Interactive Multistage Simulation of Goal-Driven Agents

Angelo E. M. Ciarlini and Antonio L. Furtado
Departamento de Informática - Pontifícia Universidade Católica do R.J.
Rua Marquês de São Vicente, 225 - 22.453-900 - Rio de Janeiro, Brasil
angelo@inf.puc-rio.br, furtado@inf.puc-rio.br

Abstract

Multistage simulation is employed to help predicting the future actions of collaborating or competing agents, interacting in the mini-world covered by a temporal database. At each stage, their goals are inferred from the current situation, with the help of rules designed to model their behavioural patterns, in turn extracted from records of their past actions. Besides goal-inference rules, the simulation machinery adapts and combines plan-recognition and generation algorithms. The resulting environment is highly interactive, allowing the user to interfere in various ways. A prototype tool (IPG), implemented in Prolog augmented with constraint programming features, was developed and is being used to run experiments.

1. Introduction

The facts holding in a database state change as a consequence of actions performed by different agents. Sequences of actions involving certain related entities can be interpreted as coherent narratives, in which the agents try to fulfil their goals. The identification of narrative patterns in a database log, together with models of each class of agents, should help to predict, through simulation, the future action of the agents, and thus provide assistance to decision-making. The agents we model are human beings and corporations in general. They may be collaborating agents, comparable to what is provided by Multi-Agent Systems [6], wherein separate modules co-operate with each other during the execution of their respective complex tasks. Alternatively, we may handle competing agents, since our approach also covers conflict situations and the possible consequences they can originate.

In [9] we showed how plan-recognition and plan-generation algorithms can be used in the context of database narratives, in order to detect, from a few observations, what plan an agent is pursuing and, in case obstacles are found, how to adapt the intended plan or to generate an alternative plan aiming at the same goal; only the current state and a single goal state were considered. We now work with a richer and more realistic scenario, in which, in view of the interactions between agents, further goals can be induced whilst different reachable states are successively examined. Our work stems from Propp's seminal work on the structure of a specific genre of narratives [18], namely folktales, and incorporates contributions from previous work on goal taxonomies [23] and on Cognitive Science [20,21,22,7]. A prototype implementation in Prolog, augmented with constraint programming features [15], has been developed in order to perform simulation experiments; the project makes provision for the future addition of modules to support the preparatory knowledge discovery tasks.

The present paper focus on the simulation part of our environment. Section 2 reviews the basic foundations of our approach. Section 3 introduces the criteria and methods used to model and then simulate, throughout a multistage process where new goals are generated in response to an evolving context, the interactive behaviour of collaborating and competing agents. Section 4 describes the simulation environment and its prototype implementation, a
running example being included to illustrate the actual use of the environment. Section 5 contains concluding remarks.

2. Database narratives

Database facts can be conveniently denoted by predicates. Imagine an application domain whose conceptual schema, expressed in terms of the Entity-Relationship model, involves, among others: the entities employees and clients; the relationship serves; and the predicates is-employee(E); is-client(C) and serves(E,C). Such a schema can be used to express, respectively, that E is an employee, that C is a client, and that employee E is at the service of client C. Specific predicate instances, corresponding to facts of these three types, might be:

- is-employee(Mary);
- is-employee(Peter);
- is-client(Omega);
- serves(Peter, Omega)

A database state is the set of all predicate instances holding at a given instant. A state provides a description of the mini-world underlying the database.

On the other hand, actions performed in this mini-world can be denoted, always at the conceptual level, by operations. Assume that, amongst others, one can identify in our example application domain the operations:

- hire(E);
- replace(E1,E2,C)

in order, respectively, to hire E as an employee, and to replace employee E1 by employee E2 in the service of client C. A specific execution of the second operation is:

- replace(Peter, Mary, Omega)

If the set of predicate instances shown before constitutes a state $S_i$, then executing the above operation achieves a transition from $S_i$ to a new state $S_j$:

$S_i$
\[
\begin{align*}
\text{is-employee}(Mary) & \quad \text{is-employee}(Mary) \\
\text{is-employee}(Peter) & \quad \rightarrow \quad \text{is-employee}(Peter) \\
\text{is-client}(Omega) & \quad \text{is-client}(Omega) \\
\text{serves}(Peter, Omega) & \quad \text{serves}(Mary, Omega)
\end{align*}
\]

An operation can be specified, in a STRIPS-like formalism [10], by declaring its pre-conditions and post-conditions, which characterize the states before and after the operation is executed. Pre-conditions establish requirements, positive or negative, which must hold prior to execution whereas post-conditions express effects, consisting in predicate instances being affirmed or negated. Pre-conditions and post-conditions must be such that all static and transition integrity constraints are preserved [11].

The definition of replace is shown below, in a semi-formal notation:

- replace(E1, E2, C):
  - pre-conditions: serves(E1, C) \land is-employee(E2) \land \neg \exists C1 \text{ serves}(E2, C1)
  - post-conditions: \neg serves(E1, C) \land serves(E2, C)

The pre-conditions make provision for the constraint that an employee cannot serve more than one client. Notice, however, that some other obvious requirements seem to be missing, e.g. it is not indicated that E1 should be an employee. This kind of simplification is justified if a strict abstract data type discipline is enforced. Notice also that the effects indicated via post-conditions refer only to what is changed by execution. It is therefore assumed that anything else that held previously will still hold afterwards (so as to cope with the so-called “frame problem”).

At a given instant of time, a factual database provides no more than the description of the current state. Temporal databases [17] enable the keeping of descriptions of all states reached, without making explicit, however, what actions caused the transitions. This sort of information becomes available if, in addition to the records of facts, a log registering the (time-stamped)
executions of operations is maintained [8]. We will then say that, besides static descriptions, databases thus enhanced now contain narratives, in the sense to be illustrated in the sequel.

Suppose that, after our example database (owned by a corporation called Alpha) has been running for some time, one extracts from the log all operations concerning a given client, say Beta, and the employees assigned to its service, the execution of which occurred during a given time interval. Let the obtained sequence, kept ordered by time-stamp, be:

**Plot 2.1**
- a. open(client: Beta)
- b. hire(employee: John)
- c. assign(employee: John, client: Beta)
- d. complain(client: Beta, employee: John)
- e. train(employee: John, course: c135)
- f. raise-level(employee: John)

The sequence above, named **Plot 2.1**, summarizes a narrative, which justifies calling it a *plot* [9]. Plots should suffice to capture (here, in easy-to-process standard notation) the essential structure of some meaningful succession of events. Indeed, **Plot 2.1** can be read under the expanded form of a natural language text – thus making explicit the underlying full-fledged narrative – such as:

**Narrative 2.1.** "Beta became a client of Alpha. John was hired at the initial level, and then assigned to the Beta account. Later, Beta complained about John's service. John participated in training program c135. There were no further complaints from Beta. John's level was raised."

Besides registering past executions of operations, the log can be further used as an agenda. Future executions, either merely possible or firmly scheduled, can be registered in the log, subject, of course, to later confirmation or cancellation.

Narratives contained in a database log, far from being fortuitous, usually reflect the goals of the several agents who promote the execution of operations. It is therefore useful to distinguish, from amongst the many possible effects of an operation, those that correspond to achieving a recognized goal of an agent; intuitively, they are the reason for executing the operation – which justifies calling them the *primary effects* of the operation.

Sometimes a goal corresponds to the combination of the primary effects of more than a single operation. Even when only one operation would seem to be enough, it may happen that its pre-conditions do not currently hold, but might be achieved by the preliminary execution of another operation. In all such cases, a partially ordered set of operations is required, the execution of which in some sequence leads from an initial state $S_0$, through an arbitrary number of intermediate states, to a final state $S_f$ where the goal holds. On the other hand, there are cases in which one finds more than one set of (one or more) operations as alternative ways of reaching a given goal. Such partially ordered sets (posets) constitute *plans*. Calling the operations discussed thus far *basic operations*, we may regard entire plans as higher-level *complex operations*, to be defined from the basic ones by successive applications of two abstraction criteria:

- **composition** - two or more operations (part-of hierarchy)
- **generalization** - more than one alternative (is-a hierarchy)

A convenient way to derive useful complex operations is to adopt a knowledge discovery approach. After the basic operations and the main goals of the various agents have been preliminarily characterized, one lets the database be used for some time, allowing the log to grow to a substantial size. Sets of plots relevant for some reason are then extracted and compared in order to detect frequently occurring *plot patterns* (mainly via most specific generalization [12]). From these plot patterns, the complex operations should be synthesized.
It is fair to say that they would then correspond to *typical plans*, in that they reflect how agents have been proceeding in practice towards their goals.

In our example, assume that the identified goals of company Alpha are: obtain clients, keep them satisfied. With respect to the second goal, a situation where a client has complained is undeniably critical. So, sequences of operations related to an unsatisfied client, from the moment when his misgivings became patent to the moment when they ceased, seem to qualify as relevant plots. We have shown before:

Plot 2.1

... 
  d. complain(client: Beta, employee: John) 
  e. train(employee: John, course: c135) 
... 

And now consider another plot, everywhere similar to Plot 2.1, except for the solution adopted:

Plot 2.2

... 
  d. complain(client: Delta, employee: Robert) 
  e. hire(employee: Laura) 
  f. replace(employee: Robert, employee: Laura, client: Delta) 
  g. fire(employee: Robert) 
... 

The two plots reveal two different strategies used by Alpha to placate a client. If the log contains a significant number of plots analogous, respectively, to Plot 2.1 and Plot 2.2, then two plot patterns are identified: as in Plot 2.1 -- *train*; as in Plot 2.2 -- *hire* - *replace* – *fire* suggesting the introduction of two complex operations: *reallocate* (by composition -- *hire*, *replace*, *fire*) and *improve-service* (by generalization -- *train* or *reallocate*). Once all operations have been defined one can build and keep for later use a hierarchically-structured *library of typical plans* by continuing with this process. With the graphic conventions proposed by Kautz [13], double arrows stand for *is-a* links (generalization) and single arrows for *part-of* links (composition). Basic operations are placed as leaves and an *end* node as the root. The library of typical plans for our example is shown in Figure 1.

![Figure 1: Library of typical plans](image)

3. **Modelling the behaviour of agents**

Our model uses plan-recognition and plan-generation as complementary processes in the generation and understanding of interactions of database agents. For a complete model, we basically need four kinds of transformations:
1. Plan recognition: observations $\rightarrow$ possible plans. Observations about database states and operations thus far attempted or actually executed are matched against a library of typical plans, to predict which known typical plans might be intended by the agents.

2. Goal recognition: plan $\rightarrow$ goal. Once a plan is recognized, through the matching process above, its corresponding goal is determined as a by-product of the process, since each typical plan kept in the library is a complex operation the main effect of which is registered as part of its definition.

3. Plan generation: goal $\rightarrow$ plan. Given a goal, a planner is applied for the generation of partially-ordered sets of operations able to achieve the goal.

4. Goal generation: database state $\rightarrow$ goal. The execution of a plan changes the current database state, and, at the modified state, further goals can be inferred by activating appropriate rules relating the new situation with the needs and motivations of the various agents.

All these transformations are provided by our system. In this paper, however, we are interested mainly in transformations 3 and 4 and the connections between them, because they are the activities directly related to the simulation of the interaction among database agents.

Most sequences of events occurring in the mini-world modelled by a database are not motivated by a single ultimate goal initially formulated. When an agent executes operations to achieve one of his goals, he changes the current database state. New goals, both of the original agent and of others, may arise as a result of the modification. Further executions of operations, corresponding to new plans, are necessary to achieve the newly identified goals. Thus, we have a progressive process, in which a plot (denoting, as explained, a database narrative) is generated on the fly by the alternation between generating goals and planning additional executions of operations. It is important to notice that new goals are motivated not only by past events but also by operations already scheduled for future execution and by the beliefs of each agent about the future behaviour of the others.

The actions performed for the achievement of a goal may help to achieve other goals, establishing a positive relationship among all of them. But, often leading to intriguing problematic situations, goals may have a negative interaction, requiring their mutual conciliation or the abandonment of certain goals.

Imagine, in the example of Company Alpha, that an employee was promoted. Seeing this, his co-workers may want to be promoted too. In order to achieve their goal, they may decide to take a course and improve their skills. If no more than two employees are allowed to take a course at the same time, we have a negative interaction, which demands a solution. The simulation of the agents’ behaviour should handle such situations and simulate all possible outcomes. This section explains how we propose to model the behaviour of database agents. We first discuss the relationship among goals, which delimits the kind of situations we intend to work on. Next, we describe how we can use interactive planning to simulate the interaction of database agents when there are many goals to be fulfilled. Then the notion of progressive generation of new goals is introduced. Finally, we compare our representation of the behaviour of agents with cognitive structures used in Natural Language Processing for text understanding. For easier reading, we shall present the various illustrative examples under the form of natural language textual narratives, rather than in the compact plot format used in our methods and prototype implementation.

### 3.1. Goal Relationships

In [23], Robert Wilensky proposed a classification of goal relationships occurring in everyday situations as either positive or negative. His classification proved to be very helpful in the characterization of situations to simulate. It can be used to model database agent interactions, since Wilensky's examples of everyday situations are not much different from those arising in database applications.
The positive relationships are divided into two cases: goal overlap and goal concord. In
the former, an agent has more than one goal and it is advantageous to plan simultaneously for
their joint achievement. An example of this situation is:

**Narrative 3.1.** “An employee wanted to be promoted and he also liked studying, so he decided to take a course
rather than work extra hours for the company”.

In the latter case, different agents have goals that are mutually beneficial. For instance:

**Narrative 3.2.** “A company had many potential clients, but not enough employees to assign to all of them. A
qualified person applied for the job. The company hired this person and assigned him to one of its new
clients”.

In [24], Wilensky stressed the importance that negative goal relationships have in the
development of a story. He affirmed that, from a literary viewpoint, stereotyped situations do
not make good stories, because they are too dull, so people have little motivation to tell or be
told about them. Interesting stories have points, i.e. something that provokes the interest of a
reader. A point often involves negative goal relationships. We quite agree that the simulation
of database agent interactions is particularly useful in situations with negative relationships,
especially if there is a competition between corporate goals and those of other agents.

Negative interactions may be caused by:

1. **Resource limitation:** Two goals require a common resource, but either there is an insufficient quantity of
   the resource or the resource is unavailable to satisfy both goals. An example of this case would be two
   people presenting themselves as candidates for the same position in the hierarchy of a company.
2. **Mutually exclusive states:** Two goals necessarily correspond to mutually incompatible states. For instance,
   a company wants to reduce its operational costs and, at the same time, an employee solicits a salary raise.
3. **Activation of a preservation goal:** The execution of a plan to fulfill a goal, may be detrimental to some
   preservation goal of an agent. For example, giving generous pay raises to employees should keep them
   satisfied and motivate them to work better, but the company may run the risk of bankruptcy.

   Negative interactions can be divided into two classes: internal (conflict), corresponding to
goals of the same agent, and external (competition), corresponding to goals of distinct agents.
Although the causes of conflicts and competitions may be similar, the situations they cause
have significant differences.

Consider the following conflict situations:

**Narrative 3.3.** “Company Alpha wants to keep its clients satisfied and all its employees assigned to clients.
John was serving client Beta. Beta complained about John’s service. Alpha hired Paul, replaced John by Paul
in the service of Beta and fired John.”

**Narrative 3.4.** “Company Alpha wanted to obtain two new clients, but they were very demanding and there
was only one employee that could be assigned to them. Alpha decided to obtain only the more profitable
client”.

**Narrative 3.5.** “John wanted to take course c135, but his client was very demanding and John had no free
time. Paul replaced John in the service of the client. John was able to take the course”.

These narratives are examples of the following types of situations caused by conflicts:

1. **Conflict resolution:** The circumstances that gave rise to the conflict are changed in such a way that the
   conflict disappears, as in narrative 3.3.
2. **Goal abandonment:** The agent opts for one of his goals and drops the other entirely, as in narrative 3.4.
   Instead, the agent could prefer to fulfill partially one or both goals.
3. **Spontaneous conflict resolution:** Something, not directly connected to the fulfillment of the goals, happens
   and changes the circumstances in such a way that there is no conflict anymore, as in narrative 3.5.

Let us consider now some narratives with competition:

**Narrative 3.6.** “John and Paul wanted a promotion, but only one could be promoted, because they worked for
the same department. Then, John asked to be transferred to another department”.

**Narrative 3.7.** “John and Paul wanted a promotion, but only one could be promoted. In order to increase his
chances, John decided to take course c135”.

Narrative 3.8. "John and Paul wanted a promotion, but only one could be promoted. Company Alpha changed its criteria, allowing the promotion of both.

We could classify situations originated by competition, such as those above, into three types:
1. Competition resolution: One or more of the agents with competing goals make efforts to avoid the competition, as in narrative 3.6.
2. Competitive plan execution: One or more agents alter their plans to try to win the competition. They can try either to outperform the others, as in narrative 3.7, or to undermine their opponents' efforts.
3. Spontaneous resolution of competition: The resolution is caused by an event independent of the effort of the competitors, as in narrative 3.8.

Despite the similarities between Wilensky's approach and ours, which enabled us to borrow from his goal taxonomy, an essential difference stands out: his planner acts as one specific agent who tries to achieve his goals and applies metaplans to deal with interactions with the other agents. Our approach postulates a planner that, in reality, is an external dispatcher, choosing operations to be executed by different agents and deciding which narratives can arise from the interaction of their various goals. We are particularly interested in modelling agent behaviour as logical rules, through which goals can be inferred based on a partial description of a narrative. In order to analyze the behaviour of a company and its consequences, it is important to model not only the other agents' behaviour, but also the policies and practices of the company itself. Our simulation methods should cope with situations in which plans are interrupted because certain goals are abandoned. In this case, actions of the plan prior to the abandonment of a goal would not be undone, and must be taken into account in view of the impact they may have on goals and plans of other agents.

3.2. Interactive Planning

Aiming at the simulation of database agent interactions, we use planning methods to discover possible operations that the agents can execute to achieve their goals. Planners implementing a simple backward-chaining strategy are not enough to solve our problem for the following reasons:
1. They do not cover the case in which many goals of many different competing or collaborating agents are to be achieved.
2. They introduce constraints that are not imposed by the satisfaction of goals but by the algorithm itself. They treat plans as totally ordered sequences, so that, for all pairs of operations, one must precede the other for execution, even when there is no causal connection between them. Assume for instance the following narrative:

Narrative 3.9. "John and Paul worked in different departments. John wanted to be transferred to Paul's department. Only one employee from each department could take course c135. Both John and Paul wanted to take that course. John was transferred to Paul's department. Paul took course c135. John was not able to take the course."

If the planner had not unnecessarily chosen and fixed an order, in this case establishing that John should be transferred before Paul would take the course, it would be able to conciliate their goals by letting John take the course before being transferred.
3. There is no priority for the resolution of goals. We may wish to establish, for instance, that corporate goals have higher priority than any other goals.
4. They neither support abandonment of goals nor competitive plan execution, which usually entail the need to force one agent to give up one of his goals.

Because of reasons 1 and 2, and also in the benefit of efficiency, we looked at non-linear planners in the Tweak [3] category. A non-linear plan consists of a set of partially ordered operations. In order to check whether a pre-condition of an operation is satisfied, non-linear planners apply the Modal Truth Criterion (MTC). They use a least commitment strategy, according to which a constraint (on the order of execution of operations or prescribing
codesignations and non-codesignations of variables) is activated during the generation of a plan only when necessary. This then happens when the constraint must be applied to consider the insertion of an operation for establishing a pre-condition of another operation, or to resolve conflicts between operations.

According to MTC, a condition C is necessarily true at the execution time t1 of an operation (user) if C was necessarily established by another operation (establisher) at time t2, before t1, and there is no operation (clobberer) that can establish the negation of C at a time t3 between t2 and t1. A condition is possibly true, at a given time t1, if it is either necessarily true or can be made true by the addition of constraints on time and on the values of variables.

In order to be able to assign priorities for the resolution of certain goals, and also to enhance the performance of our simulation environment, we adapted a non-linear planner (AbTweak [25]) that is also a hierarchical planner. It tries to resolve all pre-conditions of all operations at a certain level before taking into account lower level pre-conditions. It is a useful mechanism, particularly when we envisage forcing an agent to give up one of his goals. Being inserted first, operations for solving a high priority goal may prevent the insertion of operations aiming at conflicting lower priority goals. That will be the case if we establish that “corporate goals are more important than employees’ goals”.

The planner performs a heuristic search for good plans. It works on many plans in parallel and each time selects the candidate with minimal cost. Then, it selects a pre-condition of an operation that is not necessarily true and tries to make it true. While doing that, the planner generates all possible successors of the current plan. It considers the possibility of using establishers already included in the plan as well as the insertion of new operations. All possibilities of resolving conflicts caused by clobberers are also considered.

We use two main mechanisms to simulate goal abandonment and competitive plan execution: conditional goals and limited goals.

Conditional goals have a survival condition attached to them, which the planner must check to determine whether the goal should be pursued or not. Operations added for the fulfilment of conditional goals are also conditional. They are kept as part of the plan only if their survival conditions are valid at the appointed execution time. However, even in case of failure, the planner keeps the operations preceding the instant when the failing survival conditions ceased to hold. Survival conditions and pre-conditions must be handled in a consistent way: the planner begins by trying to solve all pre-conditions of all operations at least once; if a survival condition of an operation fails, the planner determines that its pre-conditions will not be considered for a second time. The following narrative is an example involving conditional goals:

**Narrative 3.10.** “John wanted to be promoted and then he took a course. His boss, however, afraid that John might be competing for his position, asked the general manager to fire him. John was fired”.

In Narrative 3.10, John’s final goal (getting a promotion) was frustrated because a survival condition (remaining employed) failed. However, the fact that John took a course was not invalidated.

Limited goals are those that are tried once only, and have an associated limit (expressed as a natural number). The limit restricts the number of new operations that can be inserted to achieve the goal. An operation inserted for the establishment of limited goals is termed a limited operation. A limited operation tentatively inserted in a plan can be kept only if all its pre-conditions are true. In addition, all its pre-conditions are in turn limited subgoals, the limit of which is that of the original goal minus one. A goal with limit 0 can only be satisfied by an operation already present in the plan. We use limited goals to model situations in which agents are willing to invest no more than a certain measure of effort, not necessarily the same for each agent, towards the fulfilment of their goals. The following narrative is an example:
Narrative 3.11. “John and Paul wanted to be promoted. An employee is promoted if his status is higher than a certain level. The status of an employee can be increased either by taking course c135 or if the evaluation of his service is good. An employee can take a course only if he works extra hours. John and Paul tried to improve their services. Despite their efforts, their status was still insufficient for promotion. Therefore, they had to take course c135. Paul gave up his promotion, because he would have to work extra hours to be able to take the course. Being more ambitious than Paul, John worked extra hours and took course c135. John was promoted but not Paul.”

Narrative 3.11 evidences that raising his status up to the promotion level was a limited goal (with limit of 1) for Paul. Indeed, he was willing to execute one operation: taking course c135. But, differently from John (whose limit should be equal or greater than 2), Paul would not execute two operations: working extra hours and taking course c135.

In order to adapt planning methods to the database context, we implemented two additional features. We modified MTC to cope with the Closed World Assumption [19]. All facts not belonging to the database are considered (initially) false. So, we had to provide a special treatment for the evaluation of negative conditions, because they can be established simply by observing the absence of the corresponding positive facts in the database. We also used Constraint Logic Programming [15] to deal with pre-conditions that involve arithmetic constraints, frequently involved in database updates. The added features enable the handling of such pre-conditions even in the presence of free variables, since Constraint Logic Programming leaves a goal frozen until the moment it can be evaluated. In order to process numbers, it provides a solver of systems of equations and inequations able to run in parallel with the planning procedure. The solver tries as early as possible to find out whether a frozen goal can no longer be fulfilled, in which case it immediately cuts the current branch of the search tree, thus causing an improvement in the performance of the overall system.

3.3. Progressive Goal Generation

In order to generate the goals that the agents will pursue in specific situations, one needs to establish a relation between database states and goals. To characterize modification of states by means of operations, a Temporal Logic notation, e.g. Event Calculus [14], would seem to be an obvious choice. However, because we are working with database events, we decided to also borrow from Multi-Sorted Database Logic [1]. Accordingly, our logic is a Multi-Sorted Database Logic in which a special status is conferred on the sort time-stamp. Our truth model is based on the MTC criterion, which is fully compatible with Event Calculus with Preconditions [2]. We do not formalize our logic here, because of space considerations. Instead, we describe informally the meta-predicates used for speaking about database facts and operations associated with time. Then, we indicate how to write rules, using these meta-predicates, aiming at the specification of the behaviour of agents.

The order of execution of operations being merely partial, two different criteria for truth evaluation are needed. We should be able to verify whether a fact associated with a certain time-stamp is either necessarily or possibly true. In addition, it must be possible to speak about the time when a fact was established by an operation, because a condition is valid only after (but not at) its establishment. We should also be able to mention the time when a specific operation happened (i.e. was executed). Accordingly, we have four temporal meta-predicates, noting that LITERAL stands for a positive or negative fact:

1. \(h(T, \text{LITERAL})\): LITERAL is necessarily true at \(T\).
2. \(p(T, \text{LITERAL})\): LITERAL is possibly true at \(T\).
3. \(e(T, \text{LITERAL})\): LITERAL is established at \(T\).
4. \(o(T, \text{OPERATION})\): OPERATION happened at \(T\).
And, in order to express constraints relating variables of specific sorts, we have two additional meta-predicates:

5. \(h(\text{CONSTRAINT})\): CONSTRAINT is necessarily true.
6. \(p(\text{CONSTRAINT})\): CONSTRAINT is possibly true.

The atomic formulae are meta-predicates and negations thereof.

The rules are formulated as implications, in which the antecedent is a conjunction of atomic formulae and the consequent is a conjunction of meta-predicates of types 1 and 5 only. Furthermore, whenever conditional or limited goals are involved, their associated conditions or limits, respectively, must be included in the rule definition. If the antecedent holds, then the consequent will correspond to a goal to be pursued (unless it already holds). By assumption, all variables in the antecedent are universally quantified, whereas those appearing only in the consequent are existentially quantified. For consistency with the Closed World Assumption, variables that appear only in negated meta-predicates or facts are regarded as existentially quantified within the negation. Variables are represented by capital letters. “\(_n\)” stands, as in Prolog, for anonymous variables.

To express the rule:

“If C became a client at T1 and, at time T2 (after T1), no one serves C and there is a person E who is not serving any client, then the company will try to arrange that, at a time T3, after T2, E will be serving C”

we write:

\[e(T1,\text{is}\_\text{client}(C)) \land h(T2>T1) \land \neg e(T2,\text{serves}(\_\_C)) \land e(T2,\text{person}(E)) \land h(T2,\neg \text{serves}(\_\_C)) \land \neg e(T2,\text{serves}(E,\_\_)) \rightarrow h(T3>T2) \land h(T3,\text{serves}(E,\_\_))\]

When there is a set of goals to be achieved, by potentially different agents at potentially different times, the simulation proceeds as described in 3.2. When all these currently active goals have been either fulfilled or discarded, the planning process completes one stage. Then, the set of existing rules is exhaustively evaluated, over the initial database state and the alternative future states resulting from the set of partially ordered operations generated thus far. If the consequent of a rule does not already hold and the corresponding goals have not previously been tried, these new goals will be pursued in the next stage of the planning process. The cycle ends when no new goal can be generated.

This progressive generation of goals can be compared to the triggering process in Active Databases [5]. A trigger is usually a rule defined by a triple \(<\text{Event}, \text{Condition}, \text{Action}>\). When an event happens, a condition is evaluated and, if the condition is true, the corresponding action is performed automatically. Trigger events and conditions are semantically equivalent to the antecedents of our rules. The consequent of a trigger, however, is an action, whereas the consequent of our rules simply corresponds to goals.

3.4. Cognitive Structures

In the Natural Language Processing area, some interesting data structures have been proposed to facilitate the understanding of the connection between events in a narrative. The knowledge expressed by such structures is captured in our model by means of rules, defined as above, and typical plans, organized in a library (e.g. Figure 1 in section 2).

Schank and Abelson [20] proposed the notion of scripts to avoid the combinatorial problem of considering all possible inferences about interconnections of events when trying to understand a text. A script is a stereotyped sequence of predefined actions corresponding to a well-known situation. Later, Schank developed the concept of MOPs (Memory Organization Packets) [22], which constitute an evolution of scripts, enabling the representation of the intention of each character when performing an action. Our approach not only handles
stereotyped situations, which are represented by the complex operations kept in a library, but also, through planning, copes with new unanticipated situations.

Schank proposed also the concept of TOPs (Thematic Organization Packets) [21], to register thematic relations among texts. Dyer used in his system [7] structures at an intermediate position between MOPS and TOPs, named TAUs (Thematic Abstraction Units). TAUs categorize stories by way of cross-contextual reminders and the attribution of appropriate adages. TAUs are often related to conflict situations. In general, both TOPS and TAUS correspond to a more abstract level than that which we treat directly. Still, we can associate the primary effects of end-plans to instances of TOPs. Additionally, certain complex operations and rules can be associated to instances of TAUs. Some of our rules reflect adages applicable to specific contexts. For instance, the rule: “If an untrained employee is fired, then other employees will hurry to enrol in courses” can be thought to reflect, in the context of our simple business-oriented example, the adage:

“If your neighbour’s house is on fire, make haste to put out the fire in your own house.”

4. The Interactive Plot Generator

In order to try out the model described in section 3, a prototype system — Interactive Plot Generator (IPG) — was designed and implemented. In this section we first outline the overall structure and function of IPG, and then demonstrate its use with an example related to the simple database of company Alpha.

4.1. A short description of IPG

IPG mainly consists of two modules, which can be used either separately or in combination, to handle plots of narratives: one composes plots (the Simulator) whereas the other performs plan and plot recognition (Recognizer). They are complemented by three auxiliary modules: the Goal Evaluator, the Temporal Logic Processor and the Query Interface (whereby users can retrieve information about plots generated by the Simulator). The implementation of two additional modules is planned as part of the continuation of the project, with the purpose of automating the construction of the library of typical plans and the discovery of agent behaviour rules. Figure 2 displays the architecture of the system. Rectangles represent modules, ellipses correspond to data repositories, and arrows to the flow of data. Dashed rectangles indicate modules to be incorporated in future versions.

Three forms of utilization of IPG will be mentioned here:

1. The first form is centred on the Recognizer. The user inputs observations, which are taken by the system as hints about the plot to be generated (some of the operations to be executed, a few assertions about the situation of agents at intermediate states and goals to be finally reached, etc.), and the system looks for a typical plan pattern (or set of patterns) that may fit the observations supplied. The system fills in what is missing in the characterization of agents, taking into consideration the observations and the pertinent facts found in the initial state of the database.

2. The second form, in contrast, is based exclusively on the Simulator. The user inputs observations about the situation he wants to simulate; then, the rules specifying the behaviour of agents are evaluated over these observations and the initial state of the database, in an attempt to infer new goals. From this point on, alternate phases of plan generation (to achieve such goals) and of inference of goals (at each state reachable by the plans) occur, until no further goals result.

3. The third form involves an integrated use of the Simulator and the Recognizer. Upon recognition of a plot P, some of the pre-conditions of its constituent operations may not hold in the database. One option to obtain a valid plot is to adapt P, with the help of the Simulator. An alternative is to simulate an entirely new plot, whereby the primary effects of P are achieved. Even if a recognized plot is totally valid (all its pre-conditions hold), there is still the possibility of extending it with plans induced by the goals inferred at states determined by its effects. Such extensions can either be proper continuations, added strictly after the
completion of the plot, or plot enrichments, with the additional sequences of operations interpolated or ranged in parallel.

![Diagram of IPG architecture]

**Figure 2: General architecture of IPG**

The Recognizer uses a plan-recognition algorithm adapted from the algorithm defined by Kautz [13]. The adaptations extend it to handle pre- and post-conditions of operations and establish its connection with the Simulator. The Recognizer identifies patterns by matching the observations supplied against the library of typical plans (defined as described in section 2). Observations concerning states reachable during the execution of plots are verified by consulting the Temporal Logic Processor. The plots recognized are communicated to the user, together with the indication that its pre-conditions can (or cannot) be satisfied. According to the user's choice, the Simulator can be activated to adapt or extend plots, or else to generate plots with the same primary effects identified. The Recognizer explores all alternatives recognizable and displays them to the user, as requested.

The Simulator executes planning tasks, as described in 3.2, supported when necessary by the Temporal Logic Processor. To obtain new goals, the Simulator calls the Goal Evaluator, which performs their evaluation as explained in 3.3. During simulation, the selection, from the set of partially generated plots, of those still deserving examination is made by interacting with the user. When a partial plot is exhibited, the system signals to the user that the Query Interface is available. The Query Interface enables the user to formulate queries about the situation of agents at diverse states along the plot. For this purpose, the Query Interface can also access the Temporal Logic Processor. After consulting, the user may return normally to the Simulator and decide whether or not to proceed with the current plot.

In our example database, the construction of the library of typical plans was done systematically (as in section 2) but manually. The same happened with behaviour rules, in the formulation of which we were guided by previous work on goal relationships (3.1) and interconnections of events (3.4). We are currently investigating methods to support these processes, to become part, respectively, of the future **Library Construction** and **Rule Formulation** modules.

The IPG prototype was implemented in SICSTUS Prolog [4], and extensively employs its constraint programming features to express arithmetic relationships and non-codesignations among variables, as well as to evaluate temporal logic expressions.
4.2. Running Example

In the example of company Alpha, there are three different classes of agents: employees, clients and company Alpha itself. In order to model their behaviour, we defined in the logic notation described at 3.3 — here transcribed informally for easier reading — the following rules:

1. Whenever a client C has no employee to serve it, and there are people still unassigned to any client, such people will compete for the position until one of them achieves the goal.
2. Whenever an employee is not content with his salary, he will try, in accordance with his persistence, to increase his status by one.
3. Whenever an employee has a status higher than what his salary indicates, his salary level will be raised by one.
4. Whenever the salary of an employee E goes beyond the budget of client C, whom E is serving, E will stop serving C.
5. Whenever an employee E is not associated with any client and there are neither clients without employee nor potential clients to conquer, E will cease to be Alpha's employee.
6. Whenever a client C is dissatisfied with an employee E at its service, some action will be undertaken to remedy the situation.

Rules 1 and 2 give origin, respectively, to conditional and limited goals.

An initial database is given with 3 employees (David, Leonard and John), one unemployed person (Mary) and three clients (Beta, Epsilon and Lambda). David is serving Epsilon and Leonard is serving Lambda. All employees have salary level 1 and the budget of every client company can afford, at most, salary level 2. Both David and Leonard have the ambition of reaching salary level 3, but David's persistence is stronger than Leonard's.

```
assign(john,beta)
Goal(Stage 1): serves(john,beta)
work_hard(david)
Goal(Stage 1): status(david,2.0)
work_hard(leonard)
Goal(Stage 1): status(leonard,2.0)
hire(mary) - CANCELLED
assign(mary,beta) - CANCELLED
Goal(Stage 1): serves(mary,beta) - CANCELLED
raise_level(leonard)
Goal(Stage 2): has_level(leonard,2.0)
raise_level(david)
Goal(Stage 2): has_level(david,2.0)
train(david,c135)
train(david,c136)
Goal(Stage 3): status(david,3.0)
train(leonard,c136) - CANCELLED
Goal(Stage 3): status(leonard,3.0) - CANCELLED
raise_level(david)
Goal(Stage 4): has_level(david,3.0)
hire(mary)
replace(david,mary,epsilon)
Goal(Stage 5): not(serves(david,epsilon))
fire(david)
Goal(Stage 6): not(is_employee(david))
```

Figure 3: Plot after six stages

After iterating across 6 stages, the system generates the narrative described below, again transcribing into natural language the actions belonging to each stage:

**Narrative 4.** “David and Leonard work hard and their status is raised to 2. John and Mary compete for client Beta. John wins and Mary's actions towards this objective are cancelled. David and Leonard have their salary raised. David and Leonard try to increase their status again. But now this requires that two courses be taken. David takes them, whereas Leonard abandons his attempts to reach a higher status. David's salary is raised
again. Since David's salary now exceeds Epsilon's budget, Mary is hired and replaces David at the service of Epsilon. Since David is no longer serving any client, and there is no current client to which he might be appointed, nor even potential clients to bring in, David is fired”.

Notice the looping structure generated, whereby David's repeated initiatives seem to improve his situation as far as status and salary are concerned, but their cumulative impact on Epsilon's expenditures end up compromising the stability of David’s assignment.

The plot obtained at the conclusion of the sixth stage is shown in Figure 3 in one of its possible orders. Both the constituent operations and the goals successively inferred are displayed. Planned operations later abandoned are marked as “cancelled”.

5. Concluding Remarks

Simulation offers answers to "what if" questions, so crucially relevant to decision-making. A user conducting simulation experiments has thus an opportunity to anticipate and compare the favourable and unfavourable results of different lines of action, some of which conforming to the traditional attitudes and policies adopted in the past, and others exploiting valid but still untried alternatives. Our environment leaves room to the user's interference, so that his personal knowledge can guide the choice among options. Besides helping to decide with respect to specific situations, its continued use may disclose shortcomings in how agents and operations are currently modelled, thus leading to the revision of the inadequate models.

Planning algorithms are known to be, in the unrestricted cases, semi-decidable and exponential, and, therefore, good heuristics are indispensable to speed-up convergence towards the achievement of goals marked with highest precedence. We have been looking at this topic in the context of our prototype system, but more work is still needed. Even with the present version, however, promising results have already been obtained in very diverse domains, enabling the treatment of considerably more complex plots of narratives than commonplace database applications. Folktales are providing a nontrivial benchmark, and the schema of narrative 4 suggests that phenomena involving "circular causality", with positive and negative feedback effects, such as the unstabilizing effect of government efforts to control economic crises [16], can also be handled. Further research will involve gaining experience with our interactive multistage planning paradigm, as well as investigating methods to be applied in the enhancement of the system, with particular emphasis on the automatic construction of libraries of typical plans and formulation of behaviour rules.

References