Retrieval & Adaptation in Déjà Vu, a Case-Based Reasoning System for Software Design

Barry Smyth
Hitachi Dublin Laboratory
O'Reilly Institute
Trinity College Dublin
Dublin 2
IRELAND
Email: Barry.Smyth@hdl.ie

Mark T. Keane
Department of Computer Science
O'Reilly Institute
Trinity College Dublin
Dublin 2
IRELAND
Email: Mark.Keane@cs.tcd.ie

Abstract

Several current AI techniques are based on the reuse of problem solving knowledge. Case-based reasoning (CBR) is one such technique. In CBR problem solutions are stored as cases, and to solve a new problem a suitable case is retrieved and adapted. This paper examines adaptation in the context of a case-based reasoning system for software design called Déjà Vu. The paper describes Déjà Vu's two-tier approach to adaptation which utilises both domain specific and domain independent adaptation knowledge. In addition we describe how Déjà Vu provides two adaptation support mechanisms (adaptive problem decomposition and adaptation-guided retrieval) to relieve the adaptation load and how these mechanisms use adaptation knowledge in different ways to achieve their goals.

Introduction

Case-based reasoning tries to do away with the sophisticated knowledge needed by first-principles reasoning systems in favour of cases and methods for retrieving and adapting these cases (Kolodner 1993). Adaptation methods tend to be "weak" or heuristic in nature; an adaptation system needs to know how to alter parts of an existing solution, it does not need to know how to build complete solutions from scratch. Such limited adaptation mechanisms are feasible because it is assumed that the retrieval methods are adept at selecting the best case for a given target specification, and that the case-base is sufficiently well populated to ensure that an adaptable case exists for all target problems. Thus weak adaptation methods must be supported by strong retrieval methods.

Déjà Vu is a case-based reasoning system developed to investigate the potential for CBR methods to assist or automate Plant-Control software design tasks. Plant-control software is a type of device control software for controlling robotic vehicles within industrial environments.

In Déjà Vu two different types of adaptation knowledge are used: domain specific *adaptation specialists* and domain independent *adaptation strategies*. The former carry out specialised and fixed adaptation tasks. The latter are designed to resolve any conflicts that arise due to interactions between these specialists and solution components.

Due of the difficulties associated with adaptation, Déjà Vu's basic philosophy is to try to minimise the adaptation requirements when solving a new target problem. It does this in two ways. Firstly, a hierarchical structuring of the case-base is used to support problem decomposition. This means that a target problem may be solved by retrieving and adapting a number of cases, each solving a sub-problem of the target. This decomposition improves the possibility of finding a good match between a target sub-problem and a case and thus reduces the adaptation load. Secondly, a retrieval method called adaptation-guided retrieval is used to ensure that cases are retrieved according to their adaptation requirements. This has the effect of selecting that case which is the easiest to adapt, rather than that case which is the most "similar" to the target. Our experience has been that traditional notions of similarity fall short of accurately predicting adaptation cost and their use tends to unnecessarily burden the adaptation system by retrieving sub-optimal cases.

In the next section we briefly outline Déjà Vu and the plant control software domain. Section 3 describes Déjà Vu's adaptation methods and sections 4 and 5 discuss the additional roles that adaptation knowledge plays in problem decomposition and retrieval.

The Déjà Vu System & Domain

The application domain of Déjà Vu is plant-control software design. Plant-control programs regulate the action of autonomous vehicles within real industrial environments. The examples in this paper are taken from a steel mill environment where a system of track-bound vehicles (called coil-cars) load and unload spools and coils of steel.

Figure 1 illustrates a sample plant layout and Figure 2 a schematic of a basic Load/Unload task with a coil-car, a mill (tension-reel), a loading bay (skid), and a spool or coil of steel. In Load/Unload tasks coil-cars are used to load or unload spools or coils of steel from a mill to a skid. Very briefly, these tasks involve complex sequences of actions whereby coil-cars are aligned with skids and tension-reels to pickup and release spools or coils.

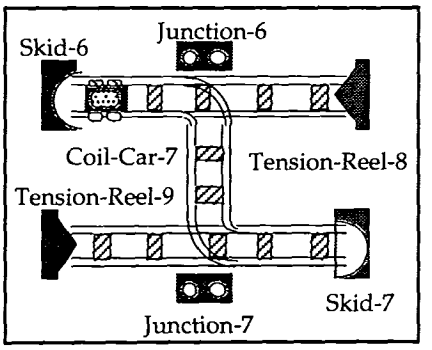

Figure 1. A Simple Plant Layout

The domain in which Déjà Vu is expected to operate has the advantage that the programming language is very high level with operations represented as nodes on a solution graph (see Figure 3 for an example). This graph can be compiled by a code generation system to produce machine executable code (Sakurai, Shibagaki, Shinbori, and Itoh, 1990; Ono, Tanimoto, Matsudaira, and Takeuchi, 1988)

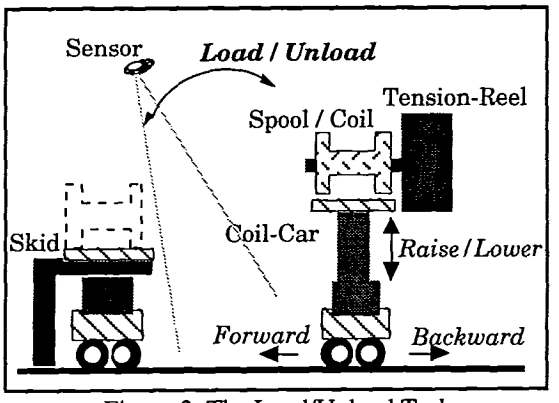

Figure 2. The Load/Unload Task

The example code in Figure 3 shows a small part of a much larger Unload program. It moves a vehicle (coil-car-7) to a loading bay (skid-7) using a "2-speed" operation. The vehicle is initially moved forward at its "fast" speed, then at some point it is slowed down, and finally, on reaching its destination, it is brought to a halt; both the slowing and stopping conditions are detected by position sensors.

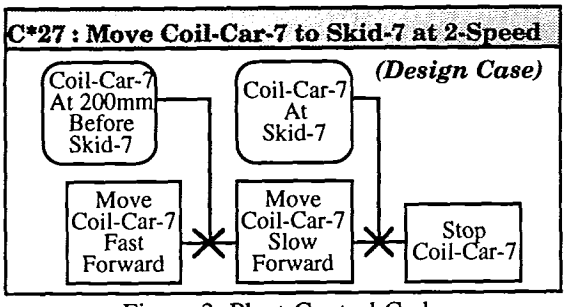

Figure 3. Plant-Control Code.

The rectangular nodes correspond to plant-control actions. The oval nodes correspond to sensor checks. The "cross marks" mean that program control cannot pass until both inputs have been complete. For example, the first "cross mark" of Figure 3 indicates that the coil-car will not be slowed down until it has first been moving at its fast speed *and* it has reached a point 200mm before the skid.

Adaptation in Déjà Vu

Déjà Vu uses an adaptation scheme that facilitates both specific local modifications, through the action of *adaptation specialists*, as well as global conflict resolution, via *adaptation strategies*. This two-tier scheme can be compared with Hammond's work on CHEF (Hammond, 1989). In CHEF adaptation knowledge is also divided into domain specific *critics* and more general *strategies*. The main difference between the two approaches is that Déjà Vu's adaptation knowledge is specially formulated so that it can be used during retrieval to *predict* adaptation cost, as well as used during adaptation to perform the necessary modifications.

Adaptation Specialists

Adaptation specialists correspond to packages of design transformation knowledge each concerned with a specific adaptation task. Each specialist can make a local modification to a retrieved case. During adaptation many specialists will act on the retrieved design to transform it into the desired target design in an incremental fashion. As well as procedural knowledge each specialist also has declarative knowledge describing its particular adaptation capability. In this way specialists are organised in terms of the modifications they are designed to carry out. In a future section we will describe how this capability information can be used during retrieval to predict adaptation requirements.

```
(DefSpecialist Speed-Specialist
  (base-speed target-speed target-slow-dist)

  :Tests (= target-speed 2-speed)
         (= base-speed 1-speed)

  :Actions

    (Add-Before (MOVE SPEED SLOW)
      (MOVE SPEED FAST))

    (Add-Before (MOVE SPEED SLOW)
      (DIST-CHECK DISTANCE target-slow-dist)))
```

Figure 4. A Speed Specialist

For example, in the plant-control domain, one common difference between a retrieved case and a target problem is that the speed of the target specification may differ from that of a retrieved case. To cater for this situation Déjà Vu uses a dedicated *speed specialist* (see Figure 4) which can satisfy the speed requirements of the target by modifying

those of the retrieved case. In this example specialist, the capability information indicates that it can be used to transform 1-speed movement cases into 2-speed movement cases; the capability is represented as a set of relevant index features (base and target speed and target slowing distance in this example) and a set of applicability tests. The action component specifies how the modification can be carried out. In the example, a node must be added to the base solution to move the vehicle at its fast speed; this node is added before the existing "move slow" node. In addition, a distance check node is also added before the "move slow" node to ensure that the vehicle slows down at the correct place, namely when the target vehicle reaches a certain position before the destination location. This position is defined by the target vehicle's slowing distance.

Adaptation Strategies

In the course of adapting a retrieved design case it is possible that interactions will arise within the modified design solution. This is because specialists do not consider the modifications made by other specialists and so interactions that occur go unchecked and may ultimately lead to design errors and solution failure. In the past, the detection and resolution of such interactions has been one of the stumbling blocks of many planning and automated design systems (Hendler, Tate, Drummond 1990). Déjà Vu attempts to overcome this problem by using an efficient scheme of interaction representation and resolution. Using a set of adaptation strategies, Déjà Vu can detect and repair many interactions that might arise. Strategies are organised in terms of the interactions they resolve and each is indexed by a description of the type of failure it can repair (again its capability information). Of course each strategy also has an associated method of repair for resolving the conflict in question.

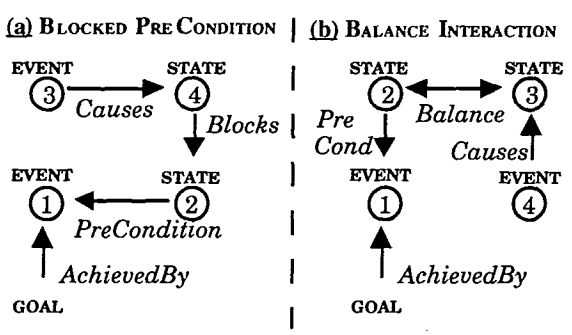

Figure 5. Two Adaptation Strategies

As an example, one very common type of interaction occurs when the effect of some event prevents the occurrence of some later event. Figure 5(a) depicts this situation; some goal event (1) is prevented by the disablement of one of its preconditions (2), the precondition having been blocked by some earlier event (3) causing a conflicting state (4). This *blocked pre-condition* interaction can be repaired in a number of ways. For instance, an event could be added before the blocking event (3) which prevents its blocking effect. The blocked pre-condition adaptation strategy contains a description of this situation as well as the appropriate repair methods. For example, increasing the speed of a vehicle may result in a precondition of a movement goal being blocked due to the lack of power; increasing speed has the effect of increasing power consumption, which in turn has the effect of reducing power availability. This conflict between speed and power may be avoided by scheduling a recharging event before the move operation takes place.

Another type of interaction, a *balance-interaction*, can occur when the value of one state is proportionally dependent on another (see Figure 5(b)). Here, some necessary goal-achieving event (1) has a precondition state (2) that depends on another state (3) that has resulted from some other event (4). For example, before moving a coil-car across the factory floor the height of the carrying platform must be adjusted to accommodate the load being transported; there is a balance condition between the height of the lifting platform on the coil-car and the diameter of the coil of steel being carried. If this balance is not properly maintained then a failure may occur (for instance, the coil-car may collide with an overhead obstacle). The repair procedure associated with this strategy involves adjusting the effected design component as appropriate; for example, lowering the lifting platform to accommodate the new load.

An Example Adaptation Session

The following example works through a sample adaptation session taken from the plant-model of Figure 1. The target problem is to move coil-car-7 from tension-reel-9 to skid-7 using 2-speed motion and carrying coil-1, a coil of steel. The case memory contains just a single case for moving a coil-car from tension-reel-8 to skid-6 using 1-speed motion, and carrying no coil of steel. The relevant adaptation knowledge consists of four specialists designed to alter the speed, direction, start and end locations, and the contents of vehicles in movement tasks. Two strategies are also relevant; the blocked-precondition strategy and the balance-interaction strategy, both mentioned above. These relevant specialists and strategies are identified, not during adaptation, but during retrieval. This will be described in a later section. For now we will assume that the relevant adaptation knowledge has been found.

Figure 6 illustrates a "before-and-after" snapshot of the adaptation session. The adapted base solution components are shown in the target solution as shaded elements. The application of the specialists is fairly straightforward. The speed specialist changes the base case from 1-speed motion to 2-speed motion by making structural changes to the solution. Two extra solution nodes are added, one that propels the coil-car at its fast speed initially and a second that detects when the vehicle must slow down. The direction and destination location specialists make simpler substitutions as shown.

The work of the strategies is more complex. The BlockedPreCondition strategy recognises that an increase in vehicle speed may lead to a problem due to a limited vehicle power. This strategy determines whether repair action is necessary by checking the coil-car's power reserves. In this example we will assume that the vehicle's power reserves are not significantly depleted and hence no repair action is scheduled. However, the second strategy, the BalanceInteraction strategy, does detect a problem in that the coil-car's carrying height must be adjusted to accommodate its new load; in the base case the vehicle carried no load. This strategy contains a very straightforward repair procedure which simply adjusts the vehicle carrying height by lowering its carrying platform.

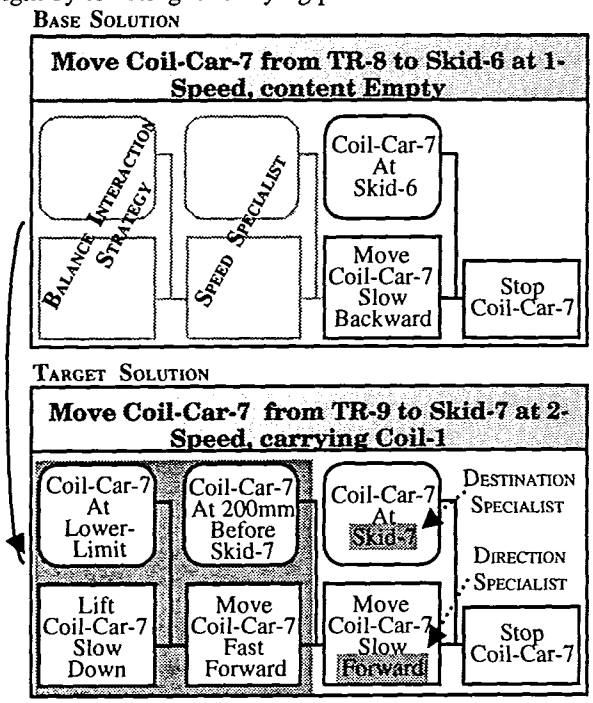

Figure 6. An Example Adaptation Session

Problem Decomposition

The main advantages of Déjà Vu's adaptation scheme are that adaptation knowledge is relatively simple to encode and many different types of changes can be catered for including substitutional and structural ones. The scheme works well once the base case is sufficiently similar to the target problem; that is, provided there are specialists and strategies available that can perform the necessary adjustments. However if the base case is significantly different from the target then problems are likely to occur as adaptation becomes more brittle when many sequences of specialists and strategies must be chained together.

This poses a problem for plant-control software design because target specifications tend to be complex, and if complete designs are stored as single cases then there are likely to be significant differences between the target and base. To combat this Déjà Vu stores complex design as case hierarchies and uses decompositional design techniques to facilitate the reuse of multiple cases (Smyth & Cunningham, 1992). Each case solves a much simpler part of the target specification and so base to target similarity is likely to be greatly improved, thus relieving the adaptation requirements.

Similar research has been carried out by Redmond (1990) in the CELIA system where complex problems are represented by collections of case *snippets* with each snippet addressing one particular goal. In addition, goal structure is maintained by linking snippets together to form hierarchies. However, Déjà Vu's hierarchies are task oriented rather than goal oriented. Each case in Déjà Vu can handle a group of task related goals rather than a single goal. In addition, Déjà Vu's cases are represented at varying levels of abstraction whereas Redmond's snippets are all stored at the same level of abstraction. This in turn leads to differences in the way that decomposition and recombination are handled by each system. CELIA is biased towards reusing snippets from a single problem collection whereas Déjà Vu will tend to mix cases from many different hierarchies in order to minimise adaptation cost.

Case Hierarchies

Déjà Vu's cases can be divided into two basic types, design cases and decomposition cases. Design cases correspond to what one would normally expect to find in the case-base of a software design system. That is, they store software design solutions. The solutions found in decomposition cases, on the other hand, are somewhat different. While they may contain some actual software code, they also contain further problem specifications, stored in *link descriptors* nodes, which define new sub-problems that must be solved before the current one can be completed.

Collections of these cases are organised hierarchically and correspond to complete, complex software designs. The solution of the root case of a hierarchy corresponds to the most abstract view of the problem represented by the cases in this hierarchy. The actual software code of the hierarchy can be reconstructed from the individual solutions of each of the design cases located at the leaf (or terminal) nodes. Any intermediate cases store solutions at intermediate levels of design abstraction.

Figure 7 sketches one particular case hierarchy. The hierarchy solves an unload task; a coil-car must unload a coil of steel from the milling press (tension-reel) and deposit it on a skid. The complete solution is quite complex and is built up from many separate design cases. The inserts show the details of two of the cases (C*11 and C*27) in the hierarchy.

The lower insert of Figure 7, shows a design case and the top insert shows a second level decomposition case. The details of this solution are very different from those of the design case. As mentioned, the solutions to decomposition cases include references (link descriptors) to further sub-

problems. The C*11 insert shows three such references. It is through these references that the individual cases are tied together as hierarchies, where each reference in the hierarchy points to a case at the next level down; the "move" link descriptor of C*11 points to C*27.

However, it is important to recognise that these references are indirect in the sense that lower level cases are never explicitly named by the link descriptors of their more abstract parent cases. The link descriptors are treated as case specifications just as if they were the specification to a new target problem. This means that, for example, if the "move" link descriptor is used as a retrieval probe, then case C*27 could be retrieved.

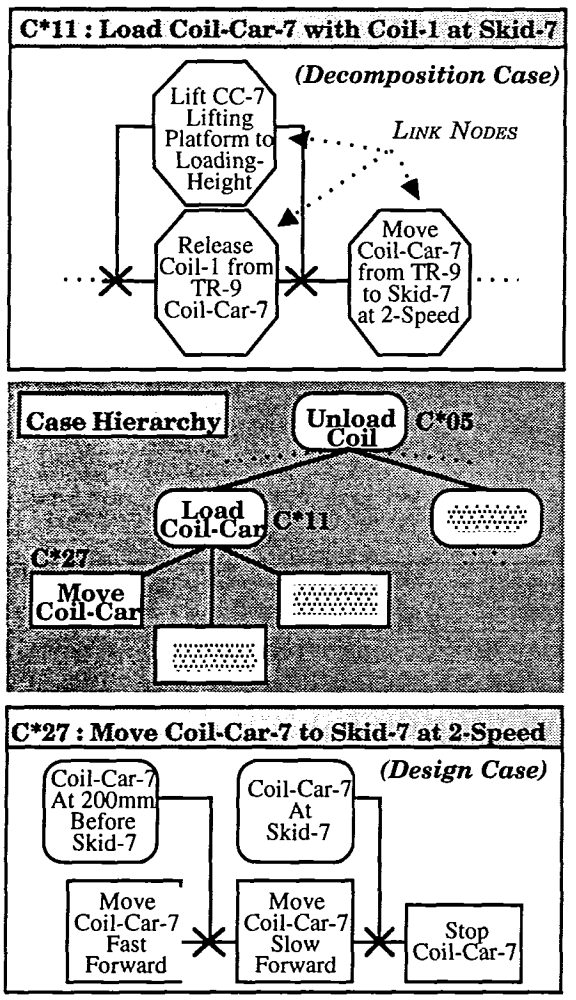

Figure 7. A Portion of a Sample Case Hierarchy

Decomposition & Adaptation

During problem solving, decomposition occurs because a decomposition case is retrieved and adapted. In fact one of the major benefits of integrating case-based reasoning and decompositional design in this way is that problem decomposition can be elegantly implemented in terms of the basic CBR processes of retrieval and adaptation. Retrieval now ensures that a decomposition strategy (decomposition case) suitable for the current target problem is selected. Adaptation ensures that this selected strategy is specially tailored for the needs of the target situation.

The adaptation of a decomposition case occurs in much the same way as the decomposition of a regular design case. Instead of changing solution nodes that correspond to actual plant-control commands, the link descriptor nodes are altered. Again specialists and strategies are used. Depending on the precise details of the target and base case this adaptation can mean anything from simple substitutions to more complex structural changes such as the deletion or addition of link nodes.

After adapting a decomposition case, each link descriptor is used as the specification of a new sub-problem for Déjà Vu to solve. As solution components corresponding to sub-specifications are constructed they are integrated into the overall target solution.

Furthermore, because these decomposition cases can be adapted it is possible to solve a target problem by reusing cases from a number of different case hierarchies; Déjà Vu is not restricted to using only those cases from one hierarchy. This is because by modifying link descriptors, the altered sub-specifications, when passed on to the next CBR cycle, may cause the retrieval of cases from other hierarchies; a case may be present in another hierarchy that better matches the newly adapted sub-specification. This means that better use can be made of the case-base. A best possible case is always being sought, even if the complete problem that this best case is a part of is, when taken as a whole, not well suited to the complete target problem. This is in keeping with one of the basic principles of CBR, that of trying to reduce adaptation costs through the retrieval of cases.

This can be contrasted with CELIA's approach where direct links are provided between related snippets. CELIA prefers to move directly from snippet to snippet, following explicit "next snippet" links. Only if a snippet turns out to be invalid does CELIA resort to the retrieval of a new snippet (possibly from a different problem collection). While this turns out to be practical in CELIA's domain it can lead to poor results in Déjà Vu. It ultimately leads to a preference for cases from the same hierarchy. When this happens many cases are reused even though more suitable cases may exist in other hierarchies. This can lead to expensive, sub-optimal adaptation stages.

Adaptation-Guided Retrieval

As a further way of limiting adaptation requirements, Déjà Vu employs a novel approach to retrieval. Traditional retrieval techniques use semantic similarity as a *proxy* for adaptation, their assumption being that the case which is the most *similar* to the target problem will also be the easiest to adapt (Bariess and King 1989; Kolodner 1989). However, this assumption is not always well founded. At best these conventional retrieval techniques select cases which are sub-optimal with respect to adaptation, and at

worst they retrieve cases which cannot be adapted properly at all. Déjà Vu's retrieval method, called adaptation-guided retrieval (Smyth & Keane, 1994), uses adaptation knowledge during the retrieval stage to ensure the selection of a case that is the easiest of those available to adapt. A further benefit of this is that during retrieval the adaptation knowledge that is relevant to a given base case is identified as part of the matching process. This means that after retrieval Déjà Vu has access to, not only the best available case, but also those specialists and strategies that are relevant to the adaptation of this case. Thus, not only is the most adaptable case retrieved, but also some preliminary adaptation work (the recognition of relevant adaptation knowledge) is carried out during the retrieval stage.

Adaptation Knowledge During Retrieval

In order to guarantee the retrieval of a case that is the easiest to adapt, the retrieval mechanism must give explicit consideration to *how* cases will be adapted. To achieve this without actually performing full adaptation, Déjà Vu uses adaptation knowledge during retrieval to *predict* the adaptation requirements of a candidate case. Put simply, a target feature X should only be matched to a candidate feature Y if there is evidence (in the form of adaptation knowledge) that Y can be adapted to give X.

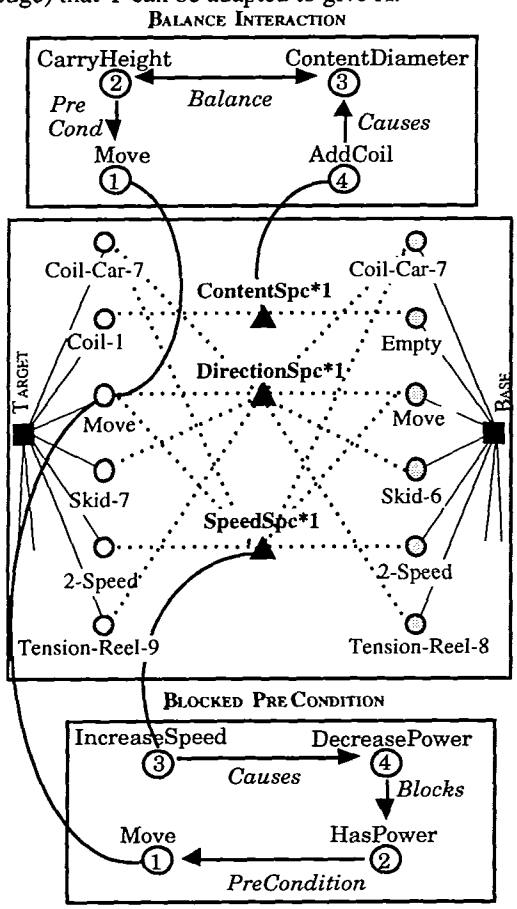

Figure 8. Adaptation Knowledge in Retrieval

The key to being able to make these type of predictions during retrieval is to store, in the adaptation knowledge, some representation of adaptation capability. This capability information can then be used during retrieval. For example, in Figure 4 we see the capability of the speed specialist which indicates that a 1-speed movement case can be adapted to support 2-speed movement. Similarly, the event-state graphs of the adaptation strategies are also used during retrieval to predict possible conflicts, and only during adaptation is repair action initiated. In adaptation-guided retrieval adaptation is *not* carried out, instead *predictions* are made that *diagnose* the need for adaptation.

In our previous adaptation example we looked at how a particular 1-speed case was transformed into a 2-speed case. Next we briefly outline how adaptation knowledge is used during retrieval to make these predictions, and how, during retrieval, relevant adaptation knowledge is identified.

Figure 8 illustrates the type of structures that are built up during retrieval (although simplified for the sake of clarity). Firstly, non-identical matches between the speed, content, direction, and location related features of the base and target are associated with specialists which can perform the necessary adaptations. Secondly, a number of strategies are also activated due to the presence of certain enabling features. For example, the BlockedPreCondition strategy is activated because of the predicted speed increase and the BalanceInteraction specialist is activated because of the addition of the coil of steel.

Conclusions

There is a school of thought that suggests that case-based methods are more correctly concerned with *retrieval* than *reuse*. This suggests that adaptation in CBR should, at all costs, be kept to a minimum. The reasons for this are relatively straightforward. For one thing it is difficult to automate adaptation and generalised adaptation knowledge is hard to come by. For another thing heuristic adaptation may degrade the quality of the solutions. This means that the quality of the final target solution may be significantly less than the quality of the retrieved case's solution.

This philosophy of trying to avoid adaptation sets CBR apart from many other reuse methodologies, especially derivational analogy, where due to the availability of a base-level planner and richer, more knowledge-intensive case structures, high-quality adaptation is possible (Veloso 1992).

This paper described Déjà Vu a case-based software design system which subscribes to this "limited adaptation" view of CBR. One objective in developing Déjà Vu was to investigate whether this CBR perspective was simply too restrictive for complex automated design tasks. Our initial results are encouraging and demonstrate that sophisticated design automation is possible as long as support mechanisms are in place that reduce the adaptation load. We described two such support mechanisms, both of which use adaptation knowledge in different ways. The first,

decomposition, uses case hierarchies and adaptation to decompose target problems into simpler problems that can be more readily solved by the retrieval and adaptation of individual cases. The second method, adaptation-guided retrieval, uses adaptation capability information to predict the adaptation needs of candidate cases.

So far a number of experimental studies have been carried out to evaluate our approaches. It is shown in Smyth & Keane (1995) that Déjà Vu's adaptation-guided approach to retrieval results in more accurate case retrievals than conventional distance-based similarity metrics. In particular, it is demonstrated that while Déjà Vu succeeds in selecting the easiest to adapt case for a target problem conventional methods retrieve these optimal cases as little as 12% of the time. In addition it is shown that because of this failure to retrieve the easiest to adapt case, conventional retrieval methods mean overly complex adaptation stages. The result is that, taken over many target problems, the overall problem solving time (retrieval plus adaptation time) for Déjà Vu with adaptation-guided retrieval is significantly better that when standard, distance-based methods are used.

Finally, our methods have also been applied to the design of MOTIF graphical user interface code, and initial studies are equally encouraging in this software design domain. In particular, similar design ability was witnessed as the system was seen to generate complex (multi-window) GUI designs. Although this version of the system has not be experimentally tested it is expected that similar results to the plant-control system will be found. In other words, retrieval should be significantly more accurate than conventional methods resulting in gains in overall problem solving performance.

References

Bareiss, R.; and King, J. A. 1989. Similarity Assessment in Case-based Reasoning. In Proceedings of the Case-Based Reasoning Workshop, 67 - 71. Florida, U.S.A.

Hendler, J.; Tate, A.; and Drummond, M. 1990. AI Planning: Systems and Techniques. *AI Magazine*, 11(2): 61-77.

Kolodner, J. 1989. Judging Which is the "Best" Case for a Case-Based Reasoner. In Proceedings of the Case-Based Reasoning Workshop, 77 - 84. Florida, U.S.A.

Kolodner, J. 1993. *Case-Based Reasoning*. Morgan Kaufmann

Hammond, K. 1989. *Case-Based Planning*. Academic Press.

Ono Y.; Tanimoto I.; Matsudaira T.; and Takeuchi Y. 1988. Artificial Intelligence Based Programmable Controller Software Designing. In Proceedings of the International Workshop on Artificial Intelligence for Industrial Applications, 85-90.

Redmond, M. 1990. Distributed Cases for Case-Based Reasoning; Facilitating Use of Multiple Cases. In Proceedings of AAAI-90. Cambridge, MA: AAAI Press/MIT Press

Sakurai T.; Shibagaki T.; Shinbori T.; and Itoh S. 1990. An Automatic Programming System based on Modular Integrated-Concept Architecture (MICA). In Proceedings of International Conference on Industrial Electronics, *Control and Instrumentation, 1303*-1308. Monterey, California, U.S.A.

Smyth B.; and Cunningham P. 1992. Déjà Vu: A Hierarchical Case-Based Reasoning System for Software Design. In Proceedings of European Conference on Artificial Intelligence, 587-589, ed. Bernd Neumann, John Wiley, , Vienna Austria.

Smyth, B.; and Keane, M. T. 1995. Experiments on Adaptation-Guided Retrieval in Case-Based Design. In Proceedings of the 1st International Conference on Case-Based Reasoning. Sesimbra. Portugal.

Smyth, B.; and Keane, M. T. 1994. Retrieving Adaptable Cases: The Role of Adaptation Knowledge in Case Retrieval. *Topics in Case-Based Reasoning*, 209-220, Lecture Notes in Artificial Intelligence.: Springer-Verlag.

Veloso, M. 1992. Learning by Analogical Reasoning in General Problem Solving. Ph.D. dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, USA.