AN INTEGRATED LIFE CYCLE FOR WORKFLOW MANAGEMENT BASED ON LEARNING AND PLANNING

HUGO M. FERREIRA
INESC Porto
Campus da FEUP, Rua Dr. Roberto Frias 378, 4200-465 Porto, Portugal
hmf@inescporto.pt

DIOGO R. FERREIRA
IST, Technical University of Lisbon
Campus do Taguspark, Avenida Prof. Dr. Cavaco Silva, 2780-990 Porto Salvo, Portugal
diogo.ferreira@tagus.ist.utl.pt

The ability to describe business processes as executable models has always been one of the fundamental premises of workflow management. Yet, the tacit nature of human knowledge is often an obstacle to eliciting accurate process models. On the other hand, the result of process modeling is a static plan of action, which is difficult to adapt to changing procedures or to different business goals. In this article we attempt to address these problems by approaching workflow management with a combination of learning and planning techniques. Assuming that processes cannot be fully described at build-time, we make use of learning techniques, namely Inductive Logic Programming (ILP), in order to discover workflow activities and to describe them 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. Following a discussion of related work, the paper concludes by presenting the main challenges that remain to be solved.

Keywords: Workflow; BPM; Inductive Logic Programming (ILP); learning; planning.

1. Introduction

Business Process Management (BPM) solutions, particularly workflow management systems, have been around since the early nineties, yet they remain especially useful in a limited range of applications where business processes can be described with relative ease. These applications include mainly production and administrative processes, rather than collaborative, unstructured processes. Ultimately, this is because workflow management defined 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.

The lack of a closed-loop life cycle, the requirement of having processes fully specified *a priori*, and the fact that these processes become rigid plans of action have been the source of several challenges. For a long time, workflow management has been facing issues such as the coordination of 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 broadening the range of applicability of workflow management systems by providing them with more flexibility[5].

Improving the flexibility of workflow management is being attempted via research in modeling techniques and execution mechanisms, but these techniques still hold on to the following fundamental assumptions:

- The first major assumption is that processes can be described by an explicit and accurate model. But in scenarios where processes are to a large extent unclear and/or unstructured, process modeling cannot be completed prior to execution (due to lack of domain knowledge, complexity of task combination and the difficulty in transmitting domain specific knowledge). As processes are executed and knowledge is acquired via experience, it will be necessary to go back to the process definitions and correct them according to work practices.
- The second major assumption is that processes, after having been modeled, can be repeatedly instantiated and executed in a predictable and controlled manner. However, even for structured processes the combination and sequence of tasks may vary from instance to instance (due to changes in the execution context such as user preferences, or changes in the environment such as exceptions and changes in business rules). In such cases, workflow processes should be adapted accordingly (e.g. by adding, removing or generating an alternative sequence of activities).

In this article we present a different approach towards workflow management - one that combines learning and planning techniques in order to overcome the challenges when the above assumptions do not hold. We will therefore assume that processes, in general, cannot be accurately and completely modeled by human effort, so learning techniques will be fundamental in order to capture business activities as a set of rules that are inferred from user actions. We will also assume that process instances routinely require modifications in order to adapt to changes in the environment. Planning techniques will be essential in order to produce an initial process model as a sequence of actions that comply with activity rules and achieve the intended goal. Process adaptation will be achieved via simple re-planning.

2. Learning workflow activities

It has since long been realized 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. For example, in Ref. 7 the author argues that formal job descriptions are seldom enough to account for the actions that an employee performs during a working day, and that much of our daily activity is governed by professional interests that are tacit in nature. Since much of organizational work relies on tacit knowledge, we should not rely on the assumption that people will be able to describe their work exactly as they do it.

If the purpose is to develop a process model then rather than relying exclusively in interviews, it may be necessary to actually observe work practices, create a process definition, and then check with the users to see if it is correct. Once the process is released for execution, possible exceptions or changes in procedures will throw the process modeler back to the starting point, requiring her to gather further feedback from the users in order to come up with a more accurate or refined process model. Process management then becomes a closed-loop endeavor, where process models both guide execution and are subject to adjustments as a result of user behavior at run-time.

The need for such a closed-loop life cycle is even more pressing in collaborative environments[a] where processes cannot be anticipated, and thus cannot be studied or modeled as a whole. Instead, what can be done is to identify and study a set of individual activities, and then try to understand the ways in which these activities can precede or follow each other. For this purpose, it is useful to describe an activity as a single step that consumes input data and produces output data. The set of available data before and the set of available data after performing an activity are two different *world states*. In general, an activity can only be performed in a given world state if that state satisfies the *preconditions* of that activity. Each activity has also a set of *effects* that change the current state into a new world state. Preconditions, effects, and world states can all be described using first-order logic.

2.1. *Defining activities as operators*

Let us consider that a process model Π is a partially-ordered set of activities where each activity α_i 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 $P_i \subseteq S$ (that is, if P_i is satisfied in S) where P_i are the preconditions of the activity. P_i is expressed as a set of propositions (predicates) and S is expressed as a set of ground unit clauses. 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 propositions R_i that α_i removes from S_i, and another set of propositions A_i that α_i adds to S_i so that the overall result is 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.

In section 5 we describe an auto-insurance claim process that uses such kind of operators. 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

[a]For recent developments towards supporting collaborative processes, see for example Ref. 8, Ref. 9 and Ref. 10.

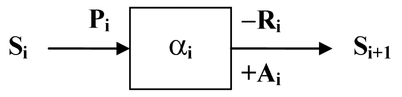

Fig. 1. Anatomy of a workflow activity

damage has already been assessed for the claim (C). The effect of this activity is that the vehicle is not damaged anymore because it has been repaired. This can be described with the following operator:

$$\begin{aligned}
&\mathit{repair\_vehicle}(C,V) \leftarrow \\
&\quad \mathit{PRECOND}: \ \mathit{damaged}(V),\ \mathit{assessed}(C) \\
&\quad \mathit{REMLIST}: \ \mathit{damaged}(V) \\
&\quad \mathit{ADDLIST}: \ \mathit{repaired}(V)
\end{aligned}$$

The result of performing this activity is that it produces a new state by removing the literal *damaged*(V) and adding the literal *repaired*(V) to the previous state. Here we assume that the absence of the literal *damaged*(V) means that the car is not damaged, but not necessarily that it has been repaired. This condition must be expressed by the presence of the literal *repaired*(V). Choosing the predicates that are used to describe each activity falls into the general problem of knowledge representation. The important issue here is that these predicates must be consistent across activities, so that the effects of one operator can match the preconditions of another, allowing the activities to be connected into a sequence of actions.

2.2. *Learning preconditions and effects*

Given that much of organizational work relies on tacit knowledge, we will assume that users won't be able to accurately describe P_i, R_i and A_i for the activities they perform, and our objective is to identify them. However, users will be able to say, given a state S_i, whether they can or cannot perform a certain activity. When an action can take place in state S_i the operator and the world states (both initial state S_i and final state S_{i+1}) constitute a positive example. When an action is not possible in state S_i then the correct preconditions for that activity have not yet been identified. In this case the operator and the initial state S_i constitute a negative example.

Provided we can collect a proper set of positive and negative examples, it is possible to infer P_i, R_i and A_i using standard machine learning techniques. Inductive Logic Programming (ILP)[11] seems especially useful for this purpose. Basically, ILP searches through different combinations of literals until it finds the correct rule for a given predicate (for example, *repair_vehicle*(C,V) can only be done if the literals *damaged*(V) and *assessed*(C) are both present in the current state). The search can be performed in a top-down fashion from a general rule to a more specialized rule by inserting additional predicates, or in a bottom-up fashion from the most specialized rule to a more general rule by removing predicates, as illustrated in figure 2.

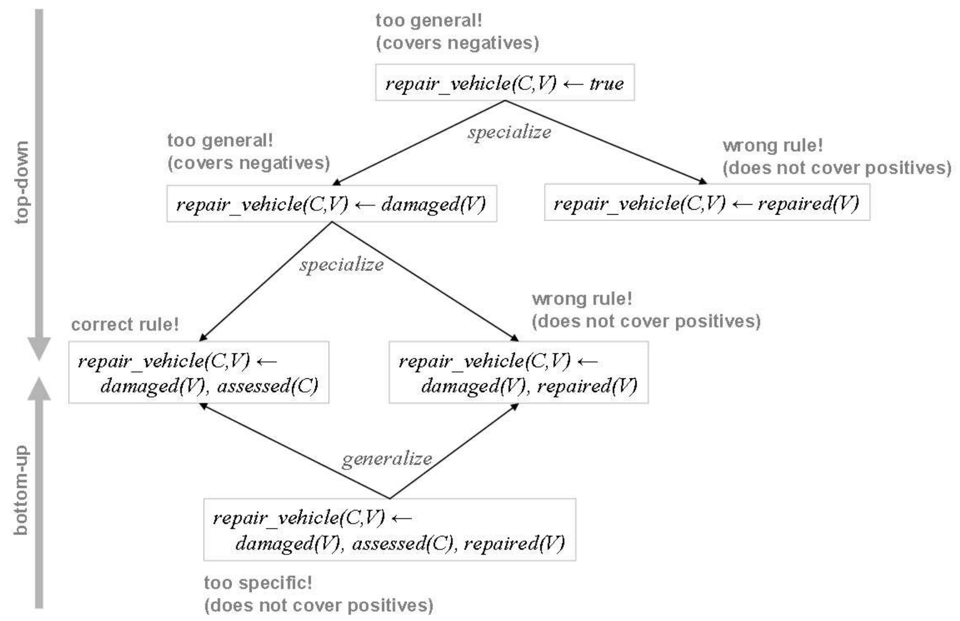
top-down
bottom-up
too general!
(covers negatives)
repair_vehicle(C,V) ← true
specialize
too general!
(covers negatives)
wrong rule!
(does not cover positives)
repair_vehicle(C,V) ← damaged(V)
repair_vehicle(C,V) ← repaired(V)
specialize
wrong rule!
(does not cover positives)
correct rule!
repair_vehicle(C,V) ← damaged(V), assessed(C)
repair_vehicle(C,V) ← damaged(V), repaired(V)
generalize
repair_vehicle(C,V) ← damaged(V), assessed(C), repaired(V)
too specific!
(does not cover positives)

Fig. 2. Top-down vs. bottom-up search for precondition rule

There are two kinds of data that ILP requires as input: *background knowledge* and *training examples*. Background knowledge is a set of literals that define the predicates to be used and establish a set of basic truths, which can be regard as statements about the current world state. Training examples comprise a set of statements where the intended rule must give either a positive or a negative result. 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 literals that remain unchanged when α_i is performed, and $S_i \backslash \{S_i \cap S_{i+1}\}$ represents the set of literals that were present in the world state S_i, but are not present in S_{i+1}.
- 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. $S_{i+1} \backslash \{S_i \cap S_{i+1}\}$ represents the set of literals that are present in S_{i+1}, but were not present in S_i.

It should be noted that there are only positive examples available for learning the effects of an activity (R_i and A_i) since there is no output state for an activity that could not be executed. Therefore, R_i and A_i must be learned without the aid of negative examples which means, according to figure 2, that bottom-up search should be applied. Hence these rules are 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, so there is no interference in the world state from other activities running at the same time.

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. In general, users will have to be assisted, in 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 set of positive and negative examples, which will be used to improve those descriptions by learning.

Once the operator definitions have been found, then the plan for reaching a goal state from an initial state can be created, again using standard AI techniques. Partial-order planning (POP)[12] seems particularly useful for this purpose. An obvious advantage of partial-order planning is that it generates plans with a maximum degree of parallelism, which is essential for long-running workflow processes, as opposed to total-order planning, which creates linear, single-thread plans. But there are several other reasons for choosing POP:

- it generates plans with a higher degree of flexibility[13];
- the planning information is easily understood by humans and allows interactive plan analysis and repair[13,14];
- it facilitates the analysis and repair of failed plans (see for example Ref. 14);
- it facilitates the handling of domains with durative actions and temporal and resource constraints[15];
- it allows easier integration with execution, information gathering and scheduling architectures[15].

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 at 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 more efficient ways, which is reminiscent of process re-engineering[16] but without an overwhelming analysis effort. If the operators are properly defined, then planning will provide the best plan in terms of number of steps that are required to achieve a given goal.

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 accurately defined to begin with. And learning won't help either, given the limited amount of initial examples that users are able to provide. Fortunately, as we will show ahead, it is not necessary to have a completely accurate description of the operators in order to come up with a correct plan.

Despite the operators being inaccurate on a first stage, if we 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, more correct plan.

This life cycle is illustrated in figure 3. The user selects a goal from a set of predefined goals specified using a common thesaurus. This thesaurus, which is developed by the users themselves, defines the predicates that are used to describe world states, activity preconditions, and effects. Then, with the available operators, the planner generates a plan in order to satisfy the given goal state. Activities may be assigned to different users, which manage their tasks through their own task lists, as in regular workflow systems. The difference here is that 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.

This life cycle has been implemented in a prototype system with the user replaced by a simulator with knowledge of the true operators. The simulator does not reveal the operators, but it says whether a given operator can be applied in a given state or not, by means of a simple boolean response. The algorithm is shown in figure 4. It begins by learning the operators form the limited set of initial examples, and then attempts to create a plan for the given goal state. If it is not possible to create a plan, probably because one or more operators still lack the appropriate

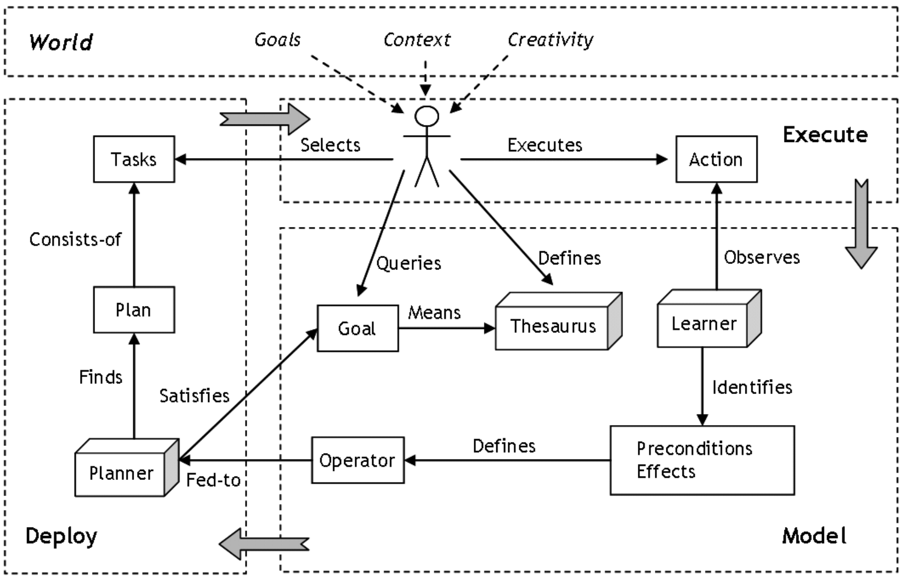

Fig. 3. Learning and planning life cycle[17]

effects, it is necessary to collect further examples in order to refine them. This is done by generating random world states and asking the simulator whether each operator is applicable in those states[b].

It has been observed that the creation of an initial plan, however incorrect, is not difficult, mainly because in general the effects of an operator can be learned with relative ease from just a few examples. The same does not apply to the preconditions, which require more examples to be learned. Even so, if the operators have most of their effects in place but their preconditions are missing, it is actually easier for the planner to find a solution, since there are fewer preconditions that need to be satisfied.

Returning to figure 4, once a plan is created the system attempts to execute it. This is done by asking the simulator whether it can execute each activity in the plan. The advantage of having the simulator in the prototype system is that it provides a response quickly and automatically. If all activities are successful, then a plan for achieving the given goal state has been found. If not, each attempt to execute any activity results in a positive or negative example which is used to further refine the operators and to generate a new plan. The system repeats this procedure until a valid plan is found.

[b]To guarantee that the randomly-generated states are actually valid states, the simulator generates each of them by applying a random set of true operators to the given initial state, which produces a new state. In the blocks world scenario ahead, this is equivalent to "shuffling" the blocks in order to obtain a random state.

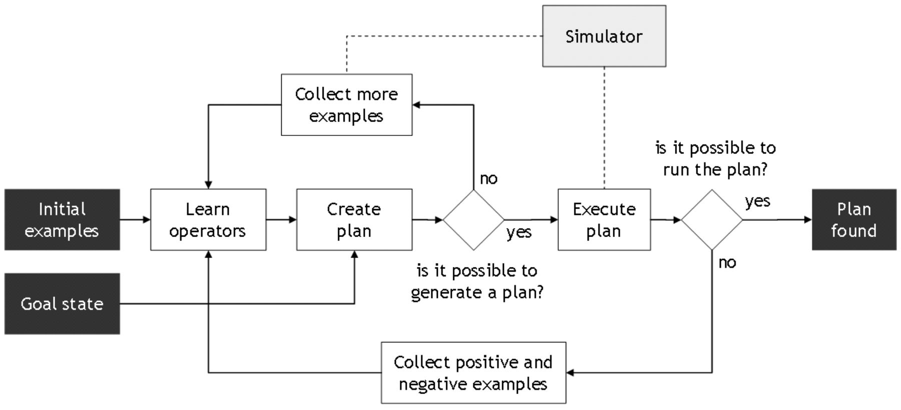

Fig. 4. Learning and planning algorithm

Different versions of this algorithm have been implemented in Prolog and Java. A first Prolog prototype made use of only top-down search in learning and treated each operator as a whole, attempting to learn both the preconditions and effects simultaneously. Since there are no negative examples for operator effects, the system was limited to positive examples. Failure to execute an activity would make the algorithm select another solution from the ILP tree. In the Java prototype (which initially was meant to be an alternative implementation of the same algorithm) the preconditions, the add-list and the remove-list are now all learned separately, which allows the system to make use of both positive and negative examples to learn the preconditions, as explained in section 2.2, and to take advantage of both top-down and bottom-up search. This second approach was shown to produce more accurate operators with less iterations.

5. Evaluating system behavior

The first approach towards testing the prototype system was to evaluate its behavior in typical planning problems. One such problem is the blocks world scenario, illustrated in figure 5. The given goal is to go from state S_I to state S_O. For this purpose, we have considered the following set of 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, or if X is not on top of Y.
- *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, or if X is not on top of Y.
- *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.

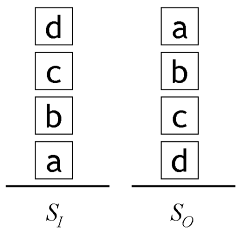

Fig. 5. Blocks world scenario

The thesaurus included only the two 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 (empty in a first stage).

After five 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 found to be necessary 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.
- All effects have been correctly learned.

5.1. *When planning fails*

We have conducted further experiments, in which the initial examples did not allow the learner to find out the exact operator effects immediately. In this scenario, it was impossible to create a plan without providing the system with more examples. This is due to the fact that the preconditions were being refined (becoming increasingly demanding) but the effects were still incomplete, so the planner became unable to satisfy the preconditions, and therefore to link operators, thus planning failed. Without collecting more initial examples, the system would come to a halt, since if it is not possible to create a plan, there are no examples from plan execution, and therefore it becomes impossible to refine the operators.

In order to avoid such difficulty, we have implemented a best-effort planner that always creates a plan, even though the plan may be incorrect. The implementation is based on Ref. 18, in which the authors describe an algorithm for producing plans with a measurable risk of failure, based on indicators such as the number of unsatisfied preconditions or the number of possible conflicts between activities. The best plan is the one with the lowest risk.

In order to make the best-effort planner produce several candidate plans, we collect several candidate rules for each operator during the learning phase. We allow the learner to search the ILP tree further in order to find multiple solutions instead of just one, and we create a different plan for each of the candidate operators. The plan with the lowest risk is the one that is chosen for execution.

Unfortunately, this approach has hardly been successful, as the best plan is not necessarily the one which corresponds to the best candidate operator. The result is that the correct operator often ends up not being considered in favor of other candidates, and the whole system tends to evolve towards increasingly specialized operators, moving further and further away from the correct solution. We currently use the best-effort planner as an effective means to collect more examples by keeping the system running, rather than by generating random world states as described in section 4.

Another reason for failing to create a plan is the existence of operators that undo the effects of one another, thereby allowing the repetition of states. For example, in the blocks world scenario it is evident that one can perform actions that make the system go back to a previous state. During planning this may happen as well, as the planner tries several combinations of operators in order to satisfy the goal state. Since POP is a systematic algorithm, it follows the same solution procedure when given the same goal, which can lead to a situation of infinite looping or recursion[19]. The simplest way to avoid this problem was to resort to iterative deepening, allowing the planner to consider plans with n actions only when it has already considered all possibilities with $(n-1)$ actions. Fortunately, as the following case study suggests, real-world business processes are less likely to involve conflicting operators.

5.2. *Case study: insurance claim processing*

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 are likely to produce information along the way without completely undoing what previous activities have done. This means that, when planning, there will be fewer conflicts between operators. In this section we briefly present such an example, albeit a very simple one.

Figure 6(a) 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.

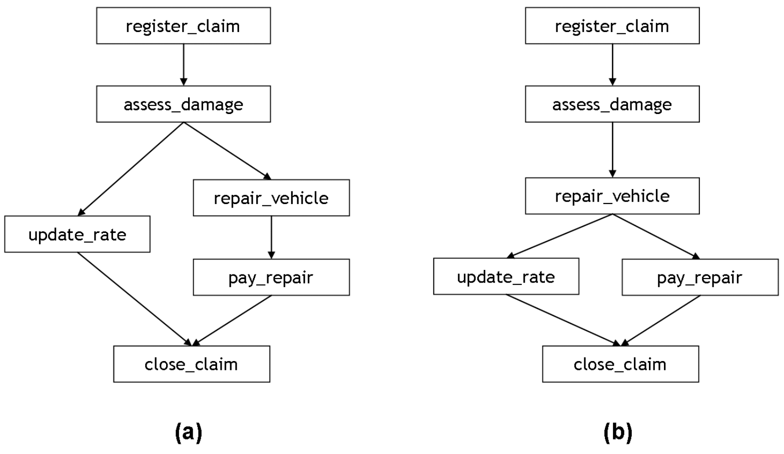

Fig. 6. True process (a) and solution found by the system (b)

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. Incorrectly learned literals are shown in square brackets.

$register\_claim(C, V) \leftarrow$
$\quad PRECOND : claim(C), vehicle(V), policy(P, V)$
$\quad REMLIST :$
$\quad ADDLIST : \underline{damaged(V)}, \underline{open(C)}$

$assess\_damage(C, V) \leftarrow$
$\quad PRECOND : \underline{damaged(V)}$
$\quad REMLIST :$
$\quad ADDLIST : \underline{assessed(C)}$

$repair\_vehicle(C, V) \leftarrow$
$\quad PRECOND : damaged(V), \underline{assessed(C)}$
$\quad REMLIST : \underline{damaged(V)}$
$\quad ADDLIST : \underline{repaired(V)}$

$pay\_repair(C, V) \leftarrow$
$\quad PRECOND : \underline{repaired(V)}$
$\quad REMLIST :$

$ADDLIST$: $\underline{payed(C)}$

$update\_rate(C, V) \;\leftarrow$
  $PRECOND$: $assessed(C), policy(P, V), [\underline{repaired(V)}]$
  $REMLIST$: $bonus(P)$
  $ADDLIST$: $\underline{raised(P)}$

$close\_claim(C, V) \;\leftarrow$
  $PRECOND$: $open(C)$, $\underline{payed(C)}$, $policy(P, V)$, $\underline{raised(P)}$
  $ADDLIST$: $\underline{closed(C)}$
  $REMLIST$: $\underline{open(C)}$

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 as ILP traverses the search tree, it searches for rules with one, two, three literals, and so on. What happened here was that after an initial negative example, the system was able to create a feasible plan without having to go back and correct those preconditions.
- The remove-list of *update_rate* has not been learned because the initial example for this operator was taken from a case where the customer did not have a bonus already. Since this predicate does not appear in any other operator, it was irrelevant for planning purposes, and no run-time examples have been collected that would suggest that it would be necessary.

Due to the third inaccuracy described above - the mistake in the preconditions of *update_rate* - the planner actually arrives at the plan shown in figure 6(b) instead the one shown in 6(a). The presence of the wrong precondition $repaired(V)$ in $update\_rate(C, V)$ makes this activity admissible only after $repair\_vehicle(C, V)$, which is not really the true sequence.

The fact that some operators are more accurate than others depends on the stepwise refinement in each planning-learning cycle. In the beginning when all the operators are poorly defined (especially concerning preconditions) the planner is able to create plans using just some of the available operators. For example, in the blocks world scenario the planner found that it needed the operator *movebt* only in the last iteration. The operators that get chosen in a planning cycle are the ones that will collect run-time examples, so the more they are chosen, the more examples about these operators will be known.

The operator *update_rate* from above had one initial example and it collected just one additional negative example at run-time. This example was enough to make the learner conclude that some precondition was missing, but not enough to make the learner identify the true preconditions[c]. Simply switching the order in which the operators are given to the planner will make the planner use different operators, therefore resulting in a different set of run-time examples. This may solve such mistake in one operator and create a similar mistake in another. This problem can be mitigated by providing a larger set of initial examples.

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 we briefly discuss some important developments:

- In Ref. 20 the authors 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, in Ref. 21 the authors 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 could be used for process optimization.

[c]In practice, the "true" preconditions are unknown!

- In Ref. 22 the authors 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 aims at supporting the complete workflow management life-cycle.

- In Ref. 23 the authors are motivated by the fact that process modeling is difficult in practice, and they present an induction algorithm for the purpose of finding a process model from a given set of previously performed activities. Information about the performed activities is obtained from a system log. The algorithm creates a stochastic activity graph (SAG) comprising a set of activity nodes and transition probabilities, and then transforms this graph into a workflow model. The resulting model is compared to the original process model which produced the log information, showing that it is possible to infer accurate process models from run-time information. This kind of approach is referred to as *workflow mining*[24]. Workflow mining shares a similar purpose with our own approach in that it attempts to create a feedback loop to adapt workflow models according to work practices. It should be noted, however, that workflow mining aims at discovering activity sequences, but not why they happen in that sequence, or whether they could be performed in another sequence. In addition, workflow mining requires the availability of system logs, which greatly simplifies the problem of eliciting rules since it does not have to rely on world state information and, hence, is oblivious to the problem of knowledge representation.

- In Ref. 25 the authors take a much broader view of the problem of adaptive workflow systems and try to identify how such a system may be implemented with the use of AI techniques. Here, *adaptive* refers to the ability of the system to modify its behavior according to 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 matching, dynamic capability matching, multi-agent toolkits). Of all these technologies, we believe that planning and (dynamic) capability matching are the most important 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.

In terms of planning, the point of comparison is that the approaches proposed in Ref. 20, Ref. 21, and Ref. 25 use planning algorithms that belong to the SNLP family[26]. While choosing POP, we also opted for such kind of planner. An exception is Ref. 22 in which the authors 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 on the run-time phase.

In terms of learning, even though the above sources suggest that both planning and learning can prove beneficial, none have attempted to combine learning and planning in order to capture and manage business processes across build-time and run-time phases. It should be noted, however, that the combination of learning and planning has been an active theme of research in AI, although not in connection with workflow management. For example, in Ref. 27 the authors have explored the combination of inductive learning and planning with the purpose of accelerating plan generation by learning search rules from experience. In Ref. 28, the author makes use of learning techniques in order to infer action strategies within a given planning domain.

In general, the application of AI techniques within the scope of workflow management has been focusing either on the learning algorithm or in the use of planning to generate workflows. Although the use of each technique is in itself interesting, it is in the combination of the two techniques that lies the potential to automate the full life-cycle of workflow process management, which includes process generation, execution, and re-planning according to run-time data.

Besides the AI-related work, research in workflow management has also recently recognized the advantages of identifying processes from execution logs[24], particularly for ad-hoc processes[29]. The main theme in such work - referred to as workflow mining, as mentioned before - is that of aiding the identification and analysis of business processes. In other words, this work assumes that workflow already exists, that the execution of such processes generate logs, and that the data contained in these logs are amenable to offline analysis and processing. Another premise is that the analysis of workflow logs results in a graph depicting the relationships between tasks within a specific process, which can be analysed for the purpose of re-engineering such processes. In any case, these methods are still centered in the idea that one can completely and correctly model a process and that process modeling and execution are fundamentally two separate phases. On the other hand, such work clearly shows that research in the area of workflow management is now tackling the problem of eliciting and supporting unstructured processes.

7. Remaining challenges

Though the combination of learning and planning seems promising, there are several challenges that must be addressed before a workflow system based on those features can be built. Here we briefly discuss some of these challenges to be addressed in future work:

- *Efficiency of the planning algorithm.* In our experiments, the system spends most of its time in planning attempts. The complexity of a planning problem heavily depends on interference between operators; for example, operators that undo what other operators do make it harder to find a planning solution, even though this is less common in business scenarios. On the other hand, as the number of activities increases so does the complexity of the planning problem, and so does the time it takes to come up with an accurate description of the operators via planning and learning. But the ability to produce on demand plans requires the planner to take no more than a reasonable amount of time computing the sequence of required actions.
- *Planning with sensing actions.* One of the must-have features of workflow systems is the ability to make branching decisions based on run-time data, i.e., the process model may produce a different behavior based on conditions that can only be evaluated at run-time. This is currently not supported by our system, which produces process models as simple partial-order plans. Fortunately, there are ways to provide planning with such features: one way is to make use of *sensing actions*[30] that gather (and may also change) information from the world at run-time. The use of sensing actions is fundamental in order to support basic workflow patterns such as *exclusive-choice* or *multiple-choice*[31]. Other patterns, such as parallel-split and synchronization, are already supported by partial-order planning.
- *Learning with inconsistent examples.* Learning relies on examples that are collected by observation of user activities. Ideally, all of these examples are consistent. In practice, however, there are several reasons that could make a set of collected examples contain inconsistencies. One reason is human error: a user may perform an activity (an event that will be collected as a positive example) only to find later in the process that the activity did not produce the desired effects. In this case the learning algorithm will make use of a wrong positive example when refining the operator. Another source of inconsistent examples is process evolution; in this case, the set of examples that is provided to the learning algorithm contains older and newer examples which, separately, would lead to different operator definitions and, together, may render learning impossible. The problem of learning inductively from inconsistent examples has already been studied, and it is known as *noise handling*[11]. However, the two sources of inconsistency must

be treated in different ways. While bad examples due to human error should be basically disregarded, the new examples, due to changes in business practices, should invalidate older ones.

- *Multiple agents acting concurrently.* The ideal way to collect examples is to observe user actions in a closed world without any source of interference. In practice, though, several users may be performing tasks concurrently, so the difference between two world states is the sum of the effects of all activities that have been executed in a given period. The problem is now how to identify the effects of the activity being observed, separating them from the effects of other activities taking place at the same time. Over time, by observing a sufficient number of examples, it is possible to identify the effects of each operator. But this introduces a degree of uncertainty whose impact on the learning and planning cycle still remains unclear.
- *Mixed-initiative planning.* Collecting examples by capturing world states may lead to a surplus of information when only a small amount of data is actually relevant. Having to deal with unnecessarily large amounts of data will make planning unnecessarily difficult. Users can be especially helpful in providing valuable hints about how to carry out certain parts of the process, hence making it easier to find the overall planning solution. Facilitating planning by means of user input is known as *mixed-initiative planning*[32]. Mixed-initiative planning could be useful not only to generate plans but also to collect further examples, since the plan the user provides is a source of additional examples of operator usage. Both of these factors could significantly accelerate the convergence of the whole system towards accurate operator descriptions.

Besides these research issues, there are at least two practical issues to be addressed before such a system can become operational:

- *Capturing world states.* It is assumed that users perform their activities over a common information infrastructure comprising one or more IT systems. A world state is a snapshot of the existing data in those systems at a certain point in time. The simplest scenario would be that of a fully-integrated system such as an Enterprise Resource Planning (ERP) solution where users work over a single common database, so the world state could be obtained from a single source. In practice, however, an information infrastructure often comprises several different systems, so it is more challenging to keep track of all the effects of any given activity. Capturing the world state then requires collecting and integrating data coming from different systems. In either case, a world state may comprise an overwhelming amount of data that cannot simply be handled directly. Some sort of mechanism must be devised in order to be able to collect and handle world state information in an efficient way.
- *Making use of common semantics.* It is central to the proposed approach that

all users make use of the same semantics when referring to any concept in their domain, so that the effects of one activity can match the preconditions of the following ones. In addition, goals must be expressed in a way that is commonly understood by all users, and world states must be described in a consistent way throughout all activities being performed in the domain. This requires the use of a common ontology throughout the whole system[d]. 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.

8. 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 by allowing the basic building operators to be refined iteratively according to user actions, it is able to generate on demand process models, given the intended goal state. Although the proposed plan may not be feasible on a first stage, the system will produce increasingly accurate plans as it collects more and more examples.

A fundamental result obtained from experiments with the prototype system is that it is possible to produce fully accurate process models even though the activities (i.e. the operators) may not be accurately described. This is evident in the simple blocks world scenario presented above, which confirms the suspicion that the first major assumption of workflow management (having an accurate model to begin with) is not actually an indispensable requirement. Flexibility in workflow management may be achieved with a combination of learning and planning, which shows as much potential as challenges that deserve further discussion and investigation.

Acknowledgements

We would like to thank J. J. Pinto Ferreira and Rui Camacho (both at the Faculty of Engineering, University of Porto) for their support while developing this work. We would also like to thank the anonymous reviewers for their valuable comments and suggestions for improvements.

References

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

[d]For the problem of devising ontologies in workflow domains, see for example Ref. 33 and Ref. 10.

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. T. Hodel-Widmer, K. Dittrich, *Concept and prototype of a collaborative business process environment for document processing*, Data & Knowledge Engineering, vol.52, no.1, pp.61-120, 2005
9. S. Dustdar, *Caramba - A Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams*, Distributed and Parallel Databases, vol.15, no.1, pp.45-66, 2004
10. P. Chung, L. Cheung, J. Stader, P. Jarvis, J. Moore, A. Macintosh, *Knowledge-based process management - an approach to handling adaptive workflow*, Knowledge-Based Systems, vol.16, pp.149-160, 2003
11. N. Lavrac, S. Dzeroski, *Inductive Logic Programming: Techniques and Applications*, Ellis Harwood, New York, 1994
12. 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
13. M. Ghallab, D. Nau, P. Traverso, *Automated Planning: theory and practice*, Morgan Kaufmann Publishers, 2004
14. 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
15. X. Nguyen, S. Kambhampati, *Reviving Partial Order Planning*, Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001), 2001
16. P. O'Neill, A. Sohal, *Business Process Reengineering: A review of recent literature*, Technovation, vol. 19, no. 9, pp. 571-581, 1999
17. H. Ferreira, D. Ferreira, *Towards an Integrated Life-Cycle for Business Process Management based on Learning and Planning*, Technical Report, INESC Porto, November 2004
18. A. Garland, N. Lesh, *Plan evaluation with incomplete action descriptions*, Proc. Eighteenth National Conference on Artificial Intelligence, Edmonton, Alberta, Canada, 2002
19. D. Smith, M. Peot, *Suspending Recursion in Causal Link Planning*, Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems, pp.182-190, AAAI Press, 1996
20. C. Beckstein, J. Klausner, *A Meta Level Architecture for Workflow Management*, Transactions of the SDPS, 3(1), pp.15-26, March 1999
21. 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

22. T. Madhusudan, J. Zhao, B. Marshall, *A case-based reasoning framework for workflow model management*, Data & Knowledge Engineering, 50(1), pp. 87-115, 2004
23. J. Herbst, D. Karagiannis, *Workflow mining with InWoLvE*, Computers in Industry, vol.53, no.3, pp.245-264, 2004
24. W. van der Aalst, B van Dongen, J. Herbst, L. Maruster, G. Schimm, A. Weijters, *Workflow mining: A survey of issues and approaches*, Data & Knowledge Engineering, vol.47, no.2, pp. 237-267, 2003
25. 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
26. D. McAllester, D. Rosenblitt, *Systematic Nonlinear Planning*, Proc. Ninth National Conference on Artificial Intelligence (AAAI-91), AAAI Press, 1991
27. C. Leckie, I. Zukerman, *Inductive learning of search control rules for planning*, Artificial Intelligence, vol.101, no.1-2, pp.63-98, 1998
28. R. Khardon, *Learning action strategies for planning domains*, Artificial Intelligence, vol.113, no.1-2, pp.125-148, 1999
29. S. Dustdar, T. Hoffmann, W. van der Aalst, *Mining of ad-hoc Business Processes with TeamLog*, Data & Knowledge Engineering, vol.55, no.2, pp.129-158, 2005
30. O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, M. Williamson, *An Approach to Planning with Incomplete Information*, Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning, October 1992
31. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A.P. Barros, *Workflow Patterns*, Distributed and Parallel Databases, vol.14, no.1, pp.5-51, 2003
32. G. Ferguson, J. Allen, B. Miller, *TRAINS-95: Towards a mixed-initiative planning assistant*, Proc. of the Third International Conference on AI Planning Systems, AAAI Press, 1996
33. B. Dellen, F. Maurer, G. Pews, *Knowledge-based techniques to increase the flexibility of workflow management*, Data & Knowledge Engineering, vol.23, no.3, pp.269-295, 1997