APT, a cooperative ML system

Claire Nédellec
LRI, bât 490, Université Paris-Sud
F-91405 Orsay cedex, France
e-mail : cn@lri.fr

Abstract

In this paper, we present APT system as an example of a learning apprentice that learns from examples of expert problem-solving. Learning in APT is not fully automated but *cooperative*, so that APT is partially in charge of tasks that are usually left to the user, such as acquisition of examples, revision of input data and validation of learned knowledge. The efficiency of APT cooperation relies on the fact that it is based on a problem-solving context that is common to both the expert and the system because its exploits examples of expert problem-solving.

1. Introduction

Various AI tools are available that partially or fully automate different phases of a KBS life cycle: knowledge acquisition, design, validation and maintenance. Our purpose here is to show through APT system that the integration of various knowledge acquisition, machine learning, validation and problem-solving methods in a same architecture can speed-up application development by greatly simplifying the validation-revision task compared to fully automatic or fully manual methods. The main idea underlying APT principles is that appropriate and correct knowledge can be acquired from an expert by recording the expert problem solving activity and generalizing, validating and exploiting it through cooperation of the expert himself with Problem-Solving and Machine Learning tools.

Our purpose here is not to formally describe the Machine Learning and Knowledge Acquisition methods of APT, but APT's features that make possible to fruitfully integrate Knowledge Acquisition from the user and automatic Machine Learning activities in the same system. APT's methods are detailed in [Nédellec, 92, 94]. APT is a *learning apprentice* as described by [Mitchell, 89], [Tecuci & Kodratoff, 89], [Dent et al., 92]: it learns by generalizing the method applied by an expert while solving a problem so that the method learned is automatically applicable to *similar* problems. A learning apprentice is composed of two modules, a Problem-Solver and a Learner, the role of which is to improve the Problem-Solver performances.

Although learning can be described as problem-solving, for sake of simplicity, we will distinguish in the following between learning on the one hand, and problem-solving on the other hand, as the main activity of the learning apprentice the result of which the end-user is interested in (such as classification, planning, etc).

2. Problem solving

APT works in two separate modes, a *solving* mode and a *learning* mode to be set by the user.

2.1. Solving mode

In solving mode, the problem-solving unit (*Problem-Solver*) only is active. It is the regular mode used when learning is no longer useful: the performances of the Problem-Solver are considered as satisfying. In this mode, the user of the system is the end-user. In APT's current version, the problem-solving method implemented consists in decomposing a problem provided as input data with its context expressed as a set of ground facts, into subproblems. The decomposition has the form of a tree where the leaves are elementary problems that do not need to be further decomposed (in a similar way as in problem-reduction method, [Nilsson, 71).

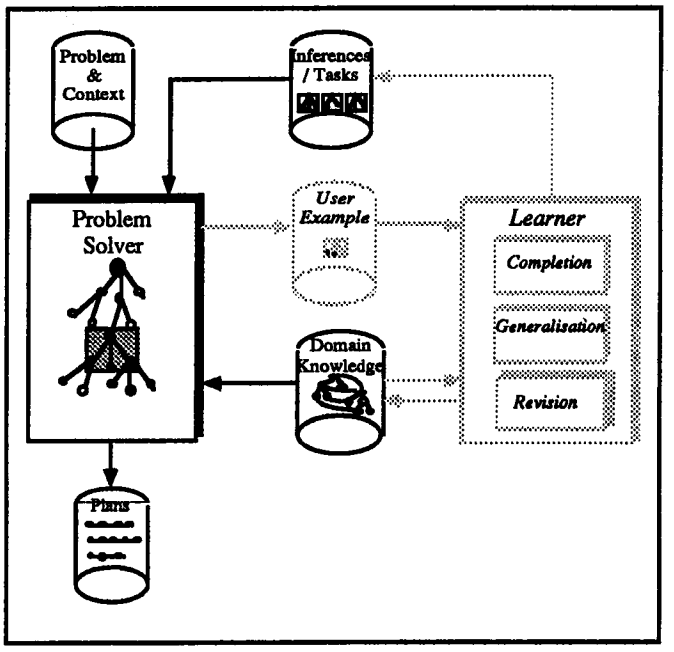

Figure 1. Problem-Solving mode

The Problem-Solver exploits knowledge bases that we will describe by making a parallel with KADS model of expertise, [Wielienga et al 92]. The domain layer is represented by a semantic network in a *Domain Theory* which describes the concepts of the domain and their possible relationships. The generality relation between concepts is restricted to hierarchies. Figure 2 gives a toy example of such a Domain Theory that will be used in the following to illustrate the methods.

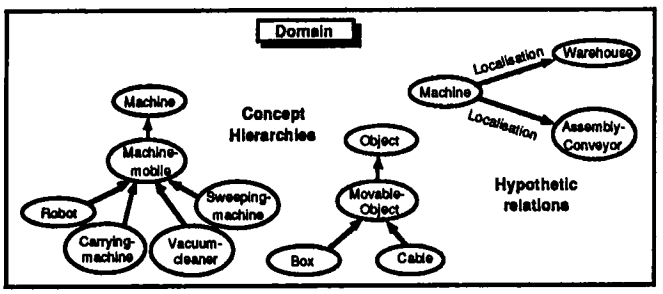

Figure 2. Domain layer example

The inference layer is represented by *inference rules*, the input (or preconditions) and output (or effects) of which are expressed by conjunctions of concepts and relations of the Domain Theory in a restriction of First Order Logic (FOL), without functional terms nor negation (Figure 3). APT also exploits a library of *elementary task structures* (Figure 3) that decompose tasks into a sequence of subtasks. Input and output of task structures are in the same form as inference structures ones.

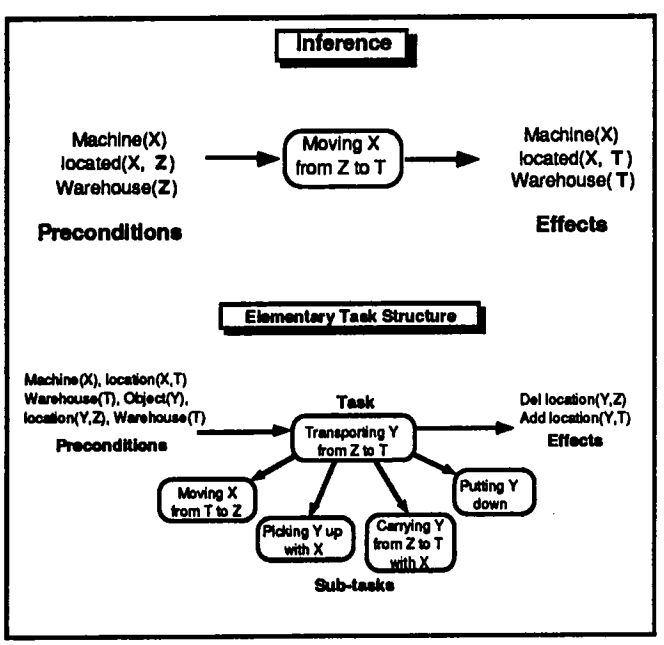

Figure 3. Inference and Task layers examples

To solve a given problem, APT builds an instanciated task structure by dynamically combining instanciated elementary task structures and inference structure (figure 3) in the form of a tree. At each solving step, APT selects an elementary task or an inference that solves the current problem. An elementary task structure can solve a problem if it decomposes it and if its preconditions are more general than the context of the problem with respect to generalized subsumption [Buntine, 88] with respect to the domain theory. The same way, an inference structure can solve a problem if its preconditions are more general than the context of the problem.

If the current problem is solved by a task, the sub-tasks of the selected task replace the current problem after having been instanciated. If the problem is solved by an inference step, then it is considered as elementary. In both cases, the context of the solved problem is updated as a consequence of the effects of the selected task (resp. inference) similarly as in planning. Problem solving ends when there is no more problems to solve.

For instance, the problem PB1: `Transport b12 from w1 to w3` with its context:

```
Robot(R2D2), Warehouse(w3),
Box(b12), Warehouse(w1),
Location(R2D2,w3),
Location(b12,w1), Open-
Door(w1), Open-Door(w3)
```

may be solved by the task structure represented Figure 3 with the domain described Figure 2. One can check that the preconditions, `Machine(X)`, `Warehouse(Z)`, `Location(X,Z)`, are more general than the context. As a consequence of the problem reduction, `Transport` problem is replaced by the `Moving`, `Picking up`, `Carrying` and `Putting Down` sub-problems and the fact `Location(b12,w1)` is replaced in the resulting context by `Location(b12,w3)`. At the next solving step, the `Moving` problem can be solved by the inference Figure 3 and the fact `Location(R2D2,w3)` being replaced by `Location(R2D2,w1)` in the resulting context and so on, until the subproblems `Picking up`, `Carrying` and `Putting Down` will be fully reduced. Problem resolution results then in an instanciated task structure where the leaves form a linear plan.

2.1. Learning mode

In learning mode, the user is the expert of the application domain or more precisely both the knowledge engineer and the expert. Their role is to teach specific problem-solving knowledge to APT when APT fails to solve a problem. The Learner generalizes it and includes it into the Problem-Solver libraries.

In order to make easier the knowledge acquisition task and to guarantee the relevance of the acquired knowledge with respect to the problem-solving task, the acquisition is performed in the course of a specific problem solving. In learning mode, the

Problem-Solver works in the same way as in solving mode, except that at each problem-solving step the Problem-Solver submits to the user the candidate tasks or inferences which can solve the current problem. The candidate solutions are instanciated[1] by the facts of the context of the current problem and the problem-solving historic is available so that the user can more easily evaluate their relevance.

For instance, in learning mode, APT would have submitted the instanciated task structure Figure 4 to the user in order to reduce the `Transport` problem example presented section 2.1. Notice that the concept and variables have been specialized according to the current problem.

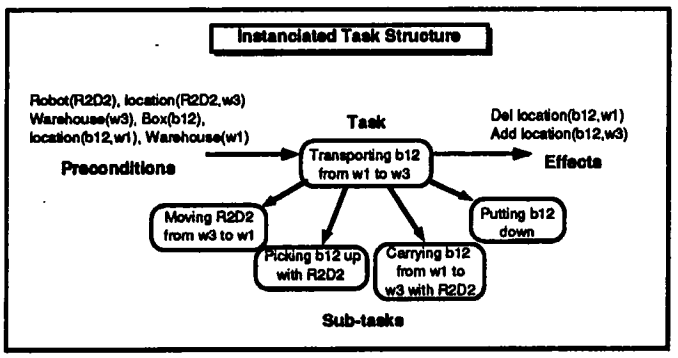

Figure 4. Example of instanciated task structure

In case the submitted solutions are considered as not satisfying (or in case there is any), the user gives the Problem-Solver an alternative solution expressed in the form of instanciated sub-problems and effects, or simply effects if the problem is elementary.

Let us suppose now that the task library is empty, APT has thus no solution to solve the `Transport` problem, `Transport b12 from w1 to w3` and the user tells APT that this problem can be solved by decomposing it into the subproblems :

1. `Moving R2D2 from w3 to w1`
2. `Picking b12 up with R2D2`
3. `Carrying b12 from w1 to w3 with R2D2`
4. `Putting b12 down`

where `Robot(R2D2),Warehouse(w3), Box(b12),Location(R2D2,w3), Warehouse(w1),Location(b12,w1)` is the subset of the current problem context that is sufficient for the decomposition to be valid and the effects are Add `Location(b12,w3)`, Del `Location(b12,w1)`. APT's Learner interrupts the problem-solving activity and generalizes this solution into a task structure (Figure 4) before coming back to the problem-solving task. This is detailed in Section 3.

A typical development scenario of an application with APT is the following ; at the beginning APT has to be provided with an initial Domain theory only, possibly incomplete and incorrect. The first time a problem is submitted to APT, APT has have no solution to solve it. Then the user gives his own solution that will be generalized as an inference or a task structure by the Learner and then will be reusable and augment the problem-solving library. This teaching process will be repeated through problem solving until the user will consider that the Problem-Solver's behavior is satisfying and its library of tasks, inferences and domain knowledge is complete and correct with respect to the task.

As the problem-solving activity is specific in the sense that it aimed to solve a specific problem in a particular context, the user is not asked to reason about abstractions. He has to evaluate and to provide concrete solutions in a way that is more similar to his expert activity than if he would be asked to provide general problem-solving methods. However a knowledge engineer should participate in the acquisition task of problem-solving knowledge in order to represent the expert knowledge in APT's knowledge representation language and intepret APT's deductions if needed.

3. Cooperative Learning

APT's Learner does not simply record the expert's specific solution but generalizes its *preconditions* so that the solution could be applicable to all relevant cases and if necessary, corrects and completes the knowledge of the Domain Theory (Figure 5). These tasks require to be efficient some validation and knowledge acquisition from the user. They are performed in a cooperative way so that the user can be efficiently guided. The cooperation is based on the same principles as problem-solving: the interaction with the user is restricted to the specific context of the current problem-solving failure that is common to both APT and the user. We will briefly describes APT's learning methods before discussing the distribution of tasks among the user and the system. More details can be found in [Nédellec, 92] and [Nédellec, 94]. For sake of generality from a Machine Learning point of view, we will describe the learning task in terms of examples and concepts instead of task and inferences.

The concept to learn here is the concept of *applicability* of the user's solution. The definition of the concept is thus expressed by the conditions of the applicability of the solution, that correspond to the input of the task (resp. inference) structure to learn. Let us call operator, the problem plus the

[1] The candidate task (resp. inference) structures are specialized by SLD resolution of the task (resp. inference) preconditions with the problem context with respect to the domain theory.

subproblems. The definition of the concept of applicability of the operator op_R is represented by a set of Horn clauses,

$\forall X_1, ..., X_n$, Applicable($op_R(X_1, ..., X_n)$) $\leftarrow$ Preconditions(op_R).

where op_R denotes the operator.

The subset of the current problem context that is pointed out by the user as conditions for the solution to be valid is first variabilized. It represents an initial and specific definition of the concept to learn that APT will generalize. In our example, it is

```
Applicable(opR(X,Y,Z,T)) ←
  Robot(X), Warehouse(Z), Box(Y)
  Warehouse(T), Location(X,T),
  Location(Y,T).
```

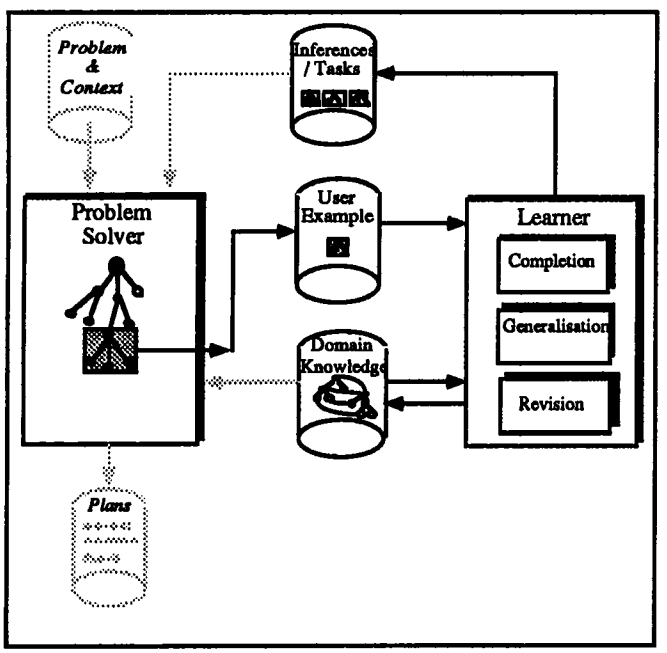

Figure 5. Learning mode

APT's learning consists in four successive phases, completion, generalization, validation and finally revision if needed.

3.1 Completion

Although the user's solution is specific, the user may omit implicit but necessary facts in the description of the conditions, the initial definition will then be incomplete and consequently too general. If it is not correctly elicited, APT learns overgeneral concept definitions, applicable to irrelevant situations. The role of the completion phase is to add these needed conditions to the initial definition. On the one hand, APT suggests completions that the user validates. To be as relevant as possible, completions are generated from both the context and the relations of the Domain Theory. On the other hand, the user can provide completions by himself. They cause Domain Theory revision in case the added literals are not defined in the Domain Theory.

3.2 Generalization

The completed definition is handled as a starting clause, lower bound of the search space for the concept definition. Applying a bottom-up, breadth-first and generate-and-test strategy as MARVIN's [Sammut & Banerji, 86], APT generates disjunctive candidate concept definitions by applying the inversion of resolution operator with the Domain Theory. One generalization step consists in replacing *one* literal of the definition by its father in the Domain Theory. In order to validate the generalization step, APT automatically generates by analogy one discriminant example[2] that is submitted to the user. Depending on the classification, the candidate concept definition is validated or not. By construction, the search space is a complete lattice of candidate definitions where the discriminant example sets are disjoint and assumed to be consistent.[Nédellec, 94] proves the learned concept definition to be complete and correct with respect to the class of the generated examples and the number of clauses learned to be minimal

For instance, in the `Transport` example, the first generalization steps generates four candidate definitions where `Robot(X)` is replaced by `Mobile-Machine(X)`, `Box(Y)` by `Movable-Object(Y)`, `Warehouse(T)` by `Place(T)` and finally `Warehouse(Z)` by `Place(Z)` with respect to the Domain Theory (Figure 2).The following generalization steps consists in performing two replacements together, then three and finally four, before climbing the Domain Theory hierarchies one step more.

3.3 Validation

At the end of the generalization phase, the learned concept definition is evaluated by the user who has to say if it is overgeneral or not. Let us suppose that the learned concept definition in our example is the following one,

```
Applicable(opR(X,Y,Z,T)) ←
  Mobile-Machine(X),
  Warehouse(Z),  Warehouse(T),
  Movable-Object(Y),
  Location(X,T),  Location(Y,T)
```

and that the user evaluates it as overgeneral. The overgenerality reveals that the assumption of the consistency of the discriminant example sets was not satisfied. The revision phase identifies the inductive leap that caused the overgeneralization and then correct the overgeneral concept definition, and the Domain Theory if needed.

---

[2]A discriminant example of a candidate definition is an example that is covered by the candidate definition and covered by none of the more specific definitions.

3.4 Revision

First of all APT backtraces through the search space in the direction of the starting clause, following the specialization direction pointed out by the user if there is one. At each specialization step, it submits the candidate definition to the user who evaluate it as overgeneral or not, until it reaches the limit between overgeneral and overspecific concept definitions. As APT has performed the smallest generalization steps, the discriminant example set of the most specific overgeneral (MSOG) concept definition must be inconsistent, (in light grey in Figure 6). APT then generates all the discriminant examples of the MSOG concept definition it can. It asks the user to classify them and corrects the MSOG concept definition with his help, so that the resulting learned definition is correct and complete with respect to the discriminant examples. The concept definition is then added to the Problem-Solver libraries as task (resp. inference) structure(s).

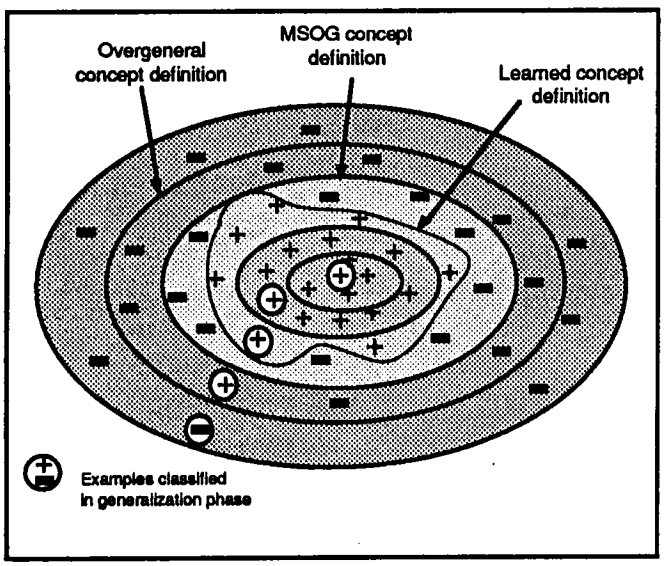

Figure 6. Example distribution within the search space

Let us suppose in our example, that the MSOG concept definition is also the learned concept definition. Its discriminant examples will be the three examples that are combinations of descendants of `Mobile-Machine(X)` that are not `Robot` (i.e. `Vacuum-cleaner`, `Sweeping-machine`, `Carrying-machine`) and descendants of `Movable-Object(X)` that are not `Box` (i.e. `Cable`). The two first examples are classified as negative by the user (`Vacuum-Cleaner` and `Sweeping-Machine` cannot transport anything), while the third one is classified as positive (`Carrying-Machine` can transport `Cable`).

Available revision operators are automatic (mostly based on predicate invention [Muggleton, 88], [Nédellec, 94]) and semi-automatic (adding literals). Possible revision operations are proposed to the user who selects the relevant ones with respect to the discriminant examples and provide the revision module with domain knowledge when needed. For instance, he has to name the invented predicates.

In the `Transport` example, APT learns the following concept definition,

```
Applicable(opR(X,Y,Z,T)) ←
  Transporting-Machine(X),
  Warehouse(Z), Movable-
  Object(Y), Warehouse(T),
  Location(X,T), Location(Y,T)
```

where the invented predicate `Transporting-Machine` has replaced `Mobile-Machine` in the body of the clause, and completes the Domain Theory (Figure 7).

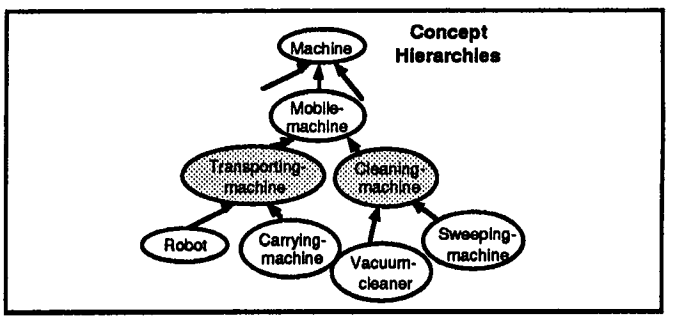

Figure 7. Revised Domain Theory

3.4 Learning task distribution

The roles of APT and the user differ in completion - generalization phases and validation - revision phases. In the two first phases, the role of the user is limited to providing an initial single example and validating APT's completions and generated examples. The interaction is strictly based on knowledge specific to the problem-solving context and examples. The tasks that are usually left to the user are automated here, i.e. the user has not to provide a complete set of classified examples of the concept. The completion phase limits the risk that the initial example would be incomplete, and revises the Domain Theory by integrating and generalizing as much as possible the user completion knowledge. The only examples the user has to classify are the examples generated by APT that are *necessary* and *sufficient* to learn. There is no risk here for the user to provide a redundant or incomplete set of examples, or incomplete example descriptions that would result in unsatisfying results he would then have to revise by himself. The examples are generated by analogy with the initial example in decreasing similarity order with a few risk of irrelevancy in order to make the user classification task easier.

The concept definition that APT induces is complete and correct with respect to the training examples, but can lead to bad performances when applied to real problem if the Domain Theory or the completed initial example are incomplete or

incorrect. Thus, validation and revision phases can be required. Instead of recording successes and failures while the performance element (i.e. the Problem-Solver) applies the learned knowledge and then triggering revision as many systems do, we have preferred another revision strategy: APT tries to validate the domain and problem-solving knowledge as soon as it is learned. On the user side, this avoids to use a Problem-Solver with low performances in a long training-revision cycle. However it is more demanding as the user is asked to evaluate more than simple examples but also some concept definitions as overgeneral or not. On the system side, this avoids to record the historic of all the examples of all the concept learned.

To overcome the lack of examples needed to guide the revision, APT generates examples again that are classified by the user. These examples belongs to the same discriminant example set of the MSOG concept definition, there are thus semantically close and easy to compare, similarly as with repertory grids. This allow a better elicitation of the knowledge which allow to distinguishes positive and negative examples with respect to the operator application. In the `Transport` example, for instance, the user is asked to say what allow to distinguish positive cases of "`Robot` transporting `Box`" and "`Carrying-machine` transporting `Cable`", from negative cases of "`Vacuum-cleaner`" and "`Sweeping-machine` transporting `cable`". The answer is obvious here, the *ability* of `Mobile-Machine` to transport `Movable-Object` is a discriminant knowledge.

More precisely, the elicitation of the discriminant knowledge is performed through the revision operations suggested by APT. For instance, it would propose to the user to replace the predicate `Mobile-Machine` in the MSOG concept definition by a new predicate that will be ancestor of `Robot` and `Carrying-machine` (concepts occurring in positive examples) and descendant of `Mobile-Machine` in the Domain Theory. This revision operation corrects the MSOG concept definition so that the identified negative examples are no longer covered. In parallel, APT proposes to create a new predicate in the Domain Theory to gather the corresponding "negative" predicates" occurring in negative examples, `Vacuum-cleaner` and `Sweeping-machine`. This toy example of the revisions operators APT can apply illustrates how the user can cooperate with APT by taking turn with the system in controlling the learning / acquisition activity and in eliciting / learning new knowledge : APT suggests operations, the users selects the relevant ones, APT propagates and generalizes the effects of the revision operation and if needed acquires knowledge from the user.

4. Conclusion

More than a machine learning system that would aim to output problem-solving library, APT application to various domains such as loan analysis and medical treatment, has proved that is applicable as a modeling tool that enables an expert to elicit and refined its model of expertise. In this framework, it plays the role of a naive learner that leads its teacher to clarify and reorganize the model of its own expertise. In particular, the ability of APT to automatically generates completions and examples from the Domain Theory by analogy with the initial example provided by the user appears as an interesting feature for triggering useful reactions from the expert.

The analysis of these application developments has led to reconsider the role of the knowledge engineer which was initially restricted to the representation of the expert knowledge and the "translation" of APT questions and suggestions. Application development can be speeded-up if the knowledge engineer is able to fully exploit APT's features. This requires that he has some insight about the consequences of knowledge changes on APT behavior. For instance, if he knows the general principles of the completion and the example generation methods, he will immediately react to irrelevant suggestions by making the appropriate changes in the domain theory or the initial example without going step by step through the validation and revision phases. Thus, the cooperation takes different form depending on if the knowledge engineers is a novice and or an expert of APT.

In APT system, expert explanations are available from each window that describe the learning methods underlying the current interaction (Nédellec, 95b). These explanations and APT user's guide are based on an Inductive Logic Programming and a KADS *models* of APT that are independent of the implementation (Nédellec, 95a) and clearly elicit the role of the input knowledge in the learning process and their consequence on the output. The performed experiments confirm that the ability of KBS to express helpful explanations requires a proper model of the domain knowledge involved (problem-solving methods as well as domain knowledge and input data) [Moore and Swartout, 88], [Swartout et al., 91]. It appears that producing useful explanations in ML requires not only an explicit model of the application domain ML is applied to, but also knowledge about ML itself, including the elicitation of implicit language and search bias. The need for a model of Machine Learning related to comprehensibility concerns and consistent with Knowledge Acquisition-based models it could be integrated with seems to be a crucial issue for developping efficient cooperative ML systems.

Acknowledgments

This work is partially supported by CEC, through ESPRIT BRA ILP (n°6020).

References

BUNTINE W., "Generalized Subsumption and its Application to Induction and Redundancy", in *Artificial Intelligence* 36, p. 375-399, 1988.

DENT L., BOTICARIO J., MC DERMOTT J., MITCHELL T., ZABOWSKI D., "A Personal Learning Apprentice" in *Proceedings of the Tenth National Conference en Artificial Intelligence (AAAI-92)*, p. 96-103, San Jose, AAAI Press / The MIT Press, July 1992.

MITCHELL, T. M., MAHADEVAN S., STEINBERG L. I. *"LEAP : A Learning Apprentice for VLSI Design* ", in *Machine Learning III : An Artificial Intelligence Approach*, p. 271-289, Kodratoff Y. & Michalski R. S. (Eds), Morgan Kaufmann, 1989.

MUGGLETON S., "A Strategy for Constructing new Predicates in First-Order Logic", in *proceedings of the Third European Working Session on Learning (EWSL-88)*, p.123-130, Sleeman (Ed), Glasgow, Oct. 1988.

NÉDELLEC C., "How to specialize by Theory Refinement", in *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, p. 474-478, Neuman B. (Ed), John Wiley & sons, Vienne, Aug. 1992.

NÉDELLEC C., "APT, apprentissage interactif de règles de résolution de problème en présence de théorie du domaine", PhD dissertation, Université Paris-Sud, 1994.

NÉDELLEC C., "Modeling Machine Learning and Comprehensibility" in *Proceedings of the IJCAI'95 workshop on Machine Learning and Comprehensibility*, Montréal, Aug. 1995.

NÉDELLEC C., "APT, un Système d'Apprentissage Coopératif", in *Acquisition des Connaissances, tendances actuelles* , C. Reynaud, Ph. Laublet et N. Aussenac (Eds.), to appear, 1995.

NILSSON N.J., "Problem-Reduction Representations" in *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill Company (Pub), Palo Alto, 1971.

SAMMUT, C.A AND BANERJI, R.B., "Learning Concepts by asking Questions", in *Machine Learning II : An Artificial Intelligence Approach*, pp 167-192, R.S. Michalski J.G. Carbonell and T.M. Mitchell (Eds) Morgan Kaufmann, 1986.

TECUCI G. & KODRATOFF Y., "Apprenticeship Learning in Nonhomogeneous Domain Theories", in *Machine Learning III : An Artificial Intelligence Approach*, Kodratoff Y. & Michalski R. (eds), Morgan Kaufmann (Pub.), 1989.

WIELINGA B. J., SCHREIBER A. T., BREUKER J. A., "KADS: A Modelling Approach to Knowledge Engineering", in *Knowledge Acquisition* (4) 1 , Special Issue 'The KADS approach to Knowledge Engineering, Academic Press (Pub.), 1992