Learning, planning, and the life cycle of workflow management

Diogo R. Ferreira[a], Hugo M. Ferreira[b]
[a]Faculty of Engineering U. P., Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
[b]INESC Porto, Rua Dr. Roberto Frias 378, 4200-465 Porto, Portugal
{drf@fe.up.pt, hmf@inescporto.pt}

Abstract

This paper describes an approach towards workflow management based on the combination of learning and planning. Assuming that processes cannot be fully described at build-time, the approach makes use of learning techniques, namely Inductive Logic Programming (ILP), in order to discover workflow activities as planning operators. These operators will be subsequently fed to a partial-order planner in order to find the process model as a planning solution. The continuous interplay between learning, planning and execution aims at arriving at a feasible plan by successive refinement of the operators. The approach is illustrated in two simple scenarios. The paper concludes by relating the proposed approach with previous developments in this area.

1. Introduction

For the past decade, workflow management as the "procedural automation of a business process by management of the sequence of work activities" [1] has mostly been thought as a one-way endeavor – first model, then execute – with its main assumption being that processes can be represented as an explicit model and that they can be repeatedly executed in a predictable, controlled way.

Despite its conceptual simplicity, or perhaps because of it, workflow management has long been facing challenges such as the coordination of unstructured/ad-hoc processes [2], the ability to handle exceptions [3], or the need to support process adaptation and evolution [4], some of which are still topics of current research today. All of these efforts aim at providing workflow management systems with more *flexibility* [5].

The search for flexibility, however, as it collides with fundamental assumptions – namely, that processes do not change over time – suggests either that flexibility deserves further study, or that the assumptions of workflow management should be thought over again. It is the second option that we explore in this paper. First, we challenge the assumption that all process modeling can be done prior to execution. Then we challenge the notion that a single process model is useful to be run more than once.

2. Learning workflow activities

It has since long been known the importance of *tacit knowledge* in human activities [6], i.e., knowledge that people employ in performing their tasks, but that they cannot fully explain. Since much of organizational work relies on tacit knowledge (see for example [7]), we should not rely on the assumption that people will be able to describe their work exactly as they do it.

In this case, it may be necessary to actually observe work practices, create a process model, and then check with the users to see if it is correct. Exceptions or changes in procedures will throw us back to the starting point, requiring us to gather further feedback from the users in order to come up with a refined or modified process model.

Let us assume that a process model Π is a partially-ordered set of activities where each activity $\alpha_i \in \Pi$ transforms a world state S_i into a world state S_{i+1}. In general, activity α_i can only be performed in a state S if and only if $P_i \subseteq S$ where P_i are the *preconditions* of the activity. Each activity has also a set of *effects* that change the current state S_i into a state S_{i+1}. For reasons to be explained in the next section, it is convenient to describe the effects of α_i as a set of clauses R_i that α_i removes from S_i and another set of clauses A_i that α_i adds to S_i so that the overall result as the new state S_{i+1}. R_i is called the *remove-list* of α_i whereas A_i is called the *add-list* of α_i. These entities are illustrated in Figure 1.

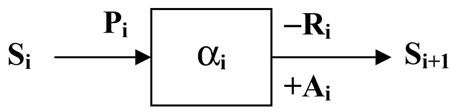

Figure 1. Anatomy of a workflow activity

Ahead in this paper we describe an auto insurance claim process. One of the activities in that process is repairing a damaged vehicle. This activity can only be done if the vehicle (V) is actually damaged and the amount of damage has already been assessed for the claim (C). The effects of this activity are that the vehicle is not damaged anymore and becomes repaired. This can be described by the following operator:

```
repair_vehicle(C,V)
   PRECOND: damaged(V), assessed(C)
   REMLIST: damaged(V)
   ADDLIST: repaired(V)
```

Now, for the above reasons, we will assume that users won't be able to accurately describe P_i, R_i and A_i for the activities they perform. They will be able to say, however, given a state S_i, if they can perform the activity or not. We will call $S_i >> \alpha_i >> S_{i+1}$ a *positive example* when α_i can be done in state S_i, and $S_i >< \alpha_i$ a *negative example* when it can not.

Provided we can collect a proper set of positive and negative examples, it is then possible to infer P_i, R_i and A_i using standard AI techniques. Inductive Logic Programming (ILP) [8] seems particularly useful for this purpose. The technique is applied straightforwardly in the following way:

- P_i is learned by specialization (top-down search) using all available positive and negative examples, where S_i is the background knowledge for each example.
- R_i is learned by generalization (bottom-up search) using all available positive examples, where $S_i \backslash \{S_i \cap S_{i+1}\}$ is the background knowledge for each example. $\{S_i \cap S_{i+1}\}$ represents the set of clauses that remain unchanged when α_i is performed.
- A_i is learned by generalization (bottom-up search) using all available positive examples, where $S_{i+1} \backslash \{S_i \cap S_{i+1}\}$ is the background knowledge for each example.

It should be noted that R_i and A_i can rely only on positive examples, since a negative example does not produce a state S_{i+1}. Hence these rules must be learned by generalization. This can be simplified, however, if we assume that it is possible to collect a set of initial positive examples in a closed environment, where a single user performs a single activity. In this case, R_i and A_i can be inferred immediately from S_i and S_{i+1} without having to learn at all.

3. Planning workflow processes

Once every activity is described in terms of its preconditions and effects – effectively, as a planning operator – then developing a process model is a matter of creating a plan, i.e. a sequence of activities that transforms an initial state S_I into a final state S_O called the *goal state*. This is only possible if, when describing activities as operators, we make use of a common vocabulary – a set of predicates with the same semantics for all activities – so that the effects of one activity can match the preconditions of another.

In general, users will have to be assisted, on an initial stage, by an "expert modeler" who defines the predicates to be used to describe world states. Later on, users are expected to be able to provide a rough description of their activities using those predicates, and a few examples before those descriptions are improved by learning. Given that a world state is defined based on the amount and kind of information available at a certain moment in time – for example, a set of records in a database –, the problem of coming up with an appropriate set of predicates from existing data sources in an enterprise information system becomes an interesting research challenge, to be explored in future work.

Once the operator definitions have been settled, then the plan for reaching a goal state from an initial state can be created, again using standard AI techniques. Partial Order Planning (POP) [9] seems particularly useful for this purpose. An obvious advantage is that partial-order planning, as opposed to total-order planning, generates plans with a maximum degree of parallelism, which is essential for long-running workflow processes. But there are several other reasons for choosing POP:

- it generates plans with a higher degree of flexibility [10];
- the planning information is easily understood by humans and allows interactive plan analysis and repair [10,11];
- it facilitates the analysis and repair of failed plans (see for example [11]);
- it facilitates the handling of domains with durative actions and temporal and resource constraints [12];
- it allows easier integration with execution, information gathering and scheduling architectures [12].

An important point about making use of planning techniques is that the process model is generated on demand, as soon as there is a goal to achieve. The plan is valid only for that goal, and will be generated again when the planning algorithm is given the same goal. On the other hand, a different goal will possibly lead to a different plan. This is in contrast with the idea – common in workflow management – of having a process model fully described by build-time, and launching it unchanged several times at run-time.

Another point worth noting is that by describing activities as planning operators and then using those operators to generate a plan, we are actually building a process model by chaining activities that have been studied independently, rather than studying a process as a whole, as it is common in workflow management. This approach allows activities to be connected in unanticipated, hopefully better (more efficient) ways, which is reminiscent of process re-engineering, but without an overwhelming analysis effort. If the operators are properly defined, then planning will provide the best plan.

4. Towards a new life cycle for workflow management

If users are not able to accurately describe the preconditions and effects of their activities, but only a rough description, and they only provide a few examples of their applicability, then it is not possible to have the operators properly defined before starting planning. And learning won't help either, due to the lack of examples.

However, if we nevertheless create a plan and ask the users for feedback regarding the possibility of executing the activities in that plan, then we will be able to collect more examples, allowing us to further refine the operator definitions. By repeating the same procedure a few times, eventually the correct operator definitions will be found, together with the intended plan. Build-time and run-time thus become intertwined, as plan execution provides examples for learning operators, which in turn are used to generate a new plan.

This life cycle is illustrated in figure 2. The user selects a goal from a set of predefined goals specified using a common thesaurus. Then, using the available operators, the Planner generates a plan. Activities may be assigned to different users, which manage their tasks through their own task lists. Trying to execute a task may turn out to be a possible or impossible action, which will result in a new positive or negative example, respectively. If some action cannot be done, these examples will be used to re-learn the operators, which will be fed to the Planner again in order to generate a new plan.

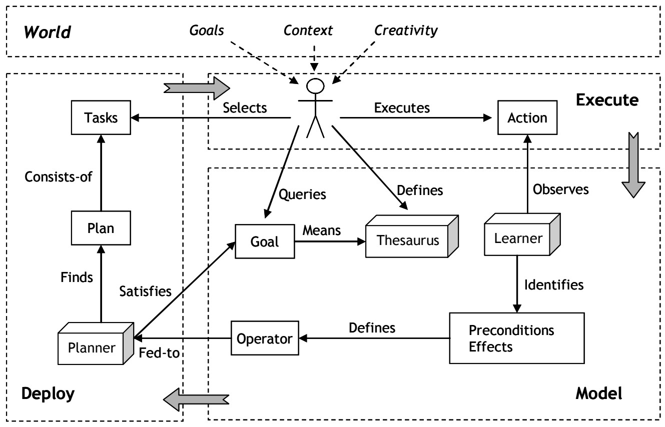

Figure 2. Learning and planning life cycle [13]

In our experiments, the user has been replaced by a simulator with knowledge about the true operators, so that it could say whether a given operator could be applied in a given state or not. Within a blocks world scenario illustrated in figure 3, the goal was to go from state S_I to state S_O using three possible operators:

- `movebb(X,Y,Z)` – Moves a block X that is on top Y to the top of block Z. Cannot be done if either X or Z have some block on top of them.
- `movebt(X,Y)` – Moves a block X that is on top Y to the table. Cannot be done if X has some block on top of it.
- `movetb(X,Y)` – Moves a block X that is on the table to the top of block Y. Cannot be done if either X or Y have some block on top of them.

The thesaurus included the predicates `on(X,Y)` meaning that block X is on top of block Y, and `clear(X)` meaning that there is no block on top of block X. A single positive example using just three blocks was given for each operator, which allowed the Learner to find out immediately its effects, whereas the preconditions remained unknown (assumed empty).

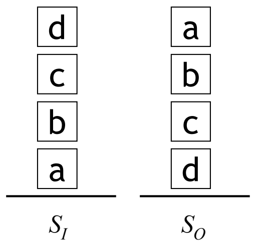

Figure 3. Blocks world scenario

After six iterations of planning and learning, the following feasible, linear plan was found:

movebt(X,c), movebb(c,b,d), movebb(b,a,c), and movetb(a,b). Surprisingly enough, this plan was found even though the operators were still not accurately described:

- movetb(X,Y) was missing a precondition: that X must be clear before being moved.
- movebt(X,Y) was missing all of its preconditions, because it was inserted in the plan only on the last iteration. This explains why X in movebt(X,c) remained unbound, since the precondition clear(X) was missing and so the Planner never attempted to satisfy it, which would have caused X to become bound to d.
- The preconditions of movebb(X,Y,Z) were correctly learned. This operator was used in every planning attempt, so it has collected more (negative) examples than the other two.

We have conducted further experiments, in which the initial examples did not allow the Learner to know the exact operator effects from the beginning. In these scenarios, however, the algorithm has quickly fallen into a situation where it was impossible to create a plan. This is due to the fact that as the preconditions are being refined (becoming increasingly demanding) but the effects are still incomplete, POP becomes unable to link operators and therefore planning fails.

Failure to create a plan then brings the system to a halt, since it is not possible to collect new examples, and therefore it becomes impossible to refine the operators. We are currently developing a best-effort planning approach that always produces a plan even if it is known to be incorrect [14], so that the planning-learning cycle will keep on running in any case.

5. Case study: insurance claim processing

The blocks world scenario is an excellent testbench since it involves operators that undo one another. In real-world scenarios, the thesaurus and the operator definitions may get a lot more elaborate, hence more of a learning problem, but less of a planning problem, since most information-based business processes mostly produce information along the way and do not undo what previous activities have done. In this section we briefly present such an example, albeit a very simple one.

Figure 4 illustrates an overly simplified auto insurance claim process. The customer calls the insurance company saying her car is damaged, and the employee at the call center registers the claim. The customer is asked to leave the car at a specific garage, where an insurance expert will assess the damage. After that, two things will happen: the car is repaired and the policy rate of the customer increased. Finally, the claim is closed and filed for later reference.

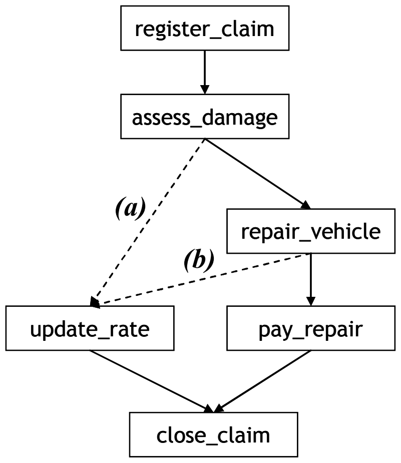

Figure 4. Auto insurance claim processing

There is a separate operator describing each activity. All of them accept two arguments: the claim and the vehicle. The thesaurus includes the following predicates: claim(C), vehicle(V), policy(P,V), damaged(V), assessed(C), repaired(V), payed(C), raised(P), bonus(P), open(C), closed(C). Given a single positive example for each operator, the algorithm took ten iterations until it came up with a feasible plan and, like in the blocks world scenario, this plan was found before all operators were accurately defined. The true operator definitions are listed below. The preconditions and effects that have been learned are shown underlined. Wrongfully learned operators are shown in square brackets.

register_claim(C,V)
   PRECOND: claim(C), vehicle(V), policy(P,V)
   REMLIST:
   ADDLIST: <u>damaged(V)</u>, <u>open(C)</u>

assess_damage(C,V)
   PRECOND: <u>damaged(V)</u>
   REMLIST:
   ADDLIST: <u>assessed(C)</u>

repair_vehicle(C,V)
   PRECOND: damaged(V), <u>assessed(C)</u>
   REMLIST: <u>damaged(V)</u>
   ADDLIST: <u>repaired(V)</u>

pay_repair(C,V)
   PRECOND: <u>repaired(V)</u>
   REMLIST:
   ADDLIST: <u>payed(C)</u>

update_rate(C,V)
   PRECOND: assessed(C), policy(P,V),
            [<u>repaired(V)</u>]

```
   REMLIST: bonus(P)
   ADDLIST: raised(P)

close_claim(C,V)
   PRECOND: open(C), payed(C), policy(P,V),
            raised(P)
   ADDLIST: closed(C)
   REMLIST: open(C)
```

Some differences between the true operators and the learned ones are worth mentioning:

- The preconditions of `register_claim` have not been learned at all. Since this operator always appeared in the beginning of the plan, where it should be, no negative examples have been generated.
- The preconditions of the operators `repair_vehicle` and `close_claim` are incomplete due to an insufficient number of negative examples.
- The preconditions of `update_rate` are actually wrong, containing `repaired(V)` instead of `assessed(C)` and `policy(P,V)`. This has happened because, after an initial negative example, the Planner was able to arrive at a feasible plan without having to correct those preconditions.

Note, however, the consequence of this last inaccuracy: as shown in figure 4, the planner actually arrives at plan version (b) instead of version (a). The precondition of `update_rate` is wrong, making this activity admissible only after `repair_vehicle`, which is not really intended to happen that way. This sort of mistake is hard to eradicate since it depends on the stepwise modifications that are introduced in the operators in each planning-learning cycle. Simply switching the order in which the operators are given to the Planner, for example, may solve such mistake in one operator and create a similar mistake in another.

6. Related work

The idea of using AI techniques to aid and enable workflow management is by no means new, and both planning and learning techniques have been applied in this domain. Here are some important developments:

▫ Beckstein and Klausner [15] aim at providing an adaptable workflow system that can easily handle exceptions and quickly adapt to changes. To this end the system consists of a least commitment planner (LCP) and a reason maintenance system (RMS). Run-time flexibility and adaptability is attained by the LCP that is used to create a new plan whenever a goal is set (process enactment) or re-plan when an environmental change warrants it (exception causes plan execution failure). The LCP is also used to facilitate workflow modeling by enabling interactive definition of the planning operators and testing plan generation. In this phase, its efforts are aided by the RMS which can for example inform the planner of open conditions or the required ordering of tasks. Besides the overall system architecture, this article also contributes with the enumeration of several characteristics required by the planner such as partial ordering and conditional planning.

▫ Just as in the case above, Moreno et al [16] also decide on the use of a partial order planner. However, the emphasis here is on supporting ad-hoc processes. Contingency planning is therefore used to deal with uncertainty as opposed to re-planning. Although contingency planning provides a means with which to increase system flexibility, it does suffer from a number of problems. First and foremost it does not ease the modeling of the planning operators because no rule checking is done. Second, the rule provided by the user must already identify possibly uncertain outcomes. Lastly, contingency planning itself is time-consuming and will not guarantee correct execution under all possible conditions (such as competing events and changes in background knowledge). Even so this article contributes with interesting ideas such as scheduling parallel activities (implicitly handles time and resource constraints), meta-modeling that deals with planning explicitly, and suggests that learning be used for process optimization.

▫ Madhusudan et al [17] present one of the few attempts to use both planning and learning within a single process management system. Unlike the two previous articles, however, the emphasis here is on supporting efficient workflow design as opposed to efficient run-time behaviour. The proposed system is based on three elements: (1) defining a process model using both standard workflow graph meta-model and predicate-based situation calculus (for the AI planner), (2) a similarity flooding algorithm used to retrieve model cases, and (3) a Hierarchical Task Network (HTN) planner that can be used to generate plans from composed and basic tasks. The system then attempts to generate, classify, index and retrieve case models. A successfully retrieved model may be used or adapted for a new problem, thus accelerating process modeling. In the event that no case is found, the planner may be used to generate a new plan by composing already existing processes or using basic tasks descriptions. Any new plans that are generated are classified, indexed and stored thereby allowing the system to learn. Its main contribution is that of attempting to solve the problems related to model storage, retrieval, reuse and assembly. It is one of the few research efforts that provides a means to facilitate the management of the complete process life-cycle.

▫ Jarvis et al [18] take a much broader view of the problem of adaptive workflow systems and try to identify how such a system may be implemented via the use of AI. Here, *adaptive* refers to the ability of the system to modify its behaviour according environmental changes and exceptions that may occur during plan execution. For this purpose, a set of five levels of adaptability are identified (domain, process, organization, agent and infrastructure) and enumerated together with their respective applicable AI technologies (rational maintenance, planning, capabilities marching, dynamic capability matching, multi-agent toolkits). Of all these technologies, we believe that planning and (dynamic) capability matching are essential for adaptiveness. It is worth noting, however, that workflow management systems have always provided some support (albeit limited) for capability matching. The system described interleaves planning (using the non-linear planner O-Plan) with execution and plan refinement. It also investigates plan patching and plan repair as a means to enhance flexibility and responsiveness.

The point of comparison is that, in terms of output, the approaches proposed by Beckstein and Klausner [15], Moreno et al [16], and Jarvis et al [18] produce plans that belong to the SNLP family [19] just like the partial-order plans obtained by POP. An exception is Madhusudan et al [17] who make use of hierarchical task networks.

Still, although the references above demonstrate that the use of planning as a tool to aid workflow management is not new, it is important to note that previous efforts tend to focus either on the build-time or alternatively on the run-time phase. In addition, these articles show that even though planning and learning can prove beneficial, none has attempted to combine learning and planning to foster flexibility in workflow management systems.

7. Conclusion

This paper proposes a new life cycle for workflow management based on the continuous interplay between learning and planning. The approach is based on learning business activities as planning operators and feeding them to a planner that generates the process model. Besides reducing the modeling effort, it supports process adaptation by allowing the basic building operators to be refined iteratively according to user actions. Another point of flexibility is that process models can be generated on-demand, given the intended goal state.

Far from proving that the proposed approach really works, the paper describes our experiments in simple scenarios, which show as much potential as hindrances that deserve further discussion and investigation.

8. References

[1] D. Hollingsworth, "The Workflow Reference Model", Document Number TC00-1003, WfMC, 1995

[2] M. Voorhoeve, W. Van der Aalst, "Ad-hoc workflow: problems and solutions", 8th International Workshop on Database and Expert Systems Applications (DEXA '97), September 01 - 02, Toulouse, France, 1997

[3] C. Hagen, G. Alonso, "Exception Handling in Workflow Management Systems", *IEEE Transactions on Software Engineering*, 26(10), October 2000

[4] W. Sadiq, O. Marjanovic, M. E. Orlowska, "Managing change and time in dynamic workflow processes", *Intl. Journal of Cooperative Information Systems*, 9(1-2), pp.93-116, 2000

[5] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, M. Teschke, "A comprehensive approach to flexibility in workflow management systems." Technical report TR-16-1998-6, University of Erlangen-Nuremberg, Erlangen, Germany, 1998

[6] H. Garfinkel, *Studies in Ethnomethodology*, Prentice-Hall, Englewood Cliffs, NY, 1967

[7] D. Stenmark, "Leveraging Tacit Organisational Knowledge", *Journal of Management Information Systems*, 17(3), pp.9-24, Winter 2000-2001

[8] N. Lavrac, S. Dzeroski, *Inductive Logic Programming: Techniques and Applications*, Ellis Harwood, New York, 1994

[9] J. Penberthy, D. Weld, "UCPOP: A sound, complete, partial order planner for ADL", Proceedings of the Third International Conference on Knowledge Representation and Reasoning, Morgan Kaufmann, 1992

[10] M. Ghallab, D. Nau, P. Traverso, *Automated Planning: theory and practice*, Morgan Kaufmann Publishers, 2004

[11] B. Drabble, J. Dalton, A. Tate, "Repairing Plans on the Fly", Working Notes of the First International Workshop on Planning and Scheduling for Space, Oxnard, CA, 1997

[12] X. Nguyen, S. Kambhampati, "Reviving Partial Order Planning", Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001), 2001

[13] H. Ferreira, D. Ferreira, "Towards an Integrated Life-Cycle for Business Process Management based on Learning and Planning", Technical Report, INESC Porto, November 2004

[14] A. Garland, N. Lesh, “Plan evaluation with incomplete action descriptions”, Proc. Eighteenth National Conference on Artificial Intelligence, Edmonton, Alberta, Canada, 2002

[15] C. Beckstein, J. Klausner, “A Meta Level Architecture for Workflow Management”, *Transactions of the SDPS*, 3(1), pp.15-26, March 1999

[16] M. Moreno, P. Kearney, D. Meziat, “A Case Study: Using Workflow and AI Planners”, 19th Workshop of the UK Planning and Scheduling (PLANSIG2000), Milton Keynes (U.K.), 2000

[17] T. Madhusudan, J. Zhao, B. Marshall, “A case-based reasoning framework for workflow model management”, *Data & Knowledge Engineering*, 50(1), pp. 87-115, 2004

[18] P. Jarvis, J. Moore, J. Stader, A. Macintosh, A. Casson-du-Mont, P. Chung, “Exploiting AI Technologies to Realise Adaptive Workflow Systems”, *Agent-Based Systems in the Business Context: Papers from the AAAI Workshop*, Technical Report WS-99-02, AAAI Press, 1999

[19] D. McAllester, D. Rosenblitt, “Systematic Nonlinear Planning”, Proc. Ninth National Conference on Artificial Intelligence (AAAI-91), AAAI Press, 1991