: *ITALIC-CAPITALS*: plans, CAPITALS: concepts, *Italics*: rules, **bold**: LISP code.

concepts and rules identified to explain the student's solution. This derivation tree is an explanation structure in the sense of EBG.

The interpretation of a derivation tree can be best demonstrated by an example. Let us assume that a programmer had to code a function definition containing a case decision. In one of the cases of this case decision he or she had to test whether the first element of a list that is bound to a local variable from the parameter list of the function definition has the truth-value NIL. The programmer coded `(equal (car li) nil)` as a solution to the plan *(NIL-TEST (FIRST-ELEMENT (PARAMETER ?LIST)))*. This plan addresses the concept NIL-TEST which indexes some rules describing how such a plan may be solved. As a best interpretation the *Equal-NIL-Test-Rule* was applied transforming the current plan into the equivalent plan *(EQUALITY (FIRST-ELEMENT (PARAMETER ?LIST)) (TRUTH-VALUE NIL))* testing for the truth-value NIL by directly comparing the first element of the list to the truth-value NIL. This plan addresses the programming concept EQUALITY that can be solved by coding an appropriate operator for the equality operation and then solving the subplans for both arguments of the equality operation. This procedure is called recursively and results in the derivation tree shown in Figure 1.

2.2 Creating, Indexing, and Using Cases

Concepts addressed in the derivation tree are the basis for creating episodic frames. These frames are integrated into the knowledge base as instances of their concepts. Therefore, cases are not stored and indexed as a whole. They are distributed regarding subplans used during problem solving. If an episodic frame is the first instance under a concept of the knowledge base, this single case is generalized from structural and semantic aspects in the data. This generalization mechanism is comparable to single-case generalization in EBG. Additionally, similarity-based generalization of data and plans can occur. With increasing knowledge about a particular learner, hierarchies of generalizations and instances are built under the concepts and rules of the knowledge base.

An example of small hierarchies after inserting frames from two cases into the knowledge base is shown in Fig. 2. Episodic instances and generalizations constitute the episodic learner model. As information about the learner is directly related to the expert-like domain knowledge, this learner model is a type of "overlay model" [3]. Information from episodic instances can be used in further diagnoses if the current part of the code matches a solution to a similar plan stored in the episodic learner model. Two different cases of matching can be distinguished. First, if the current plan including all nested subplans matches the plan stored with the episodic

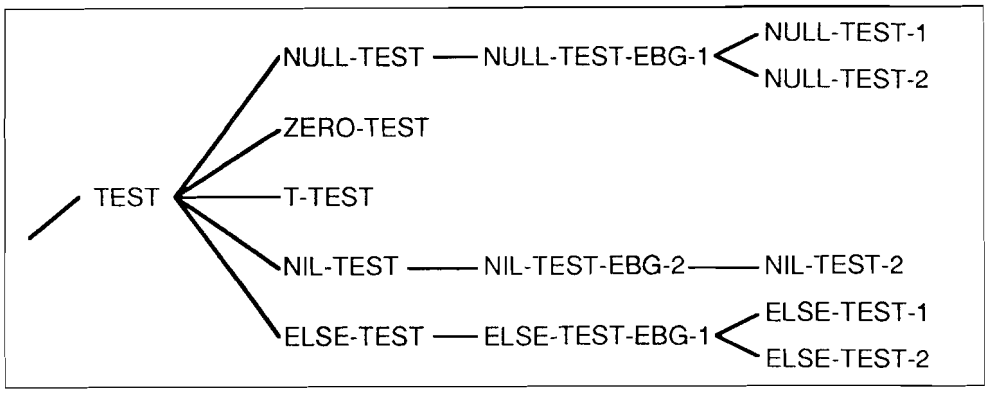


Fig. 2. Hierarchy of episodic instances and generalizations after inserting episodic frames for concepts from the derivation tree (Fig. 1) as a second episode into the knowledge base.

instance and if the currently considered part of the code exactly[1] matches the code stored with the episodic instance then the previous explanation can easily be reconstructed from episodic frames belonging to the same episode. This refers to directly reusing explanations from previous cases. Second, if subplans and/or code match only partially then the diagnostic process is triggered using information from the episodic frames. That is, rules that were successfully applied in previous cases are tried first. So, the CBR-method is used in the sense of 'shortcuts' [4] while constructing a new explanation.

2.3 Relation to Other Systems

ELM is, up to date, one of the few approaches to automated diagnosis in a tutoring or help system employing a CBR method. Another CBR approach to diagnosis of program code is realized in the newest version of SCENT [6], but it differs from ELM in several aspects. In SCENT, pre-analyzed cases are stored as a whole regarding a static granularity hierarchy that expresses aggregation and abstraction dimensions. These cases are used during analysis of the student's code to give detailed advice. In ELM, only examples from the course materials are pre-analyzed and the resulting explanation structures are stored in the individual case base of the learner model. Elements from the explanation structures are stored as instances of their corresponding concepts from the domain knowledge base, so cases are distributed in the form of instances of concepts. Generalization hierarchies of instances are built up from explanations of the program code that a student produced to solve programming problems. Therefore, generalization hierarchies reflect the process of knowledge acquisition for a particular student.

3 Evaluation Studies

3.1 Reduction of Computational Effort

In a first study, we analyzed interaction protocols obtained from 11 subjects while working on exercises in ELM-PE [12] during the first six lessons of our introductory programming course. In the first six lessons, these students worked on 36 - 40 different programming tasks producing and evaluating among 46 and 77 function definitions. The more function definitions students evaluated and tested to solve a problem the more errors they made during programming. Altogether 697 cases were observed and analyzed by the cognitive diagnosis. The median case required 76 rules to be tested when no episodic information was used. This number reduced to a median of 29 rules tested per case when episodic information was used. The same results can be looked at in a different way. In 91% of all cases (632 out of 697) applying episodic information reduced computational effort. In only 11 cases (1.6 %), taking episodic information into account resulted in additional computations.

3.2 Predicting Individual Problem Solutions

In a further simulation study, we automatically predicted the program code that 20 novice programmers produced solving recursive programming problems during three lessons about recursion of our introductory programming course. This was done with and without considering information from the individual case base of ELM. For every new task, we compared the first complete function definition the programmer coded to the

[1] During matching names of local variables and recursive function calls are unified.

Tab. 1. Number and percentage of different predictions for code with and without considering individual cases for 302 recursive function definitions produced by 20 Subjects for up to 17 different programming tasks.

Cases	Number of Predicted Solutions	
	Case-based	Canonical
Exact prediction	127 (42.1%)	87 (28.8%)
Exact prediction except exchanging arguments in commutative functions	23 (7.6%)	19 (6.3%)
Prediction correct except for a coding error	68 (22.5%)	64 (21.2%)
Bad or incorrect prediction	84 (27.8%)	132 (43.7%)

function code generated by the diagnostic component. Considering episodic information from the case base 127 out of 302 cases (42.1%) were exactly predicted (Column 'Case-based' in Tab. 1). That is, the predicted code totally matched the observed code, except for names of local variables and self-defined functions. In 11 of these cases, even an error was predicted that was observed from the programmer in a previous task. Predicting the expected code for a new task without considering individual cases (Column 'Canonical' in Tab. 1) resulted in 87 cases (28.8%) where the code was predicted exactly. In 23 vs. 19 of all cases, the code was exactly predicted except for exchanging arguments in commutative functions. In 68 vs. 64 of all cases, the code was predicted in the correct direction, that is, Subjects produced an error that was not predicted by the system. However, it was the same algorithm and the rest of the code did match. In 84 (27.8%) vs. 132 (43.7%) of all cases the system's prediction was bad or even completely false.

One may wonder why there are so many cases where the code produced by the programmer could be predicted correctly by the system, whether considering episodic information or not. This result can be put down to the fact that programming novices often try to reuse code from examples and from previous solutions (Weber & Bögelsack, in press). Therefore, algorithms and code used to solve similar programming problems do not differ so much from expected, canonical solutions. The chance to correctly predict code drastically decreases when programmers have experiences in other programming languages and use different programming concepts and different algorithms. To predict such code correctly their experiences with other programming languages must have been included into the knowledge base by formulating appropriate programming concepts and rules and creating cases from their prior experiences. These experiences are not known to the system, so it will often fail to predict their solutions.

In 200 of all 302 cases (66.2%), predictions with and without considering cases did not differ. This number may appear very high, but one must consider, that only for a small number of examples from the materials cases were stored in the case base. All other cases were built up individually when analyzing code produced by the programmer. So, there is only a chance for episodic information to trigger the diagnosis or the automatic generation of code if deviations from the expected, canonical way of programming were observed in previous cases and could be applied in the current case. In 102 of all predictions this case happened. In 74 of these 102 cases (72.5%), considering information from the case base resulted in a good prediction that was better than the canonical prediction. In 15 cases (14.7%) the canonical prediction (without considering individual cases) made a better prediction, and in 13 cases (12.7%) both simulation types failed to make a good prediction.

3.3 Comparing ELM to Analogy-based Systems

Another advantage of an individualized, episodic learner model stems from its potential to find analogies and remindings to examples from the learning materials and to solutions from previous programming episodes. For this purpose, we have developed a fast, explanation-based retrieval algorithm (EBR, [9]). We compared our EBR-method to the ARCS-model [8]. Our simulation showed that in most cases the EBR method retrieved analogs as well as the ARCS-method and in some cases the EBR-method outperformed the ARCS-method [9].

In a related experiment, we compared finding analogies by the EBR-method and the ARCS-model with similarity ratings judged by programming novices and by advanced programmers [10]. In this experiment, too, retrieving analogs by the EBR-method showed a slightly higher concordance with subjects' ratings than results from simulations with the ARCS-method. For both models, however, results from simulations were in higher concordance with ratings from the "Advanced" group compared to "Novices'" ratings. These findings are typical for user or learner models that are based on overlay models as they reflect knowledge about a learner from an

expert-like point of view. This may be an advantage in case of a tutoring system where appropriate examples and analogies must be found by the system guiding the learner to expert-like programming.

4 Conclusion

The CBR-approach in ELM offers several advantages over more traditional approaches to cognitive diagnosis and user modeling. First, considering information from similar cases during the diagnostic process can reduce the computational effort according to shortcuts [4]. Second, knowledge about a particular programmer being typical for him or her reflects his or her individual programming style. Such information can be utilized to predict the programming behavior and to show up analogies and remindings to examples and previous cases.

References

1. J. R. Anderson, F.G. Conrad, A.T. Corbett: Skill acquisition and the LISP tutor. Cognitive Science 13, 467-505 (1989)
2. J.R. Anderson, B.J. Reiser: The LISP tutor. Byte 10(4), 159-175 (1985)
3. B. Carr, I. Goldstein: Overlays: A theory of modelling for computer aided instruction (AI Memo 406). Cambridge, MA: Massachusetts Institute of Technology, AI Laboratory (1977)
4. K.J. Hammond (ed.): Proceedings of the Second Workshop on Case-Based Reasoning. San Mateo, CA: Morgan Kaufmann Publishers (1989)
5. A. Köhne, G. Weber: STRUEDI: A LISP-structure editor for novice programmers. In: H.J. Bullinger, B. Schackel (eds.): Human-Computer Interaction INTERACT '87. Amsterdam: North-Holland 1987, pp. 125-129
6. G. McCalla, J. Greer: Helping novices learn recursion: Giving granularity-based advice on strategies and providing support at the mental model level. In: E. Lemut (ed.): Cognitive models and intelligent environments for learning programming. Berlin: Springer-Verlag (in press)
7. T.M. Mitchell, R.M. Keller, S.T. Kedar-Cabelli: Explanation-based generalization: A unifying view. Machine Learning 1, 47-80 (1986)
8. P. Thagard, K.J. Holyoak, G. Nelson, D. Gochfeld: Analog retrieval by constraint satisfaction. Artificial Intelligence 46, 259-310 (1990)
9. G. Weber: Explanation-based retrieval in a case-based learning model. Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, Chicago, IL. Hillsdale, NJ: Cognitive Science Society 1991, pp. 522-527
10. G. Weber: Analogies in an intelligent programming environment for learning LISP. In: E. Lemut (ed.): Cognitive models and intelligent environments for learning programming. Berlin: Springer-Verlag (in press)
11. G. Weber, A. Bögelsack: Representation of programming episodes in the ELM model. In: K.F. Wender, F. Schmalhofer, H.D. Böcker (eds.): Cognition and computer programming. Norwood, NJ: Ablex Publishing Corporation (in press)
12. G. Weber, A. Möllenberg: ELM-programming environment: A tutoring system for LISP beginners. In: K.F. Wender, F. Schmalhofer, H.D. Böcker (eds.): Cognition and computer programming. Norwood, NJ: Ablex Publishing Corporation (in press)
13. G. Weber, G. Waloszek, K.F. Wender: The role of episodic memory in an intelligent tutoring system. In: J. Self (ed.): Artificial intelligence and human learning: Intelligent computer-aided instruction. London: Chapman & Hall. 1988, pp. 141-155

Chapter 9

Case-Based Image Processing

Image Retrieval without Recognition*

Carl-Helmut Coulon
German National Research Center for Computer Science
Schloss Birlinghoven 53757 Sankt Augustin

Abstract

This paper deals with the automatic retrieval of objects, only based on their 2-dimensional image. The resulting method should be used to support an architect using already designed supply nets for a new design. In order to be accepted by the architect the retrieval process must be very fast. Besides this strict requirement the automatic retrieval may suggest more than just the most similar supply nets, but also some silly ones. The different sections of this paper describe the solutions of the problems which occured during the development of the retrieval method.

1 Introduction

One complex subtask of the design of buildings is the design of supply nets. Figure 1 shows two different supply nets, which should be recognized as similar. The idea of the developed retrieval method is to simulate some kind of "Watching from distance". This is done by scaling the 2-dimensional images of the compared objects to different images of reduced resolutions. In the second step the images are turned to the same position before they are compared in the last step.

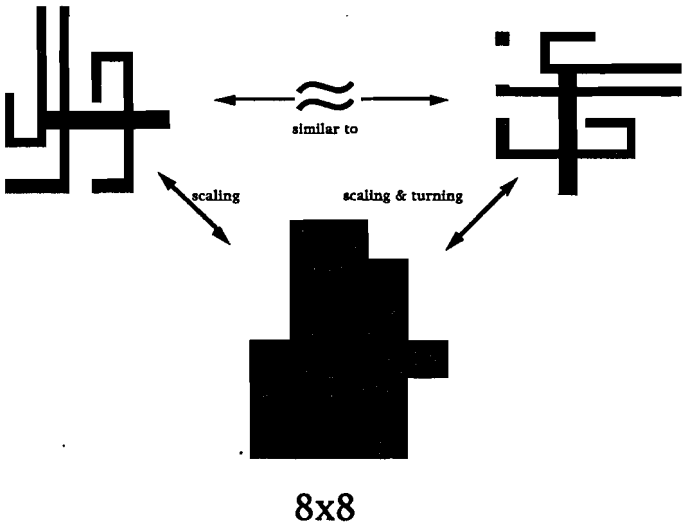


Figure 1: The Idea: Watching from distance

*This research was supported by the German Ministry for Research and Technology (BMFT) within the joint project FABEL under contract no. 413-4001-01IW104. Project partners in FABEL are German National Research Center of Computer Science (GMD), Sankt Augustin, BSR Consulting GmbH, München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

2 Scaling process

By reducing the resolution of the representations two advantages are reached. First, different objects have the same 2-dimensional image, depending on the chosen resolution. Second, the amount of information to compare is reduced extremely. In order to make the scaling process as fast as possible, it should be

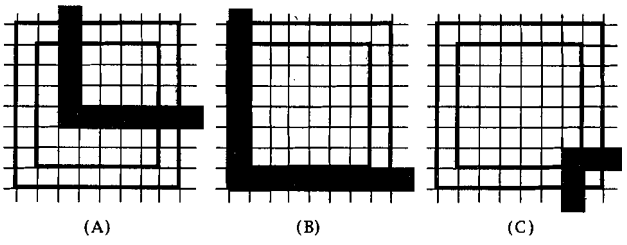


Figure 2: First Task: Scaling to an handable resolution

based on scaling routines of the chosen computer system. This leads to a tradeoff between speed and the usability of the resulting scaled image. The most efficient scaling method found so far is explained in figure 2:

Let the (A),(B) and (C) be pixel-maps of parts of a larger image. Each of them has to be scaled to one pixel. If one pixel inside the inner box of each part is black, the resulting pixel is black, too. The parameter to refine this method is the distance between the inner box and the border. As shown in pixel-maps (B) and (C) there are still results, which seem to be unwanted. Even so the results are acceptable (figure 4).

3 Normalizing the position

Supply nets should be still recognized as identical if the same structure appears after turning by multiples of 90 degrees or mirrored. In order to avoid a comparison for each of the eight possible transformations the scaled images are transformed to a "normalized" position. This position should be the same, even for similar, but not identical images. Also it must be definite. Figure 3 describes two possible methods. Method (A) weights each pixel-map by an matrix, who's highest values are at the left top. If two transformations are weighted the same, the one with the most top left black pixel is chosen. Method (B) reduces each pixel-map to a triangle and counts the black pixels reduced to the same position. This second method involves a loss of information, which increases the number of identical representations of different objects.

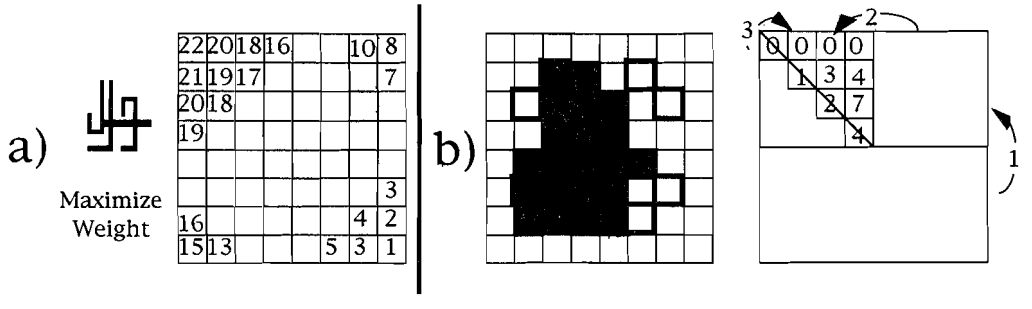


Figure 3: Second Task: Finding one normalized position

4 Organization of case-base

In order to reduce the number of images a new image has to be compared with, the case-base is organized like a tree. In figure 4 you see a part of the case-base. This part consists of the images belonging to five cases. At the level of a 12x12 resolution all images are already different. The two most left cases consist of slightly different sized supply pipes. Their 6x6 image is identical and their 7x7 and 8x8 images differ by one pixel only. A retrieval process starts at the root node and follows the connections until the number of possible similar cases is reduced to a previously defined limit.

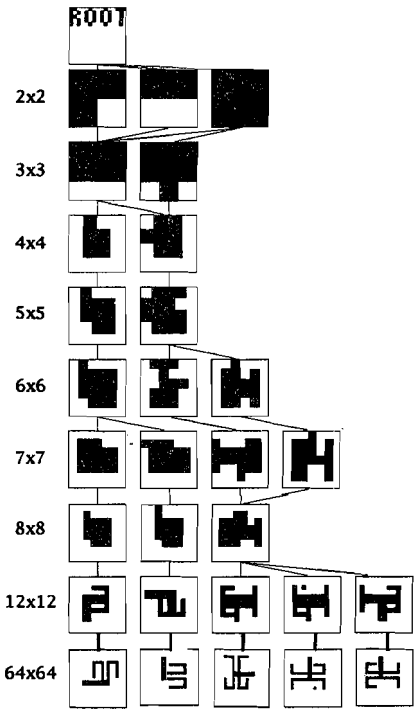


Figure 4: Third Task: Building the case tree

5 Comparison

Like the scaling process, the comparison process should be as fast as possible. Two pixel-maps can be easily compared by merging both together using the "xor" function and counting the black-pixels of the resulting pixel-map. This process is very fast. More complex comparison methods, like comparing also the surrounding of each pixel, did not lead to better results so far.

6 Possible extensions

Motivated from neurophysiological results we took up the idea to simulate one property of the human visual system. Focusing on one area of an image, the resolution decreases from this area to the border. Actually we try to simulate this by using a resolution like shown in figure 5. This stresses a problem we

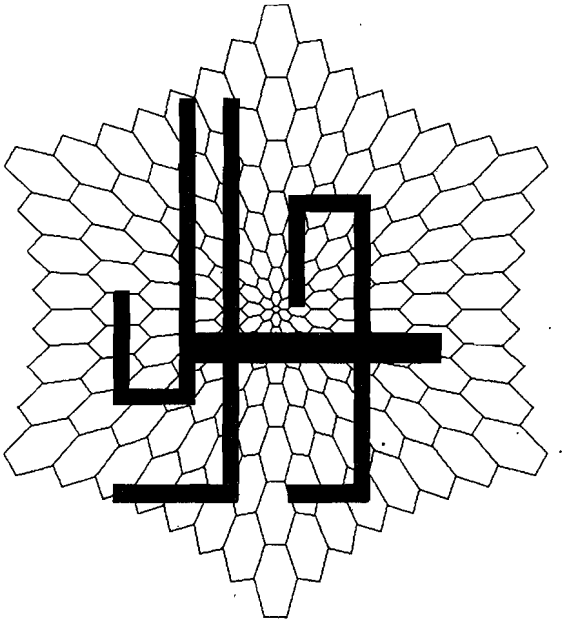


Figure 5: Extension: Thinking about the human visual system

did not elaborate so far. We will have do some more knowledge-acquisition, to find out which areas of an image are focussed by the architect, and which of the focused area lead to the recognition of similarity.

CASE-BASED REASONING FOR IMAGE INTERPRETATION IN NON-DESTRUCTIVE TESTING

PETRA PERNER
HWTK Leipzig, Department of Computer Science
Postfach 66, 04277 Leipzig, Germany
petra@informatik.th-leipzig.de

ABSTRACT. Image-based techniques are mostly used for the determination of defects in materials and components in nondestructive testing. In the former time, it has been not possible to come up with an sufficient approach for an automatic inspection system based on statistical or knowledge-based classifiers. The nature of the problem shows that case-based reasoning would be an appropriate method for an automatic inspection system in nondestructive testing. In the paper we discuss CBR for image interpretation based on an example of ultra sonic inspection. We describe the case representation, the problem of similarity over pictures and at least, we discuss the necessary features which a case-based system for image interpretation should have. The aim of the paper is not to describe a particular realization rather than to outline the tasks and problems which are related to case-based reasoning for image interpretation.

1 INTRODUCTION

Image-based techniques are mostly used for the determination of defects in materials and components in nondestructive testing. Usually this techniques do not give a unique image of the real defect image. Also, there does not exist a correlation formulized in a analytical or numerical model between the input image and the output image. Therefore, the interpretation of the output image is the task of an operator. Although, image-based techniques do not provide a good image of the defect an experienced operator is able to determine the type, the location and the size of the defect.
The operator uses for the interpretation of the image knowledge from past experiences, knowledge of the ultra sonic physics and knowledge which he acquired during the qualification of the image-based technique at test-bodies.

To aquire knowledge about the imaging behavior of the image-based technique by a particular test situation is a common procedure in nondestructive testing. This procedure is used where a model about the test physics did not provide sufficient results or for test techniques which are based on unknown physical effects. Under economical reasons, it is only possible to built one test-body. On the basis of the acquired knowledge it is possible to perform an interpretation based on analogical reasoning.

In case the costs seem to be reasonable then a defect catalogue is prepared for the particular test situation. Such catalogues do exist f.e. for the x-ray inspection of welding seams. The catalogues are used for operators training or they are taken by the interpretation of difficult cases. Then, the operator selects from the catalogue by the help of a nomenclatur one or more cases which are similar to the real case and determines by the following image comparison the case which is the closest one to the actual case.
Because of the broad variety of the defect gestalt and the technical cost for the preparation of the catalogue they do not contain for every possible defect a defect image. Therefore, we have to consider the catalogue as an inhomogenious solution space.

In the past, there has been a lot of effort to come up with a numerical classifier for x-ray image interpretation /1/ based on this catalogues. Also for ultra sonic image interpretation, a statistical classifier /2/ was built based on a class of trajectories derived from an numerical ultra sonic model. But the quality is not sufficient caused by the inhomgenous solution space.
A knowledge-based system for x-ray inspection is described in /3/. It performs in a cbr-like manner. Well known defects are stored as templates by attributes and attribute values (which are more general than a case) in the knowledge base. A decision is done by comparing the real defect with the templates. In case there is no match between a new situation and a template a confidence-factor based classification is carried out in the second phase.

The nature of the problem shows us that case-based reasoning can be an appropriate method for image interpretation in nondestructive testing.

It should be possible to come up with a classifier based on the catalogue and to design the classifier with the ability to learn from new test cases.

In the paper we want to discuss case-based reasoning for image interpretation based on an example for ultra sonic image inspection. First, we give an introduction to the test process. Then, we describe the case representation. The aim of the paper is not to describe a particular realization rather than to outline the tasks and problems which are related to case-based reasoning for image interpretation. This should give a guidance for other image related CBR tasks.

2 PHASES OF THE TEST PROCESS

The test process consists of the following phases:

Test Preparation	Planing&Design
Test Execution	Manipulation
Test Evaluation	Classification&Interpretation.

Every test process is started by test process preparation. A configuration of the image inspection system has to be done in accordance to the minimal defect size, the condition of the component to inspected and the expected defect type. That means for an ultra sonic system to determine the right sensor with the parameter appertur angle, frequency, mode type etc.. Also, it is to plan the path of the sensor over the component and it is to adjust the data acquisition unit.
An ultra sonic sensor are moved over a test-component either by hand or by an manipulator. This manipulator has to be controlled by an control unit.
The last phase of the test process is the test evaluation meaning the determination of the type , the location and the size of the defect.

A more detailed description of the test process is given in /4/. Now we want to look at an ultra sonic image and determine the type of knowledge in an image.

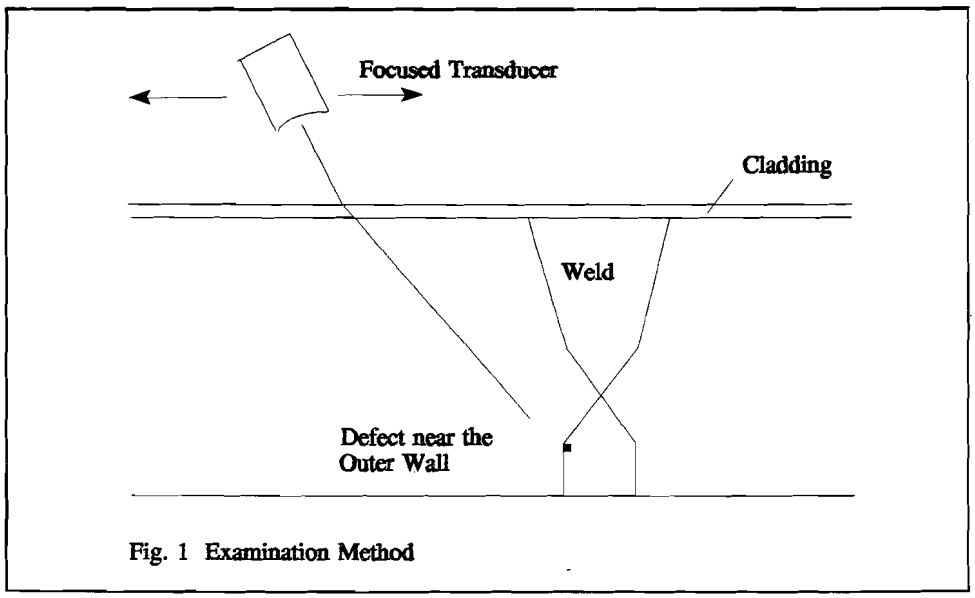


Fig. 1 Examination Method

3 EXAMPLE OF ULTRA SONIC IMAGES

A sketch of an ultra sonic Bscan image taken from /5/ is shown in Figure 2. A sketch of the test situation shows Figure 1.

The defect inside the material is a volumetric defect located near the outer wall (distance H) of the material. The ultra sonic sensor has an appertur angle of 45 and a diameter D>>d (d-diameter of the

defect) and with H<D.
The ultra sonic image of an volumetric defect can be described as follow:
The image shows 4 objects.
Object A is obtained by direct reflection of the ultra sound at the real defect. It is located directly behind the upper point of the real defect. By the location of object A in the image we can determine the real location of the defect inside the material.
Object B is left-behind object A caused by the corner effect. Because of H<D the corner effect caused by the outer wall at the left side of the sensor and the right side of the sensor cames together to one reflexion point. The size of the object is approximately two times larger then the size of the object A.
Object C is located behind object A and right-behind object B. Object C is caused by beam rotation.
Object D is located left-behind object A, B and less left-behind object C.

The sketch in Figure 1 shows an ideal ultra sonic image of a volumetric defect. A real image would be more distorted.
There are also different types of e.g. volumetric and crack like defects.

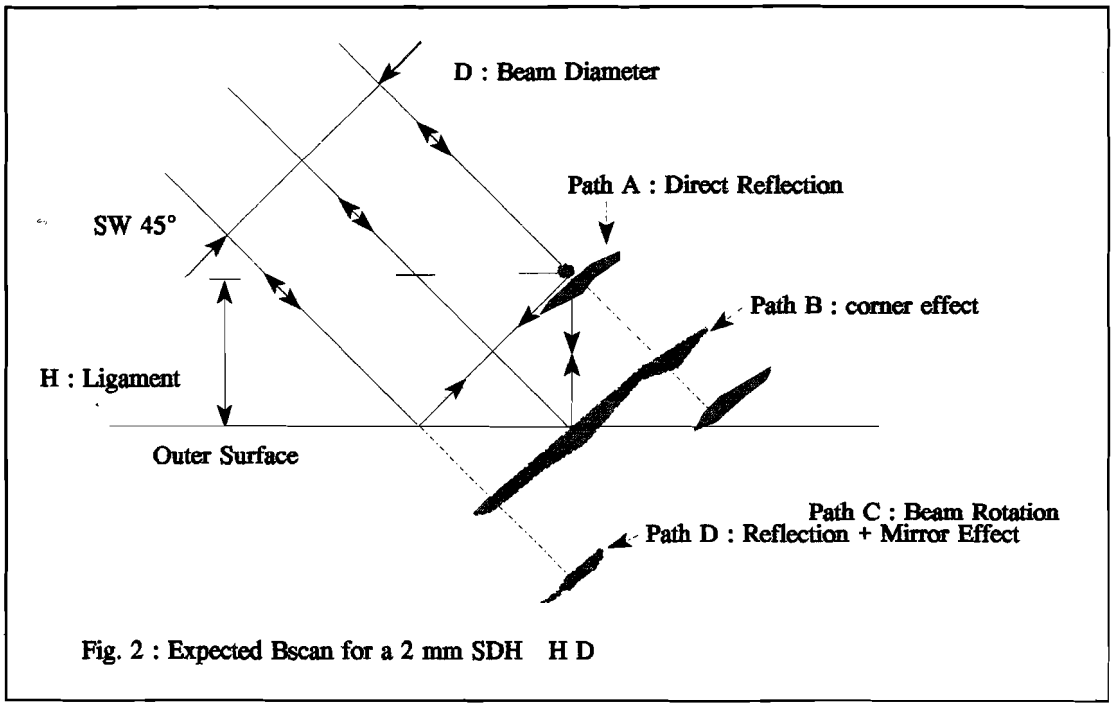


Fig. 2 : Expected Bscan for a 2 mm SDH H D

4 CASE DESCRIPTION

From Sect. 2 and 3, we can derive a case desciption which contains the following information:

- an image acquisition protocol:
  sensor parameters, the parameters of the amplifier

- an protocol about the type or the characteristic of the test component:
  In the example described above the component is a flat material with an welding seam (V-seam).
  and

- an image protocol:
  A structural description of an image consists of objects,their attributes and their relation. It may be represented in a graph-like manner. In the above described situation the objects are to consider relative to the beam angle and the track position of the sensor. The important information are:
  * the number of object,

* the spatial relationship between the objects, like "above", "behind"
*attributes like the grey level of the object which corresponds to the reflection factor, the size and the location of the object
* the shape is in a very simple manner a feature, like more longelongated or more round.

The information about the image acquisition parameter and the component very much constrain the hypotheses about defect types. This knowledge may be used for control over the case base.

Before we can use the symbols resulting in the image protocol for reasoning we have to extract the information from the image by image analysis. Procedures for the identification of an object and their description by attributes are described in /6/. In the next chapter we want to consider the qualitative description of the spatial knowledge and how we can extract this knowledge from an image.

5 RELATIVE REPRESENTATION OF SPATIAL KNOWLEDGE

For expressing the spatial representation in a qualitative manner like "above" or "above left" we need a functional model for space.
In the above described example we can think of a coordinate system which is zero in the center of mass of object A and aligned to the beam angle. Then we can describe "behind" and "above". The 4 square of the coordinate system give the spezialization "left-behind", "right-behind" and "left-above","right-above". We can shift the coordinate system from one object to another object and then look from that point of view to the spatial relations. This model would allow to describe all spatial relations concernd with the example above. It is very coarse and does not include all spatial relation like e.g. projection. The projection encloses such spatial relations like "contain" or "overlap". Therefore, Hernadez /14/ introduced an abstract map containing for each object in a scene a data structure called rpon (for relative projection and orientation node). Such an data structure can be visualized as an octagon-shaped figure, which describes also such spatial relations like e.g. "contain" and "overlap". It leads to a more complete model with respect to a concept of spatial relation.
Similar to that, Chang /7/ proposed a 2-D String for image retrieval in pictoral databases for spatial representation like orientation.
The approaches described above allows only a hard decision based on the proposed intervals. If an object is located inside of one of the parts of the rpon than it gives the spatial representation decribed by the particular area e.g. "right-back". It can not be "more-" or "less right-back". This unsharp information would require a representation of an fuzzy area /9/.

6 SIMILARITY OVER PICTURES

Which similarity is performed first seem to be very application dependent. Given a picture of an unknown person's face, we will first match the shape of each individual object and then the relative spatial relationship between eyes, nose and mouth /8/.
Different perceptions for images from different applications require to handle similarity in a flexible and adaptable manner.
As it has shown before, for our application it is important the number of objects and their spatial relations.
Only one reflection point and the second reflection point parallel shifted behind the first reflection point indicates that the defect is more volumetric-like but not crack-like. The spatial relations of the other reflection points to the first points confirms the hypotheses about the defect type. Because it can happen that the reflection points are caused by appertur the reflection factor which is proportional to the grey level gives the final conformation about the defect type. To perform similarity by the spatial relationship first and then by the attributes of the objects seem to be adequate in our application.

A good overview about similarity measures is given in /13/. Here we want to discuss structural similarity.

A flexible way to view similarity seems to be in terms of their structure. A concept "shape" will have a certain structure which shows the relationship between the different types of shape. Also an image can be described in terms of their structure (see Cha.4) and a concept derived from various images has also a structure. Therefore we need a concept for structural similarity.

An image or a concept may be represented by a graph or a graph-like structure.

Definition 1:

A graph G(P,R) consists of a set of nodes

$$P = \{p_1, p_2, \ldots, p_m\}$$, m: number of nodes

and a set of edges

$$R = \{r_1, r_2, \ldots, r_n\}$$, n: number of edges,

which connect two nodes of the graph

$$r_l = (p_i, p_j) \in R,\ p_i, p_j \in P,\ l=1,\ldots,n.$$

The general problem is to find structural identity or similarity between two structures.

Definition 2:

A graph G_1 corresponds to a graph G_2 if there exists an mapping f: $G_1 \rightarrow G_2$ such that

$$(p_{i1}, p_{j1}) = r_l \in R_{G1} \Rightarrow (f(p_{i2}),(p_{j2})) = r_k \in R_{G2}.$$

The nodes and the edges of the graph may be labeled by attributes which may or may not exact correspond to the attributes of another graph. Therefore, we need a compatibility measure which takes into account the differences between the attributes.

There are several matching techniques [6][11]:

- graph matching
- constraint search and
- relaxation.

The search for graph isomorphisms or graph homomorphisms is an NP-complete problem, meaning the algorithm is very time-consuming. The search for graph isomorphisms is only then successfull if there are no missing components. The tolerance of the algorithm against differences in the attributes and the relations depends from the accuracy of the calculation.
The advantage of the constraint search is that not all correspondences between a graph 1 and a graph 2 have to be considered. Thus, an reduction in the complexity is reached. But the result of the matching process strongly depends from the choosen constraints. Relaxation methods, especially probabilistic relaxation seem to be appropriate when there are uncertainty in the data.

It is clear that the complexity is much higher if we have to consider the whole case base. In order to reduce the complexity of the matching procedures it seems be usefull to establish a partial order of the cases. Therefore, Glasgow [9] describes a technique, called image subsumption. In a first step, an initial subsumption hierarchy is built up by an nonincremental concept learning procedure. This hierarchy is refined then during the use of the system by an incremental concept learning procedure. The subsumption hierarchay is build up by searching for part indentity of a case image and the new image which should be stored in the case base. A new concept is establish if cases stored in the case base do not subsumbe the new image completely. In that way, a concept hierarchy is built up step by step and dynamically changed according to the observation.
A feature based indexing schema is proposed in [8]. Other indexing schemes which are related to graph theory are the connectivity index, indexing based on path number and weighted path. Our intension for fuerther investigation is to give a more structured overview about structural similarity measures.

7 FEATURES OF A CASE-BASED REASONING SYSTEM FOR IMAGE INTERPRETATION

A CBR-system for image classification needs to have some particular features with respect to images. This features result from:
- the special requirements by visual knowledge acquisition (image-language problem) and
- the need to transform the numerical data of an image in a symbolic description.

The main problem with images and their translation in a language is that the knowledge about the image is usually implicit represented in humans mind. To make the knowledge explicit is often hard. Sometimes the meaning of the word does not reflect right the meaning of the image. Therefore it is necessary to support the operator in an efficient way. Therefore a CBR system for image interpretation should have a special case acquisition tool. For a more detailed description of that problem the interested reader is refered to /10/.
The problem of signal-to-symbol transformation we have already described in Section 4.

Therefore a CBR-system for image interpretation should have besides the common feature of an CBR-system:

- CASE ACQUISITION TOOL,
  which supports the user by specifying the important feature of a case,
- VISUALIZATION FUNCTION,
  which allows the user to inspect images and compare them,
- PLANNING MODULE for the image processing algorithm
- INTERFACE to an IMAGE PROCESSING UNIT and
- IMAGE DATA BASE,
  where images can be stored, retrieved and displayed.

8 CONCLUSION

In the paper, case based reasoning for image interpretation in nondestructive testing has been discussed. The case description and the requirements to the system were described based on an example for an ultra sonic image inspection problem.
In opposition to other CBR-systems there are special requirements to the case acquisition unit. The case acquisition unit should support the user by describing an image.
An automatic image inspection system based on CBR needs also an image processing unit for the signal-to-symbol transformation.
The aim of the paper was not to describe a special realization more then it should be described the problems concerned with CBR for image interpretation. Therefore, similarity over pictures as well as the problem of signal-to-symbol transformation for spatial information has been discussed briefly.
Our intension for further work is to study in more detail the opportunities of the methods statistical classification, knowledge-based classification and case-based classification for image inspection in nondestructive testing.

ACKNOWLEDGEMENT

The autor would like to thank Prof. Kroening for the opportunity to make studies in ultra sonic image inspection at the Fraunhofer-Institut of Nondestructive Testing Saarbruecken. Part of this work has been supported by the German Scientific Foundation (DFG),grant no. Pe-465/1-1, within the framework of the subject "A methodology for the development of knowledge-based system for image interpretation in nondestructive testing".

REFERENCES

/1/ L. Froehlich et. all,"Rechnergestuetzte Diagnose von Schweissnahtfehlern," BHM, 136. Jg. (1991), Heft 1,S.37-42.

/2/ G. Weinfurter,"Materialindentifikation mit Ultraschall aufgrund charakteristischer Merkmale gestreuter Impulse," Beitraege zur Jahrestagung der DGzfP, Kiel 25-27.09.89.

/3/ A. Kehoe, G.A. Parker,"An IKB defect classification system for automated industrial radiographic inspection," Expert Systems, Aug. 1991, vol.8,No.3, pp.149-157.

/4/ P. Perner et. all,"Qualifiziertes Pruefen - Eine methodische Betrachtung der Intelligenz des Pruefens,"DGzfP-Jahrestagung Luzern 6.5.-8.5.91.

/5/ M.S. J. Petitgand, D. Champigny, J.-C. Cocquillay," Ultrasonic examination of Defects Close To The Outer Surface," Report 1990

/6/ Cl.-E. Liedtke, M. Ender,Wissensbasierte Bildverarbeitung, Springer Verlag 1989.

/7/ C.C. Chang, S.Y. Lee,"Retrieval Of Similar Pictures On Pictorial Databases," Pattern Recognition, col. 24, No. 7, pp. 675-680, 1991.

/8/ T.-Y, Hou et. all," A content-based indexing techniqu using relative geometry features," SPIE Vol. 1662 Image Storage and Retrieval Systems 1992, pp. 59-68.

/9/ D. Conklin, J. Glasgow,"Spatial Analogy and Subsumption," To appear in Machine Learning: Proc. of the Ninth International Conference (ML92), Morgan Kaufmann.

/10/ P. Perner,"A Repertory Grid Based Knowledge Acquisition Tool for Visual Inspection Tasks," submitted to IEEE expert Dec.1992.

/11/ H. Niemann, Pattern analysis and understanding, Second Edition, Springer Verlag 1990

/12/ D. Hernandez,"Relative Representation of Spatial Knowledge: The 2-D Case," Report FKI-135-90, Aug. 1990, TU Muenchen.

/13/ St. Wess,"PATDEX/2-ein system zum adaptiven, fall-fokussierenden Lernern in technischen Diagnosesituationen," SEKI Working Paper SWP-91-01 (SFB)

A Rule - Rule - Case Based System For Image Analysis

S. Venkataraman , R. Krishnan and Kiron K. Rao
Advanced Data Processing Resaerch Institute
203, Akbar Road , Tarbund
Secunderabad - 500 003 , INDIA

Abstract

In this paper, the development of an object oriented case representation environment and indexing techniques are being discussed. The shell development is designed to handle 2D and 3D image data. It includes a visual programming environment. The environment has an object oriented pictorial album, its corresponding textual interpretation details (Context Knowledge) and the spatial measurable features (statistical attribute data) with low level vision techniques. Knowledge based indexing and Nearest Neighbourhood (NN) indexing techniques are used for retrieval and storage of the cases. Indexing is by a two tiered scheme with a primary and secondary key. The test-bed for the shell evaluation is done with remotely sensed data.

1.0 Introduction

Image interpretation of remotely sensed data is an open textured problem that lacks a strong domain model. A moulding of rule based techniques and Case Based Reasoning techniques is being attempted in our Rule - Rule - Case based system (RRC) to work in an interactive fashion. The central idea is to apply the rules to the target problem of the scene identification to get a first approximation of results. Later, another rule based technique and a case based technique are used to identify or classify objects in the scene. An agenda based controller schedules the rule and case based reasoning mechanism. This multi - level processing utilizes both bottom up (data driven) and top down (case driven) approaches in order to acquire sufficient knowledge to accept or reject any hypothesis for matching or recognizing the objects in the given image. The environment frame comprises of a case library of chips of images, a set of display routines, image enhancement routines, image edge extraction algorithms etc., Additionally, it has an icon editor to add or delete an interpreted chip of image or any 3D data for the corresponding 2D image chip. Programmable Hieararchial Interactive Graphics System (PHIGS) software handles the 3D image data for visualizing the image in parallel and perspective views in all possible angles, scale and view planes. This complete development is under the X-Window environment on Sun Workstation.

2.0 Image Analysis

Image analysis or image understanding refers to knowledge based interpretation of a scene which has been sensed by any sensor by computers [1]. Remotely sensed images refer to those which have been obtained from either an aircraft or spacecraft. Analysis is an interdiscipilinary research which includes different domains like image processing, statistical and syntatic pattern recognition and artificial intelligence techniques. While image processing deals with well defined mathematical convolutions image interpretation is an ill structured complex domain. Image interpretation refers to the correspondence (i.e mapping) between the description of the scene and the structure of the image. The scene is described in terms of objects in the world, while the structure of the image is described in terms of image features. There exists a wide gap between these two levels of information. the goal of image understanding syatems is to bridge the gap by computation and reasoning techniques. Hence, image analysis stresses knowledge representation and reasoning methods for scene interpretation. A semantic representation of the objects and their inter-relations is a first approximation knowledge representation schema that is possible. Rule based methods in conjunction with a case based reasoning techniques is an ideal solution for image interpretation.

3.0 Case Based Reasoning Approach

Case Based Reasoning (CBR) is an alternative way to problem solving that offers solutions to the problem of brittleness and knowledge acquisition bottleneck [2,3]. Image understanding is an apt domain for case based reasoning as rule based systems alone have proved to be very brittle and knowledge acquisition and representation is never complete in description of semantics. The major problems with CBR approach are the choice of proper indexing technique, storage and retrieval.

3.1 Case Based Shell Development

The shell developed has a CBR environment clubbed with Image processing software making it a total domain for image understanding (Figs.1 and 2). This system supports two modes of operation viz., a Development mode and a Consultation mode. During the development mode, the domain specific expert populates the case base. His supporting knowledge like which image processing tool is useful for the image under consideration, what are the steps involved etc can be added into a text file which is linked to each icon. The auxillary context information can be stored in a context file. During the consultation mode this case base is used for interpretation. Images of size upto a maximum of 256x256 pixels can be stored in the base with a display facility of four images per page. The images are classified with respect to the objects and are stored in a hierarchical manner. The images along with its attribute data are arranged in an object oriented pictorial database. Analysis reports in textual form are also tagged to each image in the database.

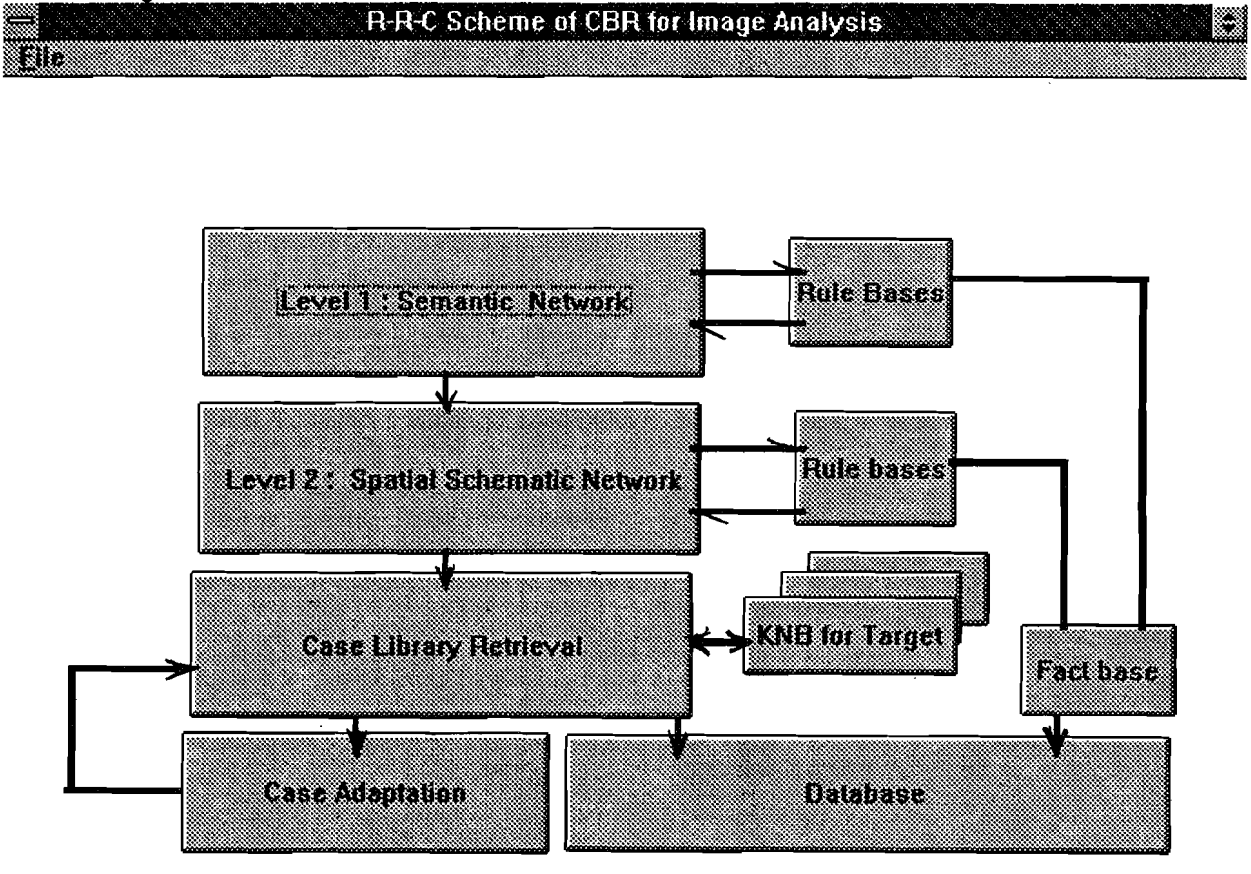


Fig. 1 R - R - C structure for Image Analysis

Images from any sensor (satellite or aerial or medical) can be stored in an object oriented pictorial (OOP) structured manner in a picture album. The OOP structure is based on the sensor and object indices. 2D image data and 3D data if any to the corresponding 2D image can be stored in the picture album. Display and manipulation of 3D data like the 3D structure lay out of a refinery for example is handled by PHIGS software. Parallel and perspective views with scaling and rotation can be handled very effeciently by PHIGS.

Image Processing tool box comprises of various tools like Image display where in one can display an image, convert file formats, obtain histogram and statistics of an image and Image editing is supported. Image enhancement option allows the user to perform various image enhancements both in the spatial as well an in the frequency domain. Edge analysis contains tools for obtaining a single pixel width edge map. On Screen Digitization (OSD) allows us to draw lines, circles, polygons etc and add text. This is used as an overlay file. Image restoration contains basis restoration algorithms. Image Math can be used to add, subtract, multiply are divide a pair of registered imaged, This also allows one to add or subtract a constant to an image. Feature extraction module allows us to extract shape features like area, perimeter, centroid, geometric moments, orientation, bounding ellipse, bounding rectangle and corner points. The input to this module is a single pixel width contour. The set of features is written on to a database. Classification module contains unsupervised classification techniques. Any of these modules can be applied on the image for interpretation before the case library is searched for similar cases or for populating the case library.

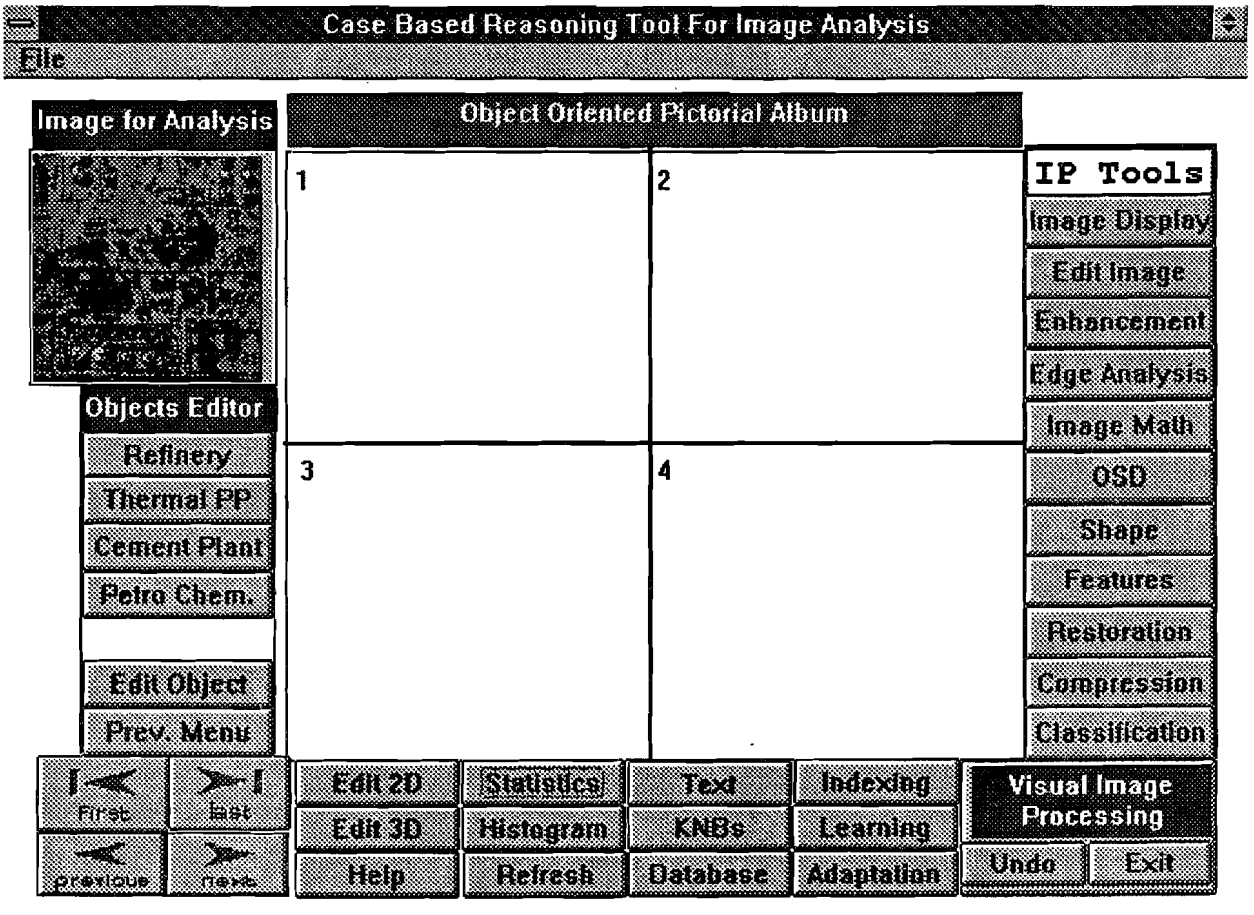


Fig. 2 . Case Base Reasoning Tool overview

3.2 Case Description

Majority of the pictorial information in an image cannot be fully described by textual and numerical information due to its essential limitations in expressing power. Two level semantic knowledge descriptor is used for knowledge representation and index key generation. Each schema is represented by a graph where nodes are associated to different objects and the pointing arrows represent relationships between the two objects.

The first level of semantic network allows the user to represent knowledge about the sensor and the image characteristics. Rule based system forward chains this data to arrive at the primary key index (Fig 2). Each entity in the sematic network is of an OOP structure.

The second level of semantic representation, geometrical shapes, spatial information and spectral properties of the object are dealt with. The semantic network grows with the type of input the user answers at each node of the network. The second rule based system looks at the fact values and forward chains to fire a rule or set of rules deciding the secondary indexing key.

The two rule bases mentioned above eleminate the necessity of storing the geometrical and spectral descriptions of imges in the case library.

3.3 Indexing

The indexing of an image assignes to the image a set of descriptions that can provide an indication of the controls of the image and a means of retrivial from the two levels. The basis of image indexing is a semantic representation of the images. A primary key from the first semantic level and the secondary key from the spectral / spatial semantic level together determine the indexing key (figs 3 and 4). Knowledge indexing and sometimes a combination of knowledge indexing and nearest neighbour method are the two methods of image indexing. Knowledge indexing refers to finding relevent pieces of facts in a knowledge base with the help of a set of descriptive properties called index. Forming an index involves the combination of intensities, shape and textural properties of the segmented regions. In general, regions will very likely extend over more than one semantic object and regions derived from different features may spatially overlap. While some regions may refer to good indices others may fail to do so. Choosing an indexing technique that can provide efficient retrival of relevant cases from the memory is a difficult task. Indexing techniques are often domain specific and task specific and thus limit the general purpose utility of the memory.

3.4 Image Retrieval

The query language in the system is a combination of retrieval by examplatory image and by textual description of the image content. When a user's request can be expressed in terms of the extracted image description, there is no need to retrieve and process the original images. If however, the textual information is not sufficient, all images are processed at the picture level to compare them with the image example. Since the image retrieval by example is the present mode of operation for retrieval in our system.

3.5 Learning

Updation of the interpreted images into the case library is possible in the development mode of operation. The interpreted data (text and attribute values) along with the images are classified and stored in the case base.

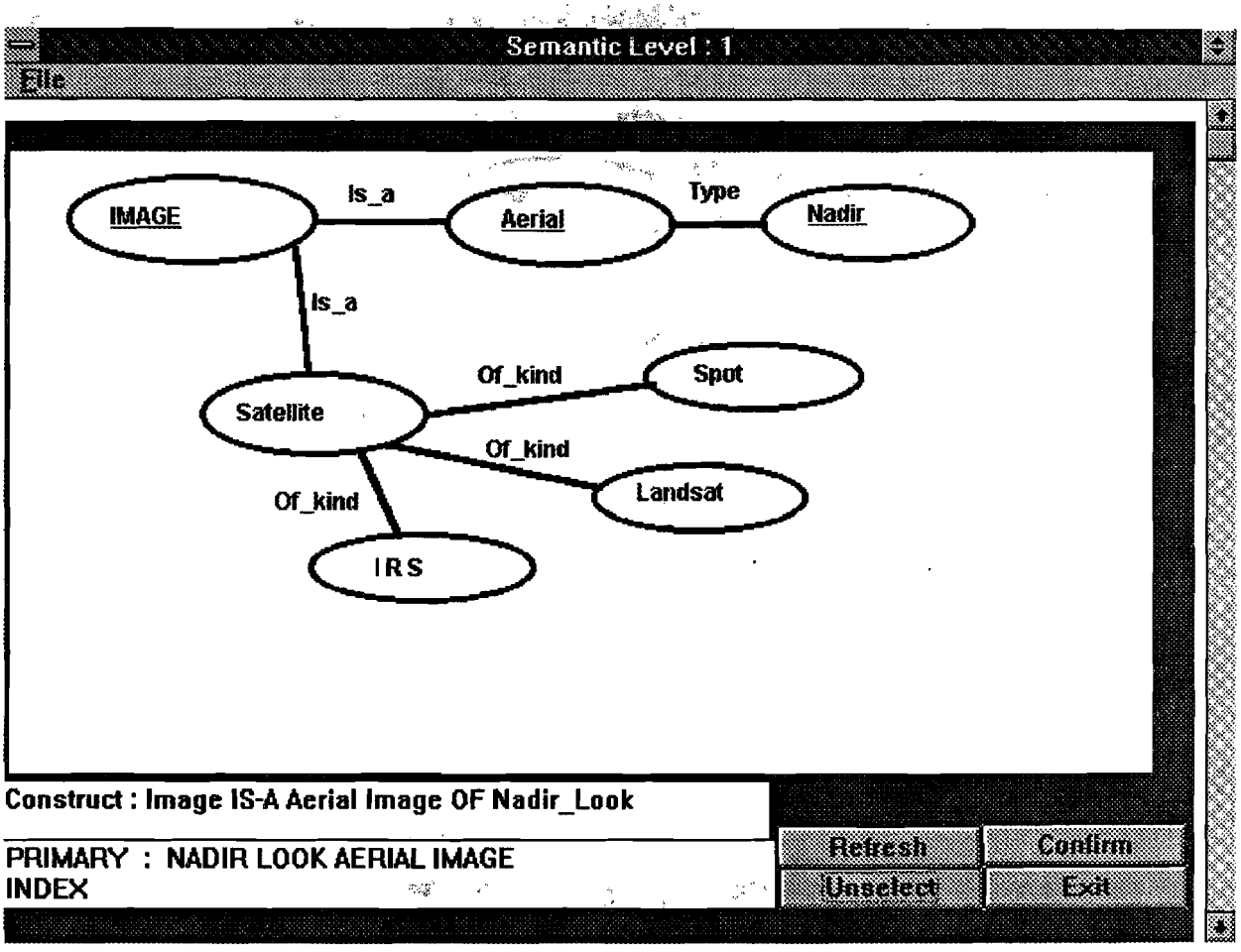


Fig. 3 Semantic Level 1

4.0 Conclusion

A tool for image analysis of any image with Case Based Reasoning, Image Processing and Knowledge based system is presented. Knowledge indexing and sometimes along with nearest neighbourhood indexing are needed for indexing the images for storage and retrieval.

Acknowledgement

We gratefully acknowledge the encouragement and critical reviews provided by A.K.S. Gopalan , Director, ADRIN, during the various stages of this project.

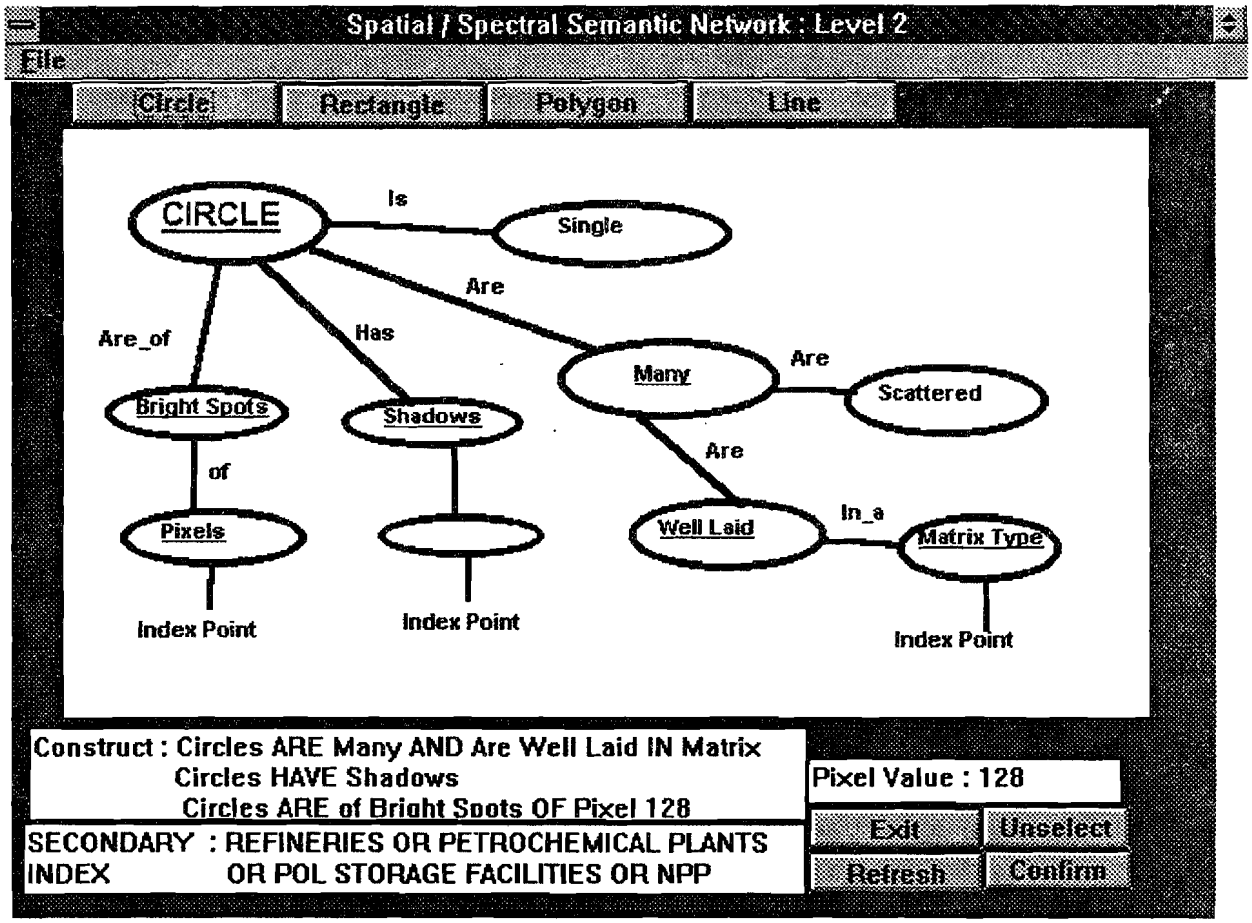


Fig.4 Spatial / Spectral Semantic Network

5.0 References

[1] Matsuyama T and Hwang Vincent S , SIGMA, 'A knowledge based aerial image understanding system", Plenum Press, NY (1990).

[2] Barletta, R , "An introduction to Case Based Reasoning", AI Expert, pp 43 - 49, Aug 1991.

[3] Kolodner, J, "Improving Human Decision making through Case Based decision aiding", AI magazine, Vol 12, No. 2 , pp 52 - 68, AAAI Press, Summer 1991.