PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 17/05

Conceptual Modelling for Storytelling (with a Case Study)

Antonio L. Furtado **Angelo E. M. Ciarlini**
Bruno Feijó **César T. Pozzer**

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 17/05 ISSN 0103-9741
Editor: Prof. Carlos José Pereira de Lucena May 2005

Conceptual Modelling for Storytelling (with a Case Study)*

Antonio L. Furtado Angelo E. M. Ciarlini[1] Bruno Feijó César T. Pozzer

[1]UniRio - Departamento de Informática Aplicada

{furtado,bruno,pozzer}@inf.puc-rio.br angelo.ciarlini@uniriotec.br

Abstract: How to characterize a literary genre is a much debated problem, which can be approached with useful results by combining models drawn from both Literary Theory and Computer Science. Once a genre is specified with some rigour in a constructive way, it becomes possible not only to determine whether a given plot is a legitimate representative of the genre, but also to generate such plots, an ability of obvious relevance to Storytelling theory and practice. A conceptual modelling method with this purpose is presented, based on a plan recognition/ plan generation paradigm. The method leads to the formulation of static, dynamic and behavioural schemas, expressed in temporal logic, and to multi-stage interactive plot generation, supported by a prototype tool. The tool is of special help for the composition of plots through the adaptation of plots fully or partially generated in automatic mode. A case study, involving a simple Swords and Dragons genre, illustrates the discussion.

Keywords: Storytelling, Literary Genres, Conceptual Modelling, Simulation, Logic Programming.

Resumo: Como caracterizar um gênero literário é um problema debatido desde longa data, que pode ser abordado com proveito combinando modelos tomados da Teoria Literária e da Ciência da Computação. Desde o momento em que um gênero é especificado com algum rigor e de modo construtivo, torna-se viável não apenas determinar se um dado enredo é um representante legítimo do gênero, como também gerar tais enredos, o que constitui uma possibilidade de relevância óbvia para a teoria e a prática de Narração de Estórias. Um método de modelagem conceitual com este propósito é apresentado, cujo fundamento é um paradigma de reconhecimento e geração de planos. O método conduz à formulação de esquemas estáticos, dinâmicos e comportamentais, expressos em lógica temporal, e à geração interativa em estágios múltiplos de enredos, com ajuda de uma ferramenta experimental. A ferramenta apoia de modo especial a composição de enredos através da adaptação de enredos completa ou parcialmente gerados de modo automático. Um estudo de caso, envolvendo um gênero simples de Espadas e Dragões, ilustra a discussão.

Palavras-chave: Narração de Estórias, Gêneros Literários, Modelagem Conceitual, Simulação, Programação em Lógica.

* This work has been partly sponsored by the Ministério de Ciências e Tecnologia da Presidência da República Federativa do Brasil.

In charge for publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

1. Introduction

We address here the question: what is a literary genre? And we would like to find a way to answer the question that, in addition, (a) will allow us to determine whether or not a story can be classified as belonging to the genre, and (b) will come equipped with a device to generate such stories. Put in this way, the question looks over-ambitious but, as we shall argue, a useful approximation to the solution of this much debated problem is attainable.

Studies in *Narratology* [Ba] teach us that the composition of stories is a three-layered process, and that each layer can be analysed separately: fabula, story and text. Informally speaking, *fabula* is a series of *events* happening in a real or fictional world. The *story* corresponds to how the fabula is reported by the author. Finally, the *text* is the materialization of the story in natural language words.

To give one example, the mythical career of Ulysses, with the events in strict chronologic order, is the *fabula* that Homer had in mind when composing the *Odissey*. The *story* tells the hero's homecoming, and is organized in twenty-four books, in some of which the poet allows Ulysses to tell his own exploits in a long "flash-back" (technically, a case of *anachrony*). Finally, the epic poetic *text* produced by Homer is a series of dactylic hexameter verses, written in classic Greek. Clearly a prose translation of the *Odyssey* in a modern language would be a different text, but, if it is faithful, it should preserve the story as narrated by Homer, and consequently the original fabula. A more radical change may occur when someone retells the story, cutting or summarizing a number of episodes, linearizing what remains so as to eliminate the anachronies, etc., which, of course, results in a different story. Extreme cases of change would even modify the fabula, for example eliminating as allegedly incompatible with modern taste the interventions of the goddess Athena, and combining in one character (a *conflation*) two or more of Penelope's suitors.

In our work, we shall deal exclusively with the fabula layer. Accordingly, we shall view a genre as a set of plots, taking the word *plot* in the sense of a sequence of events. And the types of events allowed in the genre being defined will be restricted to a fixed repertoire, as done by the Russian literary theoretician Vladimir Propp [Pr] in his seminal work with the fairy-tales genre. This decision classifies our approach as primarily *plot-based*, in the terminology of *Storytelling* research [Sg], as opposed to a *character-based* orientation [CCM]. However it will become clear, in the course of the presentation, that we also contemplate some character-based aspects.

The method comprises three levels of *conceptual modelling*, wherein three schemas are successively structured, so as to provide: 1. a description of the mini-world where the narrative takes place, 2. what events can be enacted by the participants, and 3. what motives guide their behaviour. Several models, borrowed from Literary Theory and from Computer Science, are combined. Plots belonging to a genre are, to a certain point, comparable to sentences belonging to a language, which suggests the use of some Chomsky grammar, such as *story grammars* [Ru], as a mechanism to accomplish purposes (a) and (b) above. Our option was for a *plan-recognition/ plan-generation paradigm* [FC1], which is fully compatible with the nature of the schemas and, in addition to sharing with grammars the

ability to handle *syntax*, is particularly apt to cope with *semantic* and *pragmatic* aspects. The formalism is based on *temporal logic*, and the notation adopts the clausal format required by the Prolog logic programming language, in which the planning algorithms were written.

The paper is organized as follows. Sections 2, 3 and 4 discuss the three schemas, using as illustration the same case study, where a very simple *Swords and Dragons* genre is specified; each section first sketches informally the elements needing specification, then presents the pertinent modelling notions, and ends by showing their application to the example. Section 5 covers the concept of plot *adaptation* as a form of literary composition accessible to people who are not professional writers. It also describes certain features, available in a prototype tool developed as part of our project, which can help users to compose plots interactively, adapting automatically generated plot sequences, and having their interventions verified by the system to ensure that the resulting plots are valid representatives of the genre. The section ends with the report of a number of experiments with interactive plot generation. Section 6 concludes the presentation. The appendices list the three schemas and the initial state assumed in the experiments.

2. The static schema

2.1. Informal presentation of example

The example scenario (fig. 1) shows an ample field, on which certain landmarks can be distinguished. These are the White Palace, the Gray Castle, the Red Castle, the Church and the Green Forest. The White Palace is the home of princess Marian and also houses a temporary visitor, the knight Hoel. The Gray Castle is the home of Hoel and of another worthier knight, called Brian. The Red Castle is occupied by Draco, a flying dragon. In the Green Forest lives the magician Turjan. The White Palace and the Red Castle are protected by armed guardians, and the Green Forest by magical trees; the other places, including the Gray Castle, have no such defenses.

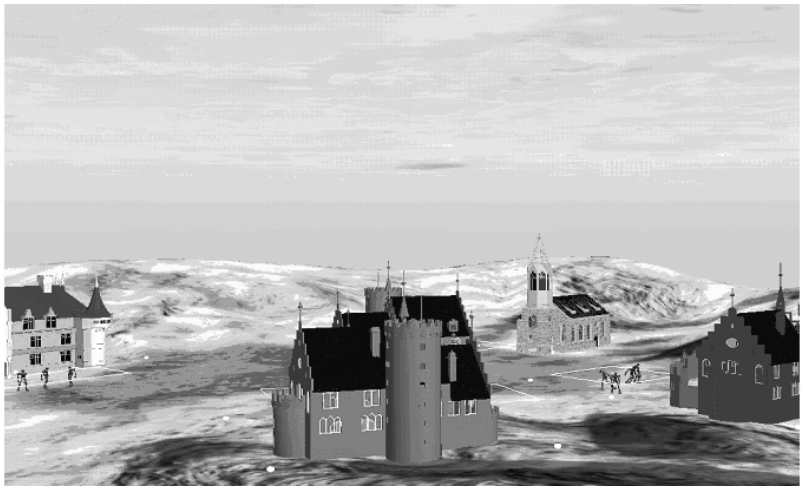

Figure 1: the scenario

These characters, both the persons and the dragon, can be described by their nature and their strength. As to nature, the princess and the knights are reputed to be on the side of goodness, whereas the dragon is evil; in contrast to all others, the magician is neutral. At the beginning, unsurprisingly, all characters are alive, and no one is stronger than the dragon. Differently from these leading characters, the protecting guardians figure as mere extras, individually undistinguishable. Relevant only in groups, they are a feature of the places they are charged to protect, and the protection afforded is characterized by the size of the group and by kind (which reflects the nature of the place-owners).

The inter-personal relations are simple. All characters are acquainted with each other, but demonstrate no mutual feelings initially, except for the two knights, who have a strong positive affection for the princess. At a later time, one of the heroes and the princess may eventually get married. On the negative side, the dragon may sometime have kidnapped the princess, and be keeping her under its custody. Even though, with the single exception of Hoel, they are in their homes at the beginning, the princess, the knights and the dragon are normally free to be at different places in other occasions; the magician, however, is confined to his sylvan refuge.

2.2. Modelling notions

The efficacy of the *Entity-Relationship* (ER) model, with a number of extensions of practical relevance, has long been recognized in the realm of the *application domains* of Business Information Systems [EN]. As will be argued here, it can be equally helpful for modelling the static aspects of *literary genres*.

An *entity* is anything of interest by itself, material or abstract, animate or not. Entities form *classes*, whose *instances* are distinguished by an (occasionally composite) *identifier*. Besides the identifier, other *attributes* may characterize the entity instances. Attributes have *values* of some type (alphabetic, numerical, etc.). Attributes of type *Boolean* (with values `true` or `false`) and *composite* attributes (with sub-divisions) are special cases.

Two or more entity classes may be associated through a *relationship*. In the so-called "unary" relationships, the same entity class participates more than once. Similarly to entities, relationships can have attributes. Binary relationships can be one-to-one (1-1), one-to-many (1-n) or many-to-many (n-n).

Extending the original ER model, entity classes can also be related through a *generalization/ specialization* hierarchy. If an entity class C specializes a more general class C', then class C is said to *inherit* all attributes defined for C'. Conversely, each instance of C is also an instance of C'. This *is-a* connection is transitive across the arbitrarily many levels of the hierarchy.

One more addition is necessary to the ER model, as a bridging notion towards the dynamic and behavioural levels to be described later. Whereas the other qualifying notions refer to what the entities *are*, we need, in order to indicate how they are expected to *act*, to

assign *roles* (in the theatrical sense, and in the sense of the *agent* concept, currently popular in Artificial Intelligence and Software Engineering) to certain entities.

One of the major contributions of the ER model is the ER *diagram*. Following a rather common tradition, we represent entity classes by rectangular boxes, attributes by circles and relationships by diamond shapes. The identifying attributes are underscored. Triangular nodes are employed to distinguish *is-a* connections. The assignment of roles (represented by ellipses) to entity classes is indicated by pointed edges.

The diagram only represents the schema, and therefore does not show particular instances, a subject to which we turn now. We call a *fact* an assertion about the existence of entity instances, or the values of the attributes of entity instances, or the existence of relationship instances, or the values of the attributes of the relationship instances, or the assignment of roles to entity instances. The set of facts holding at a given instant of time constitutes a *state*.

The static specification of a genre (exactly as that of a business application domain) requires that only *valid* states be admitted. A valid state must conform to certain *static integrity constraints*. Some constraints are inherent in the ER model: relationship instances can only exist among existing entity instances, attributes are in general single-valued (although certain values may be allowed to change along time), attribute values must be of the specified type, etc. Declaring a relationship to be 1-1, 1-n or n-n is an integrity constraint expressible in the ER diagram. Other integrity constraints imposed by conventions of the genre (or regulations of the application domain) are left to be expressed outside the diagram in some appropriate notation.

We find that logic and, more concretely, logic programming as provided in Prolog, meets adequately our notational demands. In particular, integrity constraints can be conveniently expressed by *rules*. In previous papers we have described the *modal temporal logic* that we use, and how it is transliterated into Prolog syntax [CF].

The clause patterns used in the specification of the static schema are given below. Note the use of square brackets for conjunctive lists (with "," as separator) and round brackets for disjunctions (with ";" as separator), in conformity with Prolog conventions.

```
entity(<entity-class>,<identifier>).
relationship(<relationship-class>,[<entity-class>,...,,entity-class>]).
attribute(<entity-class>,<attribute>).
attribute(<relationship-class>,<attribute>).
boolean(<attribute>).
composite(<attribute>,[<attribute-part>,...,<attribute-part>]).
is_a(<more-specialized-entity-class>,<more-general-entity-class>).
role(<role>,(<entity-class>;...;<entity-class>)).
```

The notation for representing facts is a straightforward consequence of the schema notation; it is enough to mention that terms `db(<fact>)`, i.e. *database facts*, denote the component entries of the initial state. From a pure formal logic standpoint, the schema is composed of *metalanguage* clauses, whereby the *language* used to write the clauses that

represent database facts is defined. In Prolog programs, however, both kinds of clauses assume the same notation, and can be used in combination in many very helpful ways, allowing, for example, to determine all attribute-value pairs qualifying a given entity instance at some state.

2.3. Example specification

The major entity classes are `characters` and `places`. The former are identified by `name` and have `nature`, `strength` and the condition of being `alive` (of Boolean type) as attributes. The latter are identified by `place name` and have the composite attribute `protection`, composed of `kind` and `level`. Between `characters` and `places` there are two 1-n relationships, `home` and `current place`. Relationship `acquaintance`, twice involving `character` participants (a "unary" relationship, in this sense), has attribute `affection` and is n-n. Relationship `married` is defined between `persons` only, and `kidnapped` between `persons` and `characters` in general (the kidnapper can be, and quite often is, a `dragon` in our simple genre); `married` is 1-1 and `kidnapped` 1-n (more than one `person` can be simultaneously held by one kidnapper).

The choice of a convenient type for attribute values is crucial. For example, one would at first choose something like `good` and `evil` as possible values for `nature`, as well as for `kind` of `protection`. We preferred instead `1` and `-1`, which permits their use in various arithmetic comparison formulas, involving `strength` and `level` of `protection` (as will be seen in section 3.3). An even more important choice was done for `affection`. Again, the intuitive preference might be some word indicating, for a pair of characters A and B, in this order, how A currently "feels" for B. Here, our choice was motivated by what is practically a consensus in *affective computing* [Ve] research: drives and emotions are better expressed as points in numerical scales within a given range (typically from 0 to 100). This makes it easier to describe increases and decreases in emotional intensity. Also, we decided to allow zero and negative values to denote, respectively, neutral and adverse feelings. Finally, in order to take advantage of the real-number constraint programming package of the Prolog version utilized, we write all numbers as reals, although we are only concerned with integer values.

The entity class `character` admits `person` and `dragon` as specializations. Furthermore, `princess`, `knight`, and `magician` specialize `person`. For our purposes, we did not care to specialize `place`, but this is largely a matter of taste; one mightn readily come up with a variety of distinguishing criteria applicable to our scenario.

Our choice of roles – `hero`, `victim`, `villain` and `donor` – is a subset of the seven *dramatis personae* proposed by Vladimir Propp [Pr] for Russian fairy-tales. Roles `hero` and `donor` are here assigned only to `knights` and `magicians`, respectively. On the other hand, although it is more natural to assign the role of `victim` to a `princess`, and that of `villain` to a `dragon` (as we do in the initial state exemplified here), we also allow in our specified genre that `knights` may figure as `victims` or `villains`.

The Entity-Relationship diagram in figure 2 displays these various components and the connections among them. The formal specification of the static schema, in the Prolog-compatible notation indicated before, is shown in appendix I.

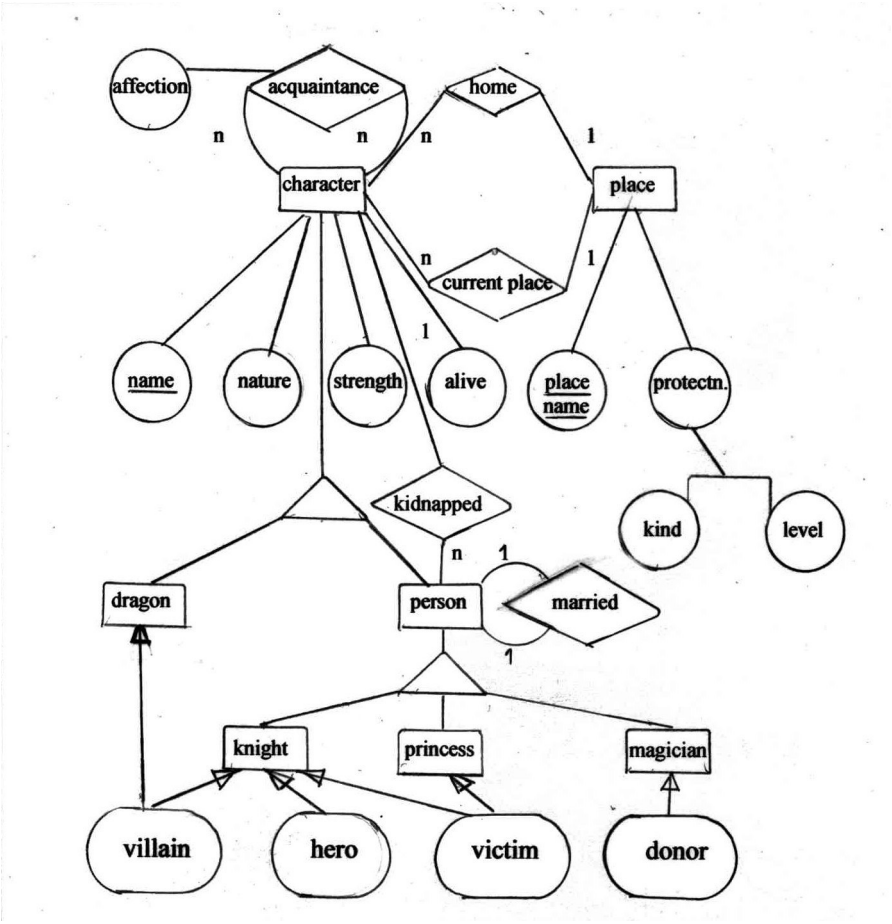

Figure 2: Entity-Relationship diagram

A number of *static integrity constraints* are assumed. The most obvious is that whatever attribute a `character` may have should only retain any significance while the character is `alive`. All attributes here are single-valued. If a `character` is playing the role of `villain`, his nature must be `-1`, whereas `heroes` and `victms`, who are the "good" `characters`, are rated `1`. Thus, in view of the single-valuedness of attributes, a `knight` can be at the same time `hero` and `victim`, but not `hero` and `villain` simultaneously. A `donor` does not take sides, his neutrality being marked by an intermediate `0` value. Reflecting the inclination of the owners, the `kind` of `protection` of the several `places` coincides with the `nature` of the `characters` who make them their `home`.

As the diagram shows, but not our Prolog clauses, only relationship `acquaintance` is unrestricted; the others are either 1-1 or 1-n, which constitutes an obvious static constraint. For a `magician`, his `current place` must at every state coincide with his `home`, a

restriction that does not affect the other `characters`. Moreover, `married` can only hold between `persons` of opposite gender, an attribute left unmentioned in our specification.

As will be explained in the next sections, we rely, for the small example discussed in this paper, on the combined correctness of the given initial state and of the dynamic schema to guarantee that such constraints will not be violated. However this is not a limitation of our formalism, which permits, as an option, that constraints be declared explicitly as part of the static schema, to be directly enforced by the implemented algorithms.

A genre is of course compatible with an ample choice of (valid) initial states. Different intitial states lead to the development of possibly very different narratives, all of which are constrained to remain within the limits of the defined genre. The initial state assumed in this paper is described in appendix II. One may notice in particular that the villainous Draco is stronger than the two `knights`, of which Brian is somewhat better provided, and that the potential `victim`, princess Marian, is indifferent to both `knights`, despite their perfect love (`100` in `affection`) for her. True to his role as `donor`, Turjan the Archmage is as enigmatic as one would expect, neither good nor evil (`0` for `nature`).

The *closed world assumption*, familiar to database practioners (corresponding to the logic notion of *negation as failure*), justifies the conclusion that no one is `married` or `kidnapped` at the inital state, simply because no such facts are explicitly recorded in the database.

3. The dynamic schema

3.1. Informal presentation of example

In our limited Swords and Dragons genre, actions are mostly physical. Heroes, villains and even victims are able to fight, and take measures to raise their chance of victory. Before engaging in personal battle, a character often has to penetrate through the group of guardians surrounding the other character's present location, sometimes quite hard to transpose, unless the latter foolishly dismisses a number of them. And the combat proper will consume the energies of a fighter. Is his (or her or its) strength enough to defeat and kill the adversary? If not, it is advisable to seek a powerful magician to obtain a surplus of fighting power.

But, as donors tend to be in folktales, a magician is a capricious being, easily irritated when approached without courtesy. He may then pretend to yield to the hero's request but will in fact reduce his strength to the bare minimum necessary to start a combat – just to be inevitably defeated in the sequel.

Heroic knights are destined to love damsels, who in turn may not respond to their entreaties at the beginning. But, if a villain kidnaps a princess and one hero succeeds in freeing her, then gratitude and admiration may well change her inclination.

Many actions are closely associated with places. So, for a villain, to kidnap a victim means to bring her to his lair; and marriage is naturally celebrated at a church. All characters, execpt donors, continually move across the scene to accomplish their missions.

3.2. Modelling notions

The dynamic level of specification takes us from *descriptions* of states, as covered by static schemas, to *narratives*. While states are sets of facts, narratives are composed of *events*. An event is a transition from a valid state S_i to another state $S_{j,}$, which should also be valid, i.e. conforming to the established static integrity constraints. In addition, we shall require that the transition itself be valid, which means that it should obey a further set of restrictions, to be called *dynamic integrity constraints*.

A rigorous discipline, first expounded in Software Engineering research in terms of *abstract data types* [GTW] and, later, of *object-oriented models* [Em], is in general sufficient to enforce both kinds of constraints, by restricting state changes to what can be accomplished by applying a limited repertoire of pre-defined *domain-oriented operations*. The operations must be defined in such a way that, if one starts from a valid initial state, their execution will always preserve all constraints. Usually harder to handle is the requirement that the repertoire be enough to allow that *all* intended valid states be reachable.

By a fortunate coincidence, a similar notion has been proposed in Literary Theory by the Russian researcher Vladimir Propp [Pr]. In order to specify the genre of fairy-tales, he described a set of 31 *functions*, comparable to what we shall keep calling (domain-oriented, or, more appropriately here, genre-oriented) *operations*, which he claimed to be enough to account for a large sample extracted from an anthology of fairy-tales compiled by Alekxandr Afanas'ev [Af].

From now on, we equate the notion of *event* with the state-change brought about by the *execution of an operation by some agent*. This has proved convenient for our purposes, although, admitedly, it may not be sufficient in certain contexts. For example, events caused by natural phenomena cannot be contemplated, at least not directly.

In order to formally specify operations, the *STRIPS* (from *Stanford Research Institute Problem Solver*) [FN] method is very convenient, both for real-life domains and for fictional genres. Each operation is defined in terms of its *pre-conditions* and *post-conditions*. Pre-conditions are conjunctions of positive (say `f`) or negative (`not f`) facts, which must hold at the state in which the operation is to be executed. Post-conditions (or *effects*) consist of two sets of facts: those to be asserted and those to be denied as a consequence of executing the operation. Integrity preservation depends on a careful adjustment of the interplay among pre-conditions and post-conditions over the entire repertoire of operations.

This interplay has an even more important consequence, which is to establish a partial order for the execution of operations, since, if the pre-conditions of an operation O_1 may

only be satisfied by the post-conditions of another operation O_2, then O_2 should be executed before O_1. This suggests, in turn, a *backward chaining* inferential strategy for generating *plans*, a subject to be treated in section 4.2. But, at this point, two comments are in order: (a) *logic programming*, as offered by Prolog, is fully adequate for developing such planning algorithms based on logic inference; (b) *constraint programming* algorithms (available, as said before, in the version of Prolog that we use) provide a very powerful complement to logic inference for handling numerical attributes.

The notation for declaring the *signature* of operations should be extended, in order to associate pragmatic information, especially agency, with the parameters. For this purpose, Fillmore's *case grammar* proposal [Fi] is applicable. Various choices of cases have been listed by different authors. Those to be employed here are: `agent`, `coagent`, `recipient`, `patient`, `object`, `destination`. In the parameter-list of an operation, each parameter is characterized by case, paired with either the entity class or role involved. Indicating a role, instead of an (entire) entity class, limits the participation in a case to those instances of one or more entity classes to which the role has been explicitly assigned. Notice that the case `agent` (and also `coagent`) introduces an *agent-oriented* [HS] modelling view, on top of object orientation.

The dynamic schema is specified with the following syntax, wherein each operation is defined by two complementary clauses:

```
operator_frame(<operator-id>,<operator-name>,[<case>: (<entity class or
role>;...;<entity class or role>),...,<case>: (<entity class or
role>;...;entity class or role>]).

operator(<operator-id>,
     <operator-name>(<parameter list>),
     [
     <pre-conditions>
     ],
     [
      <effects>
     ],
     <estimated cost of operation>,
     [<main effects>],
     [],[]).
```

Obs.: The utilization of the two last components of the operator clause, which appear here as empty lists, will be described later (section 4.2).

3.3. Example specification

Ten operations have been provided for our Swords and Dragons genre:

```
1. go(CH,PL)
2. reduce_protection(CH,PL)
3. kidnap(CH1,CH2)
4. attack(CH,PL)
```

5. fight(CH1,CH2)
6. kill(CH1,CH2)
7. free(CH1,CH2)
8. marry(CH1,CH2)
9. donate(CH1,CH2)
10. bewitch(CH1,CH2)

All operations share one evident pre-condition: the agent must be alive. Most operations also require that the agent be not in a kidnapped status, wherein his freedom to act would be necessarily limited. And, for operations involving two characters, both must be in the same current_place. Operations involving a physical confrontation are only admitted between characters of opposite nature. A mandatory post-condition is that, when an attribute is modified to receive a new value, the list of effects always prescribes the exclusion of the old value, since all attributes are single-valued in our example. Specific characteristics for each operation are reviewed below:

1. The agent of operation go(CH,PL) can be any character, except a donor; the destination is of course a place. A pre-condition is that CH should neither be currently kidnapped (a general requirement, as said above) nor be keeping someone kidnapped. Presumably the kidnapper must be constantly vigilant, to counter any attempt towards the victim's liberation. The effect of the operation is to make PL the current_place where CH is.
2. Only the potential victim can commit the imprudence to dismiss some of the guardians of the place where she currently is, by being the agent of the reduce_protection(CH,PL) operation, whose object is a place. Previously the number of guardians serving as sentinels must be positive, and each execution of the operation reduces it by a factor of 10 (written as 10.0, given the real-number format adopted). The exact decrement is to be determined at the *dramatization* stage (see section 5.2).
3. Villain and victim are the roles assigned to CH1 and CH2, the agent and the patient, respectively, of operation kidnap(CH1,CH2). A vital pre-condition is that the strength of the villain be enough to break into the place where the victim is. The formula for the comparison says that his strength should be greater than that of his victim, added to the level of protection of the place. But the kind of protection is also taken into consideration, being multiplied by the level (remember that kind is a number, 1 or -1, to indicate whether the guardians are either on the side of goodness or of evil); as a result, if the victim is currently in a place dominated by evil, the level of protection will actually be subtracted from her strength. Kidnapping results in the victim being imprisoned in the home of the kidnapper.
4. A hero, not currently kidnapped (recall that the same individual who plays the role of hero can simultaneously be a victim), or a villain can be the agent of attack(CH,PL) intent on decimating the group of guardians protecting PL, which constitutes the object of the action. The nature of the agent must be the contrary of the kind of protection of the attacked place. The level of protection (associated with the number of guardians), which must be positive beforehand, is reduced by a factor of 30. The operation has the side-effect of displeasing those who

have their home in PL: their affection for the attacker now becomes strongly negative (-100).

5. Two characters of opposed nature, but never a donor, currently having strength of at least 10, can play the agent and coagent of fight(CH1,CH2). The level of protection of the place where the combat happens must be null or negative; so the troop protecting such locations must first suffer an attack, before the leading characters can face each other. The confrontation is extenuating for both participants, which is indicated by the mutual subtraction of their strengths as a result.
6. Agent and coagent of kill(CH1,CH2) are as in the preceding operation. The killer's strength must be strictly greater than 10; and the character killed must either no longer be able to fight or have the bare minimum necessary for that, which is expressed by requiring that his strength be at most equal to 10. The obvious effect is that CH2 is no longer alive.
7. Operation free(CH1,CH2) can be performed by a hero, to the benefit of a kidnapped victim, only after the kidnapper is dead. Besides the effect that CH2 is no longer kidnapped, the operation has the virtue to raise to the maximum value (100) the affection of the grateful victim for her liberator.
8. In our version of marry(CH1,CH2), the agent CH1 must be a hero and the coagent, CH2 a victim, usually the proverbial maiden in distress rescued by a loving knight. Their mutual affection has to be greater than 80 (note this might already be true at the initial state, but then there would be no need for heroic action). They must be single. To thus acquire the married status, their presence at the Church is required.
9. The first operation whose agent must exclusively be a donor, a role that is reserved to magicians in our genre, is donate(CH1,CH2), whereby the recipient, always a hero, is given an amount of fighting power. The measure of the new strength of CH2 depends on how he approaches the donor CH1. A courteous attitude is rewarded with an increase of 80 above his current strength, whereas rudeness, demonstrated by an attack against the defenses of the magician's home, is punished by having his strength set to the minimum required for fighting (10), regardless of what his previous rating used to be.
10. The second operation having a donor as agent, namely bewitch(CH1,CH2), has, as patient, either hero or victim, which are the two classes of characters normally endowed with a good nature. The surprising double effect of the operation is to instill an evil nature into CH2 and, at the same time, make him or her very strong (a strength of 100).

It is worksome but not too hard to check how the combined interplay of pre-conditions and post-conditions in this repertoire contributes to preserve static and dynamic integrity constraints, once the validity of the postulated initial state has been verified. As an example with a static constraint, one can readily see that, at every state reachable through the operations, the current place of the donor is invariably his home, provided that this was true at the initial state.

Killing an enemy is a task requiring wise tactics, in view of the dynamic constraints involved. If CH1 intends to kill CH2, he may or may not have to fight against the other

beforehand. Value 10 is especially critical in this regard: it is not sufficient for CH1 as prospective killer, whereas CH2 can be killed if he has this value exactly (or less than it, of course). So, there is no need to fight if CH2 already has strength 10 or less. On the other hand, 10 is the minimum required to start fighting, which may induce an ill-advised CH2 to challenge CH1 (see how the discourteous recipient is treated in the donate operation described above).

Now let us examine what happens when fighting takes place. Clearly only the situation wherein CH1 is stronger than CH2 needs to be considered. Suppose CH1 has strength 30 and CH2 has 20. As indicated as an effect of the energy-consuming fight operation, the strengths of the two opponents are subtracted from each other, so CH1 ends up with 10 and CH2 with -10. As a consequence, CH2 can now be killed – but not by CH1, who became too weak for that. (Notice that the same happens with strengths of 20 and 10 respectively, which is ironical, since in this case CH1 could have dispatched the enemy directly without further ado...).

As an even subtler dynamic constraint, observe that, once kidnapped, a victim has no way to escape from custody by her own action, inevitably needing the initiative of one or more heroes. When dealing with fiction, one is allowed to make certain assumptions that may seem unrealistic. One of the general principles governing the genesis of fictional stories is that *functional events* [Cu,Ba] should be included, plausible or not entirely so, as a prompt to adventurous deployments. As will be seen in section 4, this "maiden in distress" stituation works as an inducement for heroic quest.

In our example specification, if one starts from a valid initial state and only the nine first operations above are used, the generated plots should conform to all constraints, and be recognizable as legitimate representatives of the intended genre. The pre-conditions and post-conditions of these operations were carefully balanced for that. However, if the tenth operation – bewitch – happens to be utilized, this may no longer be true. The introduction of a disturbing element serves a purpose here: to create the possibility of transgressing some of the conventions of the genre, such as the understanding that all paticipants retain their nature throughout their lives. Again, fiction has a latitude that one would hardly admit in business application domains.

The dynamic schema is shown in appendix III. The notation is reasonably self-explanatory. Curly brackets are used to distinguish, in the lists of pre-conditions of operations, arithmetic equations and inequations that will be handled by the constraint programming algorithms. One detail deserves attention, which illustrates one way to accomodate conditional effects in the presence of pre-conditions expressed in constraint programing formulas. In operation donate(CH1,CH2) we need to establish that:

if the current strength of hero CH2 is L1, then
    if the affection LA of the donor CH1 for CH2 is null or positive, then
        the new strength L2 of CH2 will be L1 + 80
    else L2 will be 10

Checking whether `LA≥0` or `LA<0`, being a simple arithmetic comparison, does not seem to require constraint programming. But one special feature is needed here, namely *delayed evaluation* − as noted before, we do planning through backward chaining, and so variable `LA` may still be uninstantiated when the above comparisons are initially encountered. It turns out that, in the Prolog version in use, delayed evaluation is only available together with the constraint programming algorithms, which do not treat the two mutually exclusive comparisons as a mere test of current values, but as inequations expressing numerical objectives to be satisfied. If, by mischance, the first comparison to be tried is the one that happens to fail, the algorithms may involve the planner in an often intolerably long series of useless attempts to produce a value of `LA` satisfying it.

Fortunately, the intended conditional effect can be achieved by means of two unconditional purely arithmetic formulas, which are satisfiable with any possible value that `LA` may have:

(1) $\alpha = \max(0, \min(1, LA + 1))$
(2) $L2 = \alpha \times (L1 + 80) + (1 - \alpha) \times 10$

Recall that, although the Prolog text uses real-number format, we are only dealing with integers. With this consideration in mind, it is easy to see that, in expression (1):

if $LA \geq 0$ then $\alpha = 1$
if $LA < 0$ then $\alpha = 0$

and therefore only one of the summands in (2) will be non-zero, as desired.

4. The behavioural schema

4.1. Informal presentation of example

The various characters are moved to act by their inner drives. Typically, a knight like Brian is anxious to be invested with superior heroic force, so that some day he can become a dragon-slayer. In contrast, princess Marian never imagines that there may be any possibility of violence, and finds no use for the presence of so many guards around her palace.

Draco is continually in the alert for signs of a weakening in her protection, awaiting a chance to come and achieve the maiden's abduction. Attempts to kidnap may meet resistance, with considerable risk to the victim. On purpose or by accident, the dragon may end up killing his fragile prey.

Depending on the outcome of the villainy − abduction or death of the princess − one hero, or both, would feel impelled to either rescue or, at the very least, avenge her. Taken alive from captivity, she will be full of tender feelings for her saviour. Both would love each other, and thus be ready to have their marriage celebrated.

If the two knights participate of a heroic quest on behalf of the princess, they may or may not collaborate. They both love her, and are bound to compete, loyally or not, to win her hand.

Turjan, finally, does not seem to wish anything. He stands still in the forest, were people sometimes search him. The heroes come to demand a gift of fighting energy, and his reaction depends on how he is disposed toward the newcomer. Desiring nothing, he never makes any plans. But, when one less expects, he can with a gesture transmute a kind person into a powerful creature of evil.

4.2. Modelling notions

For those entity classes or roles whose instances are animated agents, there may exist *goal-inference rules*, basically of the form `rule(<situation>,<goal>)`, specifying, in a temporal modal logic formalism, the *goals* that will motivate these agents when certain *situations* occur during a narrative. The rules use the following meta-predicates to speak about the occurrence of an event or the truth value of a literal (a fact or the negation of a fact) at certain times:

- `h(T,LITERAL)`: `LITERAL` is necessarily true at time `T`;
- `p(T,LITERAL)`: `LITERAL` is possibly true at time `T`; and
- `e(T,LITERAL)`: `LITERAL` is established at time `T`; and
- `o(T,EVENT)`: `EVENT` occurred at time `T`.

In order to express constraints relating variables, there are two additional meta-predicates:

- `h(CONSTRAINT)`: `CONSTRAINT` is necessarily true; and
- `p(CONSTRAINT)`: `CONSTRAINT` is possibly true.

Having, at a given initial state, applied such rules to determine goals for the various agents, one is in a position to apply a suitable *plan generator* to start composing a plot, as a partially ordered sequences of events, where each event is associated with the execution of one of the operations defined in the dynamic schema. The (simulated) execution of the operations results in a new state wherein, again, the goal-inference rules are applied, and so on and so forth, until a state is reached where no new goal is inferred (or one arbitrarily decides to end the process). It should be stressed that a plot composed in this way can be seen as the combination of any number of individual plans, aiming at the goals of each agent, often with mutual interferences.

Willensky [Wi] has done a comprehensive study of positive and negative interferences between goals and plans of the same agent, and also between those of different agents. Negative interferences result in contradictions to be resolved, and positive interference offer optimization possibilities. In both cases, diverse strategies can be employed to find how to alter the goals and the generation of plans, in order to obtain a consistent plot, in

which even *failed* individual plans may figure. Our prototype tool provides (but no example will be shown here) two main mechanisms to handle goal abandonment and competitive plan execution: *conditional goals* and *limited goals* [CF]. A conditional goal has attached to it a *survival condition*, which the planner must check to determine whether the goal should still be pursued. Limited goals are those that have an associated *limit* (expressed as a natural number). The limit restricts the number of events that can be inserted to achieve the goal. Other strategies are being considered for future inclusion in our method.

An alternative way to derive plans for goals is to take, from a conveniently structured library, a pre-existing *typical plan*, adapting it if necessary to specific circumstances. We have been using a structure for such libraries of typical plans that also allows *plan-recognition* by a method proposed by Kautz [Ka]. The method consists of matching observed events against the plan definitions (also called *complex operations*) stored in the library, trying to find one or more plans of which these events may be part.

Our typical plans (complex operations) have the same syntax shown for (basic) operations in section 3.2. If the complex operation results from a *composition* of other possibly complex and/or basic operations, the two last parameters (shown as empty lists in the `operator` clause pattern of section 3.2) will contain, respectively, the component operations, each with a different f_i, prefix, and pairs $[f_i\text{-}f_j]$ declaring any order requirements holding between them. Complex operations formed by *generalization* are also represented, branching down to specialized operations corresponding to alternative ways to reach the same main effects; clauses `is_a(<more-specialized-operation>,<more-general-operation>)` declare this structural link.

4.3. Example specification

The first two, out of our six goal-inference rules (listed in appendix IV), are to be activated right at the initial state. Rule one refers to the `heroes`. The leading `hero`, at least, should be prepared for future missions and so, if there exists some `villain` stronger than him, there is reason to seek for an even superior `strength`. To determine who can be regarded as foremost among his peers, the rule arbitrarily chooses one `hero` with maximum `strength` from the outset. Note the compact expression `{LS>Lv}` in the goal position of the rule, where `Lv` is the `strength` of the `villain` and `LS` the new level of `strength` to be reached by the chosen `hero`; thanks to the addition of constraint programming to Prolog, it becomes possible to set up such goals, which will lead the recursive plan generation algorithm to look for a way to instantiate the variables still free (`LS` in this case), so as to satisfy the numerical objectives formulated.

The second rule applies to the `victim`. It is very common in folktales that a `victim` can be blamed as partly guilty for the villainny that she will suffer. As Propp observed, her *complicity* is revealed as she, for example, exposes herself by weakening the defenses surrounding her. Accordingly, the rule assesses the initial `level` of `protection` of the `place` where she is, and asks for its reduction. As already seen in pre-conditions and post-conditions of operations, the `nature` of the `victim` and the `type` of `protection` of the

place appear as coefficients, affecting the sign of the terms in the inequality. Note also that a different variable, PLACE1, denotes the location of the victim at future time T; this allows *two* possibilities for achieving less protection: the planner can either apply (one or more times) the reduce_protection operation to the original PLACE – in which case the two variables will be treated as identical –, or can cause the imprudent maiden to go to some different location already offering an inferior protection.

If the goal of the preceding rule is reached, the third rule is triggered, instilling in the villain a desire to take advantage of the more fragile condition of the victim, by having her kidnapped. Although this is the type of villainy that determines the normal continuation of the plot, it may happen instead (through the user intervention, as will be seen in the next section) that the villain perpetrates a different villainy, by murdering the victim. To cover this circumstance, it became necessary to add to the situation part of the rule the seemingly redundant requirement that the victim needs to be still alive if the villain proposes to have her kidnapped. Without this additional requirement, we would have a goal conflict with the fifth rule, to be reviewed soon.

The fourth rule says that, if kidnapping has occurred, the goal of reverting this situation will arise. The rule does not explicitly refer to the heroes as the necessary agents who accomplish the deed, contrary to the rule just described, which does mention the agent, namely a villain. Nevertheless, even with this apparent neglect, the present rule effectively causes one or more heroes to be recruited for the mission, because, due to the overall specification of the genre, no other character might succeed.

The fifth rule applies in a situation in which the villain has performed the action of killing the victim. All that remains for the heroes (once more not explicitly mentioned) to do is to vindicate her death, by making the villain lose his life. The rule employs the modal o operator, which denotes the execution of an indicated operation. If both this rule and rule three were activated at the same occasion a contradiction would result: the goal that the villain be not alive makes it impossible to execute operation kidnap, required to satisfy the goal of rule three. Evidently the motivating situations for the two rules are mutually exclusive and so they should never be simultaneously active, since it does not make sense to kidnap a dead victim – but we find useful to report this as a problem, to illustrate how crucial a careful analysis of the specification is. Indeed, at an early design phase, we overlooked the necessity to spell out in the situation part of rule three that the victim should be alive, and took some time to realize what was causing trouble to the plan generator.

The sixth and last rule purports to lead the plot to a happy ending: if two persons love each other with perfect love (or almost perfect, since the required affection is merely 95), and are still single, they will want to get married. That the married attribute for each person is tested in one direction only should not sound peculiar: operation marry (cf. appendix III) asserts the attribute in both directions (and, as always, we must rely on the correctness of the initial state for complete information about already married people). Note also that the combined effect of the specification clauses restrict marriage to a hero

and a victim, roles that are respectively reserved to a knight and a princess, thus enforcing the opposite gender requirement.

As to *typical plans* (or *complex operations*), we shall limit ourselves to present informally those that we have been considering, but still did not implement (therefore this part is missing from appendix IV). When included, they are expected to form a mixed is-a / part-of hierarchy in four levels (the fifth level is occupied by the basic operations, already introduced), as in figure 3. The informal description below does not supply details about parameter lists and respective case structure, pre-conditions, and post-conditions. Proceeding top-down we have:

Level 0 - adventure – Located at the root position, operation adventure has components: do_villainy, retaliate, accompany and donate; and specializes into: rescue or avenge.

Level 1 - rescue, avenge - these are the two species of adventure. The rescue variety has components: abduct, liberate, marry, accompany, donate. The other variety, avenge, has components: murder, execute, accompany, donate. While, as the figure shows, there are direct edges leading to some of the components, other components, namely accompany and donate, are *inherited* from adventure via the is-a link. Note that, for both rescue and avenge, the is-a inheritance mechanism would also indicate do villainy and retaliate as components – but the existence of direct edges to specific forms of villainy and retaliation (the pair abduct, liberate for rescue and murder, execute for avenge) in fact *overrules* the is-a non-specific paths. In other words, one can say that the choice of a villainy *preempts* the choice of the appropriate retaliation.

Level 2 - do villainy, retaliate, accompany - do villainy specializes into: abduct or murder; retaliate specializes into: liberate or execute; accompany specializes into: help or false help. Names are, as usual, a matter of personal preference, but we tried our best to select meaningful words; accompany, for example, evokes the convention, pointed out by folklorists, that certain persons who aid (or hinder) the hero in his mission march by his side (playing the role of helpers or of false heroes), while others (the typical donors) usually stay behind and take no part in the action.

Level 3 - abduct, murder, execute, liberate, help, false help - Both villainies have a first component that signals the complicity of the victim. So, abduct has components: reduce protection, attack, kidnap; while murder has components: reduce protection, attack, fight, kill. Both retaliations involve killing the villain, and include all preparatory actions which may or may not be needed in view of current circumstances. Variety liberate has components: attack, fight, kill, free, whereas execute has components: attack, fight, kill. Sincere helpers can contribute in various ways, not necessarily doing all that is listed here, and noting that kill should rather be reserved as a prerrogative of the main hero. A clever false helper is likely to join the battlefield when the struggle is over, and subreptitiously open the doors of the dungeon to the victim, thereby seducing her with an eye to matrimony. Thus, help has components: attack, fight, free. Effortless false help has components: free, marry.

We left out two basic operations from this hierarchy. Pervasive as it is when physical events are contemplated, operation `go` is in fact an ultimate component of practically all others, and therefore is assumed to be present even if not indicated explicitly. On the contrary, `bewitch` was deliberately excluded. Plots including `bewitch` are *not* to be considered typical in the context of our genre, since they reveal the `magician's` inclination to subvert an until then innocent world, by acting as a *trickster*.

A structured library with these typical plans (complex operations) is shown in figure 3. Single arrows denote composition (`part-of` link) and double arrows denote generalization (`is-a` link).

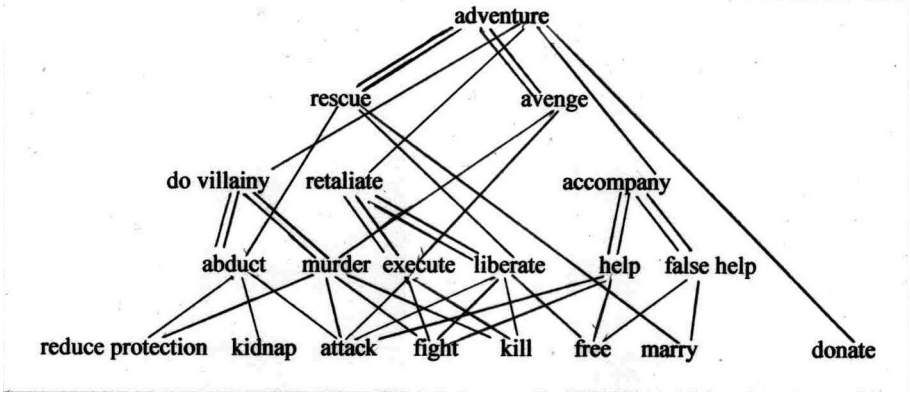

Figure 3: hierarchy of typical plans

5. Generating genre-restricted plots

5.1. Composing by adaptation

Plots are primarily developed by successively adding events in a sequence. And, as an event is being considered for addition, one may pause to consider whether a different but possibly similar event might be more appropriate for that position in the sequence. This process parallels that of forming a sentence in natural language. Thus, on the basis of this well-known homology between literary and linguistic structures, we feel justified to borrow Ferdinand Saussure's [Sa] notions of a *syntagmatic axis* and a *paradigmatic axis* to refer, respectively, to the concatenation of events, which extends "horizontally" the number of positions, and the "vertical" movement to compare and perhaps select a more suitable member from a class of events considered analogous (by some criterion) and therefore apt to occupy the position.

For composing a plot, a *modular strategy* may look attractive, as it does to software engineering practitioners when called to design large software systems. One may start (along the syntagmatic axis) aligning events corresponding to large narrative units, like the typical plans / complex operations of the previous section. In fact, these recall the "canned" *packages* that software engineers use (or *reuse*, in the sense of adapting for purposes different from those which originally led to the development of the package). At a

subsequent stage, one moves down "in depth", so to speak, to explain these broad events in terms of smaller episodes, richer in detail. Putting this strategy to use implies the existence of a third axis, which might be called the *meronymic axis* (meronymy being understood as the relation between whole objects and their parts).

Literary theory being such a rich and complex field, it would be unwise to claim that these three axes encompass the plot composition process in all conceivable dimensions. But they seem appropriate to cover what we need here for the present discussion. On the one hand they are closely associated with three out of the "four master tropes" indicated by Kenneth Burke [Bu]: *metonymy*, *metaphor*, and *synecdoche* correspond respectively to displacements along the syntagmatic, paradigmatic, and meronymic axes (metonymy is taken here in the specific sense of substitution through *contiguity* [Ko]). On the other hand, they also appear relevant in the context of *ontologies* [SS,OA], a subject of growing interest to Information Systems and Software Engineering specialists; in terms of ontologies, displacement along the three axes signifies the traversal of hierarchies of concepts, connected by `next-to`, `is-a`, or `part-of` links.

Composition along the three axes should normally proceed in a disciplined fashion, so as to preserve the conventions of the chosen genre. Curiously, the fourth and last of the main tropes listed by Burke, namely *irony*, sounds as a deviation from the right path, leading to the *transgression* of such rules, an intriguing possibility that we shall not discard here (see our previous comments on the `bewitch` operation, in section 3.3).

People who have no special talent for literary composition, like ourselves, find difficult to invent interesting plots. Storytelling researchers [Gl] repeatedly point out that there may be problems when users participating in a game are prompted to function as "authors". But we usually do not feel so uncomfortable if asked to *adapt* an existing plot, by introducing small modifications in a gradual fashion. What was said until now for composition is equally applicable for characterizing different kinds of adaptation, which can be regarded as further displacements or adjustments along each of the three axes. And transgression appears as an extreme form of adaptation, of special appeal to post-modern tastes [SW,Bl].

Our experience suggests that composing a plot by adaptation may be productive for users in general, regardless of their literary skills, especially if the development environment has the following characteristics:

- Development is not done in a single piece with a final outcome in mind, but rather in a step-wise fashion, with short-term opportunistic goals induced by situations holding at the moment.
- The user has the option to manually declare the next goals to try and/or the events to take place next.
- If he takes this option but is not aware of the types of facts, individual characters, etc. represented, and does not know what operations are supported (which are the only way to make events happen), appropriate menus are displayed.
- But he also has the option to let the "system" go on, in an automatic mode.
- And he can decide for one or for the other option at each stage.

- With the manual insertion option, the system checks whether the insertion is valid; if it is not it tries to mend it, typically by inserting further events to satisfy pre-conditions, and, if this is not possible, informs the user that his insertion had to be rejected.
- In turn, confronted with a continuation just produced by the system that does not meet his expectations, he can order it to exhibit a series of *alternative* subsequences for his choice, if such exist.
- On the other hand, if he feels as a beginner, without confidence in his ability to guide the composition process, he can, as his very first attempt to familiarize himself with the authoring craft, allow the system to proceed automatically through all stages – and then a sort of "standard" plot is produced, which he can criticize and later try to reshape to suit his preferences, well in the spirit of composing by adaptation.
- Finally, as a clue that he may or may not care to take into consideration, the system can display before his eyes the layered hierarchy of typical plans available in its library.

These facilities help extending the plot along the syntagmatic axis. Also, menu selections and the proposal of alternative paths allows paradigmatic choice. As will be seen next, the environment supplied by the **Plot Manager** module of our prototype tool offers all the above enumerated facilities, except the last one, which is currently being studied as part of our research, and should be of assistance towards the creation of plots by successive refinements, seen as a top-down movement along the meronymic axis.

However, in one specific but vital aspect, this multi-level development strategy is already handled by the **Drama Manager** module: we refer to the decomposition of the plot events into smaller-grain *actions*, adequate for coherent visualization. A `kidnap(CH1,CH2)` event, for example, is broken into actions such as: `CH1` getting closer to `CH2`, grasping `CH2`, and taking her to where he lives. The creation of a more general repertoire of detailed actions, adequate for an ample variety of genres (even if not for *all* conceivable ones), can perhaps be envisaged, starting, say, from Schank's classic *primitive actions* proposal [Sc].

5.2. Adaptation features of the implemented system

The underlying philosophy of the system consists of providing the user with efficient means for exploring coherent alternatives that the story may allow at a given state, and for guiding the plot at the level of events and characters' goals. The nucleous of the system is the **Interactive Plot Generator** (**IPG**) module, which incorporates a hierarchical plan-generation algorithm, based on Abtweak [YTW], and written in SICSTUS Prolog [CW].

In general terms, the user has direct control only over the **Plot Manager** module. This module, in turn, communicates with the **IPG** module to execute plot generation and enforce coherence, and with the **Drama Manager** module to control plot visualization. The **Plot Manager** comprises the user graphical interface (implemented in Java), whereby the user can participate in the choice of the events that will figure in the plot, and decide on their final sequence. Each event is represented by a rectangular box that may assume a specific

color according to its current status (fig. 4).

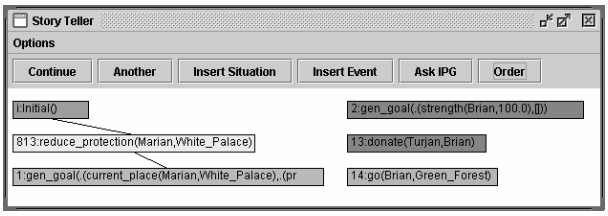

Figure 4: Interface of the **Plot Manager**

The user neither has direct control over the scene, nor over the characters themselves. Moreover, user intervention is always indirect, in the sense that any user intervention must be validated by **IPG** before being incorporated to the current plan.

Plot generation and dramatization are two separate processes, in contrast to pure character-based approaches [CCM], where user interaction affects plot structuring at real-time. This means that only during the simulation process the user has an opportunity to intervene in the creation of the plot.

As explained previously, plots result from goals that the characters aim to achieve. At each simulation step, new goals may be inferred and automatically added to the plot, which causes the insertion of a new set of events. The events inserted in the plot so far are sent to the graphical interface for user intervention via the **Plot Manager**.

The user intervention may be – with the possibility of changing the option at each step – either weak or strong. In a *weak intervention*, the user just selects partially-generated plots that seem interesting from his perspective. For this sort of user control over automatic plot generation, the **Plot Manager** offers two commands: `another` and `continue`. The command `another`, requests from **IPG** an alternative solution to achieve the same goals of the step just finished. The command `continue` asks **IPG** to try to infer new goals and resume the simulation process. These weak forms of intervention usually lead the plot to situations that the author of the story (designer of the schemas) would have devised beforehand.

For *strong intervention*, aiming at the creation of more personalized stories, the **Plot Manager** offers two complementary means. Firstly, the command `insert situation` allows users to specify situations that should occur at specific times along the plot by inserting some additional goal to be reached. The specific details of how the goal will be accomplished are left to **IPG**, which is charged to find a solution, if one exists, using the planning algorithm. It must be noted that, in view of performance considerations, a valid computable plan may fail to be obtained if the search limits currently configured in **IPG** are exceeded. As in the purely automatic generation, the user may confirm the solution (by indicating `continue`) or request an alternative (`another`), which, as said before, is a case of weak intervention. Secondly, at a lower interaction level, the user is allowed to explicitly add events to the plot with the command `insert event`. To validate the insertions, the user must invoke **IPG** through the `continue` command. At this moment, all user defined

operations are submitted to **IPG**, which runs the planning algorithm to check whether they are consistent with the ongoing plot. If not, **IPG** tries to fulfill possible unsatisfied constraints by inserting further new operations in a specific order. The user may also `remove` user defined operations that were not yet incorporated to (or were rejected by) the planner.

Whenever the user proposes to execute an `insert situation` or `insert event` command, menus are displayed to show the defined types of facts (for the insertion of goals) or operations (corresponding to events). Once he has made his choice, he can complete it by filling the parameter positions, again with the help of separate menus for possible values (in particular, `names` of `characters` and `place names`).

There is one kind of user interaction that is actually mandatory and must be done before dramatization, namely the conversion of the partially-ordered generated plan into a strict sequence, thereby completing the composition of a proper plot. Notice that, if the simulation is resumed afterwards, this addition of new temporal constraints is also an intervention, because it can affect the inference of new goals. To determine the sequence, the user connects the events in a sequential order of his choice, respecting the temporal constraints supplied by **IPG**. The plot's configuration emerges as the user moves the cursor to draw edges linking the operation boxes, starting from the root. To help the user in this process, we employ colors to distinguish operations that are already connected (yellow), operations that – in view of the temporal constraints – can be immediately connected (green) or cannot yet be connected (red). The starting root is blue and the current operation being rendered is cyan. To connect two operation boxes, the user must click with the mouse over the origin and drag over the destination (the same process is used to remove a link between two operations). Once the current plot (or part of it) is thus connected into a linear sequence, it can be dramatized by invoking the **Drama Manager** with the `render` command. The tool also offers a facility for querying the **IPG** module, through the `ask` command, about the state of any element of the narrative at a specific time T_i, using our temporal modal logic. This feature is helpful for advanced users to find out, for instance, why an operation or goal is not being allowed, and for authors intent on revising and tuning the story requirements.

With respect to the axes along which a plot is composed, mentioned in section 5.1, it is clear that the `continue` and the two `insert` commands effect the extension of plots along the *syntagmatic* axis, whereas the `another` command and the menus showing available alternatives for insertion provide support for *paradigmatic* choice. A more far-reaching capability, still under study, has to do with the *meronymic* axis. The user may find helpful to inspect a library of typical plans (cf. fig. 3), associated with the genre he is dealing with at the moment, as a clue from which he may extract inspiration while guiding the system to compose a plot of his liking. If the structured library is large and/or somewhat intricate, it may be convenient to display only two levels at a time (one root node and its leaves); the user would traverse the structure by indicating an upward move from the root, or a downward move from a selected leaf; a sideways move (comparable to what the `another` command now provides) should be possible wherever `is-a` links give access to alternatives.

An even more selective way for a user to access the library is also being contemplated. He would mark one or more events already inserted and/or being considered for insertion, and then ask the *plan recognition algorithm* (based on Kautz's method and already available to us in a separate but fully compatible implementation) to *match* these events, as *observations*, against the library in an attempt to identify one or more typical plans subsuming them. For example, the list of observations `[kill('Brian', 'Draco'), marry('Brian', 'Marian')]` fits in the `rescue` plan only, whereas `[attack('Brian', 'Red_Castle'), kill('Brian', 'Draco')]` fits in both `rescue` and `avenge` plans and thus suggests two alternative ways to structure the narrative from which the user may draw his preferences. If outside references are consulted, a library can grow much beyond what the designers specifying a genre could conceive by themselves. For folktales, for example, there is the monumental *Index* elaborated by Aarne with Thompson's participation [Aa]. *Themes* and *motifs*, as fragments of typical plans that might then be put together as part of user-composed plots, have always been an inexhaustible source of inspiration for novice and even experienced authors.

5.3. Examples of interactive step-wise plot composition

The first example is run in a fully automatic way. The user merely keeps pushing the `continue` button to let the system execute four stages of plot generation. As the button is pushed for a fifth time, the message `'Nothing to do!'` shows on the screen. Figure 5 displays this sort of "standard plot", after the user has connected the event boxes to indicate the total order of his preference.

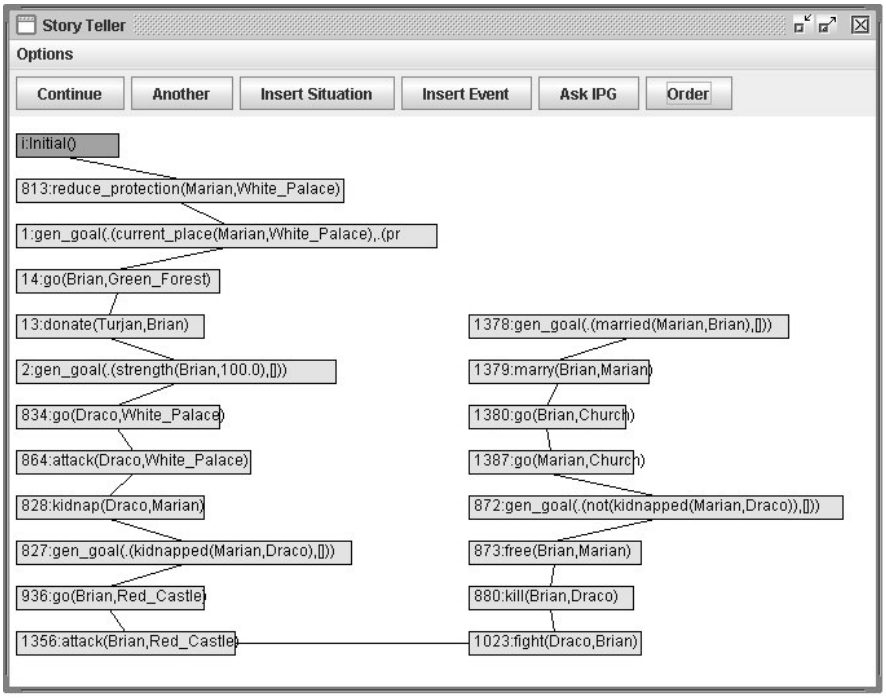

Figure 5: The standard plot

Upon traversing the plot, a simple-minded template-based facility can "read" it and produce the coarse text of figure 6 (notice that the first two scenes appear in a reversed sequence, which is alright, since they are independent of each other, and might even be thought to happen simultaneously):

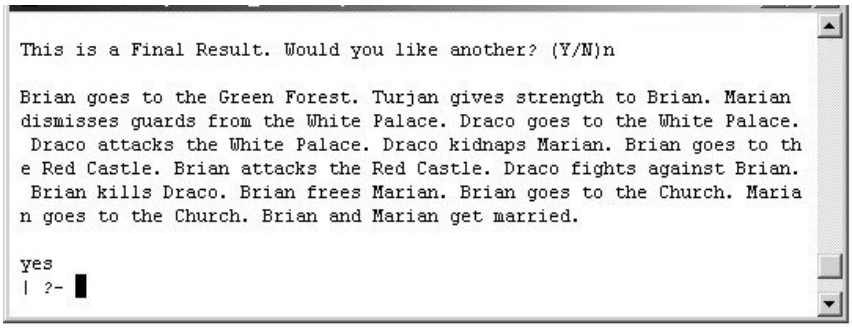

Figure 6: Template-based text

which the user may wish to rewrite in a somewhat less compact style:

> Feeling that her freedom to move around is being curtailed by the excessive care of her guardians, princess Marian decides to dismiss some of them. Meanwhile, in the Gray Castle, sir Brian finds out to his dismay that Draco is again lurching in its neighbouring scarlet abode, and that his own no more than human strength is no match for that of the dragon. What if it attempts some dreadful act? The worthy knight journeys to the Green Forest, in search of the magician Turjan. He approaches the silent mage with due courtesy, and is rewarded with a generous gift of fighting power: he is now prepared to face any challenge. He would soon be called to duty! Sensing that the charming princess has imprudently lowered her guard, Draco flies to the White Palace, shatters its defenses and elopes with the maiden. Still in the recesses of the Green Forest, Brian is warned that his beloved one is in the clutches of the enemy. He promptly invades the Red Castle, braving the resistance of its host of living skeletons, fights against the dragon, and utterly destroys the winged abomination. Marian is free, and her heart melts at the sight of her liberator. They seal a much desired love pact, blessed by the Church, and live happily ever after.

The second trial starts form the same initial state exactly. After letting the system go automatically for one stage, thus causing the princess to become more fragile and the main hero more vigorous, the user assumes personal control, leading the second hero, Hoel, to imitate his companion. The user's detailed interventions follow (and figure 7 shows the complete generated plot):

Push `continue` once, and then:

insert event	go('Hoel','Green_Forest')
insert event:	attack('Hoel','Green_Forest')
insert event:	donate('Turjan','Hoel')
insert event:	go('Hoel','White_Palace')
insert event:	go('Draco','White_Palace')
insert event:	attack('Draco', 'White_Palace')
system adds:	attack('Draco', 'White_Palace')
insert event:	kill('Draco','Hoel')

insert event: fight('Marian','Draco')
insert event kill('Marian','Draco') – system: `'Expand limit exceeded!'`
insert event kill('Draco','Marian')
`continue` gives: go('Draco','Green_Forest')
fight('Draco','Brian')
kill('Brian','Draco')

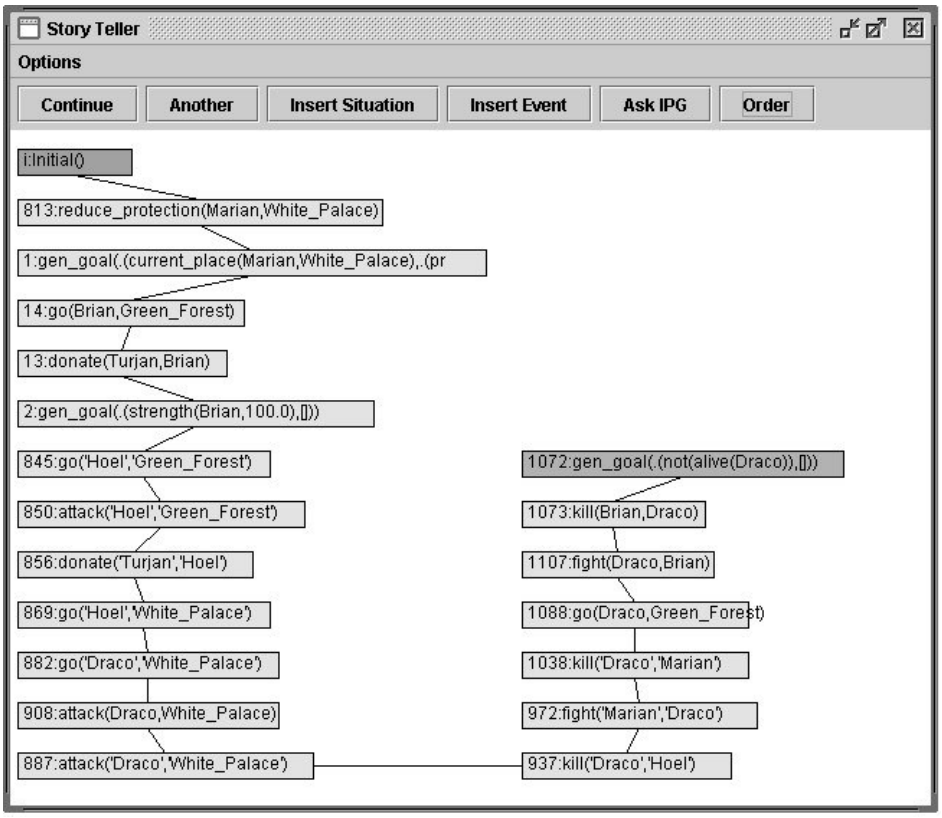

Figure 7: Discourteous hero and vengeance mission

The events added in obedience to the user's directives, after Marian had foolishly reduced her protection and Brian had harnessed power for the future, can be thus interpreted:

> Sir Hoel also proceeds to the Green Forest, expecting to receive the same gift of fighting energy. However he approaches the donor with rash violence, slashing at the trees with his sword, in his haste to open a shorter pathway. Turjan is enraged. He feigns to meet Hoel's demand, but in fact reduces his strength to a minimum: the discourteous knight might still be able to engage an enemy in combat, but should certainly be overpowered. Attracted by the news of the vulnerable condition of the princess, Draco arrives at the White Palace. He first encounters Hoel and readily dispatches him. He then comes to the princess. But Marian opposes an unexpected resistance, and, in the course of her struggle with the dragon, she is deadly wounded. In the Green Forest, sir Brian hears that Marian has been murdered. By a clever ruse he entices Draco to come to him, confident that this knight would be no harder to eliminate than the other. They fight and Brian is victorious, taking mortal vengeance for Draco's double villainy.

Before starting the third run, the user changes, in the initial state, Brian's `strength` to `60` and Hoel's to `40` (`60.0` and `40.0` in real number format). The first move is to push `continue` three times. At this point, Brian is the only hero in the story. At each time the user repeatedly pushes `another`, the following changes are observed, successively:

1: `fight('Draco','Brian')` becomes `fight('Brian','Draco')`
2: Hoel starts to participate, with `attack`, `fight` and `free`; `kill` is for Brian
3,4,5: Hoel does some of these actions; `kill` is always reserved to Brian
6: Hoel only does `attack` and `free`
7: Hoel only does `free`

From 2 to 6, Hoel figures as a well-intentioned *helper* (fig. 8 corresponds to 6).

After 7, the user keeps pushing the `continue` button, until the system signals '`Nothing to do!`' . Now one has Hoel as a *false hero* in Propp's terminology: Brian does all the effort, while Hoel just comes when the enemy has been exterminated, and then, by posing as the one who brings out the princess from prison (the isolated `free` event), he claims her hand as reward (fig. 9).

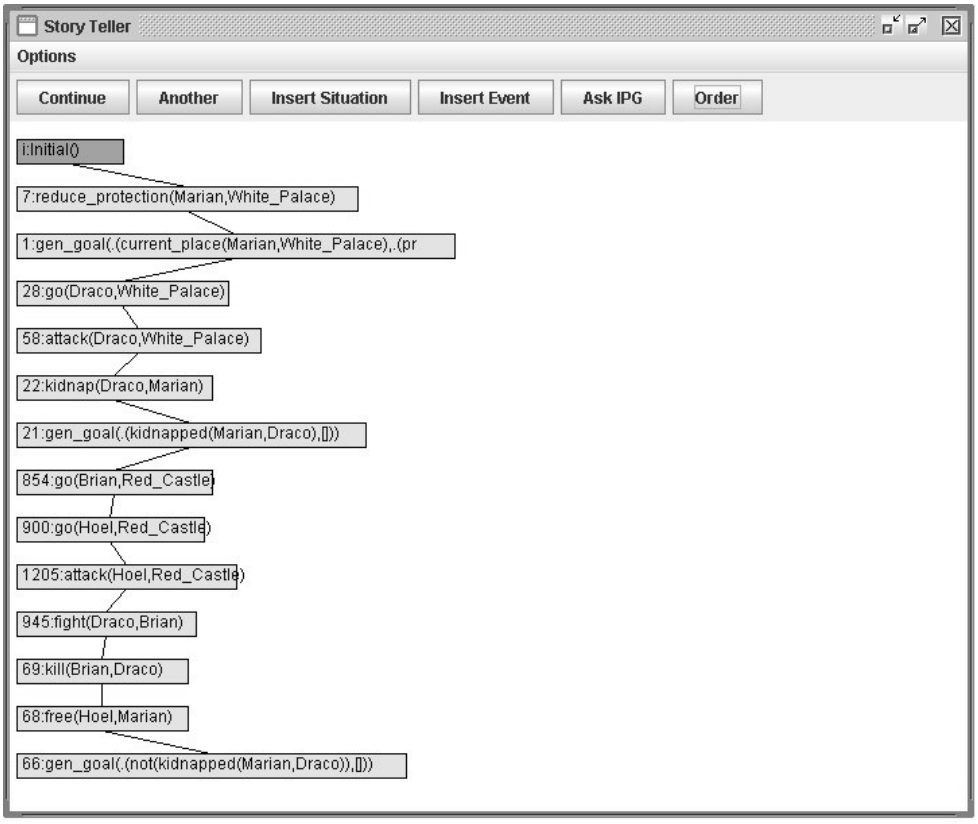

Figure 8: Companion hero as helper

Figure 9: False hero

The last experiment is far more radical than the others, since it involves a transgression against an assumed basic convention of the genre: the invariance of the `nature` of the participants.

Starting from the same initial state of the preceding run, the user pushes `continue` until the system replies '`Nothing to do!`'. But then, as if unsatisfied, the user takes over (figure 10 has the resulting plot):

insert event: go('Marian','Green_Forest')
insert event: bewitch('Turjan','Marian')
insert event: go('Marian','Church')
insert situation: not(alive('Brian'))
`continue` gives: fight('Marian','Brian')
kill('Marian','Brian')

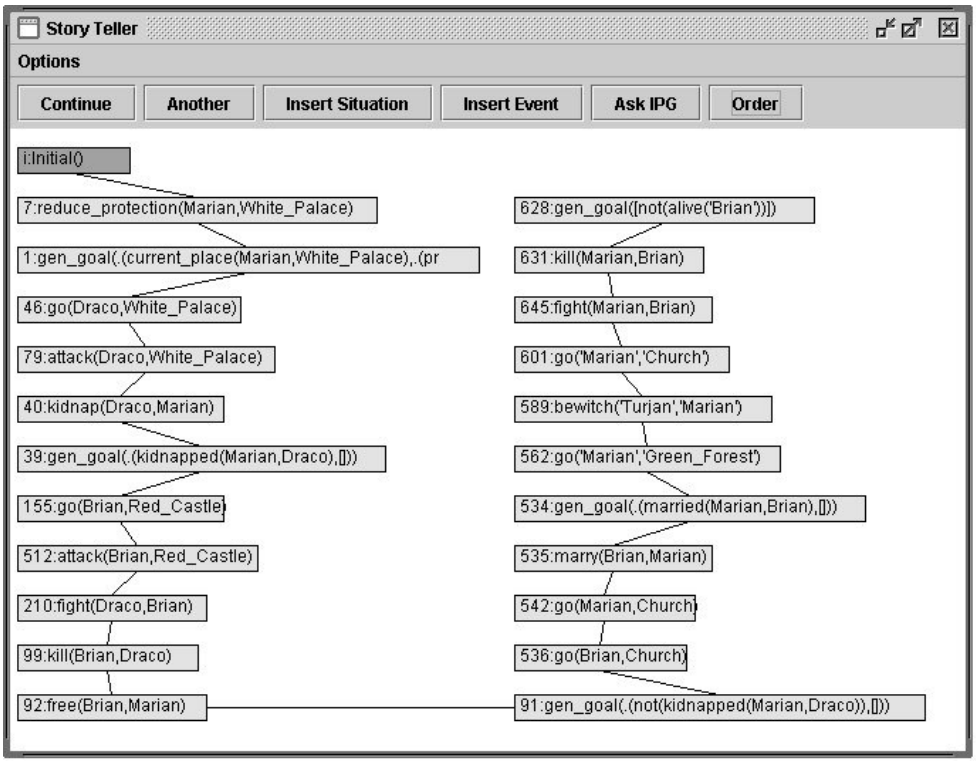

Figure 10: Mischievous princess

The magic transfomation of the princess made possible this unorthodox outcome. Thus, after what seemed the culminating event, namely the marriage of the leading characters, a dissonant sequence ensues:

> Soon after the wedding ceremony, princess Marian seems to hear a soundless appeal, calling her from the Green Forest. Unnoticed by her consort, she leaves the Church and steps into the forest, attracted by Turjan's spell. The enchanter extends his hands in her direction, casting a charm that transfigures her inmost essence. Full of malevolent power, she now rejoins the unsuspecting knight, not yet recovered from the struggle with Draco, and grabs him with unrelenting fury. Brian falls dead to the ground, dazed before his dearest one, destroyed by a fate he could not comprehend.

Or, in short, 'they did not live happily ever after'...

6. Concluding remarks

Following in Propp's footsteps, we proposed to extend his approach, originally restricted to fairy-tales, so as to be able to define literary genres in general. In contrast to *grammar-driven* methods [Ru], which are predominantly concerned with purely *syntactical* aspects, our three-schemata conceptual modelling method is based on a *plan-recognition/ plan generation paradigm* [FC1], which covers the *semantic* and *pragmatic* aspects as well. Relating the mini-world factual description, provided by the static schema, with the definition of events, wherein pre-conditions and post-conditions are expressed in terms of such facts, we are able to determine the *meaning* of a plot, by simulating the successive

state changes thereby induced in the mini-world. And plots do not emerge by blind chance; because they are set in motion by the goals of the participants, plots are *intentional* sequences of actions, coherent with the different inclinations of the characters involved.

In addition, their goals exhibit a mutual dependence determined by the often peculiar, sometimes even strange, conventions of a fictional genre. The role – a notion that is duly stressed in our model – played by each character largely determines what kind of conduct is expected from him, which in turn can only be deployed if the other characters also act as they are supposed to, always in accordance with their assigned roles. Without this careful orchestration of goals, as we tried to achieve with the six goal-inference rules for our simple Swords and Dragons genre, the plots would fail to converge towards a satisfactory outcome. Culler's insightful observation is helpful here [Cu, page 209]: "The plot is subject to teleological determination: certain things happen in order that the *récit* may develop as it does" – and he proceeds quoting Genette's allusion to the "paradoxical logic of fiction", which requires that every unit of a story be defined by its functional qualities, among which are correlations with other units.

And yet some limitations of our proposal must be acknowledged. It seems adequate for characterizing genres where the stories exhibit a high degree of regularity, but would not cope with the complexities of genres wherein the degree of variability is high. And, even for a genre that can be treated, it would be presumptuous to claim that our specification would correspond exactly to the intuition of ordinary readers. With Chomsky grammars, it makes sense to define a language L as the set of sentences that can be parsed or generated by a grammar G^L, where L may or may not have independent significance (e.g. as a natural language). Likewise, we can speak of a genre G* merely as the set of plots P that our plan-based specification can recognize or generate. Surely we would still try, as much as possible, to assess its *closure*, either through logical induction or by running experiments, in comparison with the intended scope of the target genre G. Completeness proofs are in general harder than proofs of correctness.

An interdisciplinary approach, such as ours, opens promising perspectives. We were glad to find how models of literary origin (especially Propp's functions) can naturally combine with models familiar to computer scientists (such as the ER model, STRIPS, object and agent orientation, etc.). In fact we came to realize that, besides (sufficiently regular) literary genres, the application domains covered by most business information systems offer an excellent opportunity for the application of the very same methods. Systems, such as banking, are obviously constrained by providing a basically inflexible set of operations and, generally, by following strict and explicitly formulated rules.

Our project initiated having plots, rather than textual narratives, in mind. We have already started to address the creation of texts from plots, still needing much investment in Computational Linguistic techniques to improve their quality [FC2]. Our efforts are now mainly concentrated on the continuing development of our tool for interactively generating and dramatizing stories, through alternating stages of goal inference, planning, user intervention and 3D visualization [CPFF].

References

[Aa] Aarne, A. *The Types of the Folktale: A Classification and Bibliography*. Translated and enlarged by Thompson, S., FF Communications, 184, Helsinki: Suomalainen Tiedeakatemia, 1964.

[Af] Afanas'ev, A. *Russian Fairy Tales*. N. Guterman (trans.), New York: Pantheon Books, 1945.

[Ba] Bal, M. Narratology - *Introduction to the Theory of Narrative*. University of Toronto Press, 2002.

[Bl] Bloom, H. *A Map of Misreading*, Oxford University Press, 2003.

[Bu] Burke, K. *A Grammar of Motives*, University of California Press, 1969.

[CCM] Cavazza, M., Charles, F., and Mead, S. "Character-based interactive storytelling". *IEEE Intelligent Systems*, special issue on AI in Interactive Entertainment, 17(4):17-24, July 2002.

[CF] Ciarlini, A., and Furtado, A. "Understanding and Simulating Narratives in the Context of Information Systems". *In Proc. ER'2002 – 21st. International Conference on Conceptual Modelling*, Tampere, Finland, oct. 2002.

[CPFF] Ciarlini, A., Pozzer, C., Furtado, A., Feijó, F. *A Logic-Based Tool for Interactive Generation and Dramatization of Stories*, Dept. de Informática, Pontificia Universidade Católica do R.J., monograph 07 , 2005.

[Cu] Culler, J. *Structuralist Poetics: Structuralism Linguistics and the Study of Literature*, London: Routledge & K. Paul, 1977.

[CW] M. Carlsson, M., Widen, J. *SICSTUS Prolog Users Manual, Release 3.0.* Swedish Institute of Computer Science, 1995.

[Em] Embley, D. *Object Database Development: Concepts and Principles*, Addison-Wesley, 1997.

[EN] Elmasri, R., Navathe, S. *Fundamentals of Database Systems*, Addison Wesley, 2003.

[FC1] Furtado, A., Ciarlini, A. "The Plan Recognition / Plan Generation Paradigm". In *Information Systems Engineering: State of the Art and Research Themes* - Solvberg, A., Brinkkemper, S., Lindencrona, E. (eds.), Springer, 2000.

[FC2] Furtado, A., Ciarlini, A. "Generating Narratives from Plots Using Schema Information" . In *Proc. NLDB'00 Applications of Natural Language to Information Systems*, Versailles: France, 2000.

[Fi] Fillmore, C. "The Case for Case". In: Bach, E., Harms, R. (eds.), *Universals in Linguistic Theory*, Holt, Rinehart and Winston, 1968.

[FN] Fikes, R. E. Nilsson, N. J. "STRIPS: A new approach to the application of theorem proving to problem solving", *Artificial Intelligence* , 2(3-4), 1971.

[Gl] Glassner, A. *Interactive Storytelling*, Natick: A K Peters, 2004.

[GTW] Goguen, J. A., Thatcher, J. W., Wagner, E. G. "An initial algebra approach to the specification, correctness and implementation of abstract data types". In *Current Trends in Programming Technology*, Yeh, R. T. (ed.), Prentice-Hall, 1978.

[HS] Huhns, M., Stephens, L. "Multiagent Systems and Societies of Agents", In *Multiagent Systems – a Modern Approach to Distributed Artificial Intelligence*, G. Weiis (ed.), MIT Press, 2000.

[Ka] Kautz, H. A. "A Formal Theory of Plan Recognition and its Implementation". In Allen, J. F. et al (eds.), *Reasoning about Plans*, San Mateo: Morgan Kaufmann, 1991.

[Ko] Koch, P. "Frame and Contiguity. On the Cognitive Basis of Metonymy and Certain Types of Word Formation" In Panther, K. Radden, **G.** (eds.), *Metonymy in Language and Thought,* Amsterdam: John Benjamins, 1999.

[OA] Ozcan, R., Aslandogan, Y. *Concept Based Information Access Using Ontologies and Latent Semantic Analysis*, Dept. of Computer Science and Engineering, University of Texas at Arlington, Technical Report 8, 2004.

[Pr] Propp,V. *Morphology of the Folktale*, Laurence Scott (trans.), Austin: University of Texas Press, 1968.

[Ru] Rumelhart, D. E. "Notes on a schema for stories". In Bobrow D. G. and Collins A. M. (eds) *Representation and understanding: Studies in cognitive science*, New York: Academic Press, 1975.

[Sa] Saussure, F. *Cours de Linguistique Générale*, Paris: Payot, 1967.

[SA] Schank, R.C., Abelson, R. P. *Scripts, Plans, Goals and Understanding*, Hillsdale: Erlbaum, 1977.

[Sg] Sgouros, N.M. "Dynamic generation, managing and resolution of interactive plots". *Artificial Intelligence*, 107, pp. 29-62.

[SS] Staab, S., Studer, R. (eds.) *Handbook on Ontologies*, Springer, 2004.

[SW] Selden, R., Widdowson, P. *A Reader's Guide to Contemporary Literary Theories*, The University Press of Kentucky, 1993.

[Ve] Velasquez, J. D. "Modeling emotions and other motivations in synthetic agents". In *Proc. of the Fourteenth National Conference on Artificial Intelligence*, Providence, 10-15, 1997

[Wi] Willensky, R. *Planning and Understanding - a Computational Approach to Human Reasoning*, Addison-Wesley, 1983.

[YTW] Yang, Q., Tenenberg, J., Woods, S. "On the Implementation and Evaluation of Abtweak". In *Computational Intelligence Journal*, Vol. 12, Number 2, Blackwell Publishers 295-318, 1996.

Appendix I

```
/* STATIC SCHEMA */

entity(character,name).
entity(person,name).
entity(knight,name).
entity(princess,name).
entity(magician,name).
entity(dragon,name).
entity(place,place_name).

is_a(person,character).
is_a(knight,person).
is_a(princess,person).
is_a(magician,person).
is_a(dragon,character).

attribute(character,nature).
attribute(character,strength).
attribute(character,alive).
attribute(place,protection).

boolean(alive).
composite(protection,[kind,level]).

relationship(home,[character,place]).
relationship(current_place,[character,place]).
relationship(acquaintance,[character,character]).
relationship(married,[person,person]).
relationship(kidnapped,[person,character]).

attribute(acquaintance,affection).

role(hero,knight).
role(victim,(princess;knight)).
role(villain,(dragon;knight)).
role(donor,magician).
```

Appendix II

```
/* INITIAL STATE */

/* entity instances and their attributes */

db(knight('Brian')).
db(knight('Hoel')).
db(princess('Marian')).
db(magician('Turjan')).
db(dragon('Draco')).

db(nature('Brian',1.0)).
db(nature('Hoel',1.0)).
db(nature('Marian',1.0)).
db(nature('Draco',-1.0)).
db(nature('Turjan',0.0)).

db(strength('Brian',20.0)).
db(strength('Hoel',15.0)).
db(strength('Draco',45.0)).
db(strength('Marian',10.0)).
db(strength('Turjan',45.0)).

db(alive('Marian')).
db(alive('Brian')).
db(alive('Draco')).
db(alive('Hoel')).
db(alive('Turjan')).

db(place('White_Palace')).
db(place('Red_Castle')).
db(place('Gray_Castle')).
db(place('Green_Forest')).
db(place('Church')).

db(protection('White_Palace',[1.0,70.0])).
db(protection('Red_Castle',[-1.0,20.0])).
db(protection('Gray_Castle',[1.0,0.0])).
db(protection('Green_Forest',[0.0,20.0])).
db(protection('Church',[1.0,0.0])).

db(acquaintance([CH1,CH2])) :-
  db(character(CH1)), db(character(CH2)), dif(CH1,CH2).


/* relationship instances and their attributes */
/* note: not all values of the affection attribute are given */

db(home('Brian','Gray_Castle')).
db(home('Hoel','Gray_Castle')).
db(home('Marian','White_Palace')).
db(home('Draco','Red_Castle')).
db(home('Turjan','Green_Forest')).
```

```
db(current_place('Brian','Gray_Castle')).
db(current_place('Hoel','White_Palace')).
db(current_place('Marian','White_Palace')).
db(current_place('Draco','Red_Castle')).
db(current_place('Turjan','Green_Forest')).

db(affection(['Brian','Marian'],100.0)).
db(affection(['Hoel','Marian'],100.0)).
db(affection(['Marian','Brian'],0.0)).
db(affection(['Marian','Hoel'],0.0)).
db(affection(['Marian','Draco'],0.0)).
db(affection(['Turjan','Brian'],0.0)).
db(affection(['Turjan','Hoel'],0.0)).
db(affection(['Draco','Brian'],0.0)).
db(affection(['Draco','Hoel'],0.0)).
db(affection(['Brian','Draco'],0.0)).
db(affection(['Hoel','Draco'],0.0)).


/* Roles of the agents */

db(hero('Brian')).
db(hero('Hoel')).
db(victim('Marian')).
db(villain('Draco')).
db(donor('Turjan')).


/* a general ER rule */

db(X) :-
  \+ var(X),
  entity(E,_),
  X =.. [E,V],
  is_a(E1,E),
  Y =.. [E1,V],
  db(Y).
```

Appendix III

```
/* DYNAMIC SCHEMA */

operator_frame(1, go, [agent:(hero;victim;villain),destination:place]).
operator_frame(2, reduce_protection, [agent:victim,object:place]).
operator_frame(3, kidnap, [agent:villain,patient:victim]).
operator_frame(4, attack, [agent:(hero;villain;victim),object:place]).
operator_frame(5, fight, [agent:(hero;villain;victim),
  coagent:(hero;villain;victim)]).
operator_frame(6, kill, [agent:(hero;villain;victim),
  patient:(hero;villain;victim)]).
operator_frame(7, free, [agent:hero,patient:victim]).
operator_frame(8, marry, [agent:(hero;victim),coagent:(hero;victim)]).
operator_frame(9, donate, [agent:donor,recipient:hero]).
operator_frame(10, bewitch, [agent:donor,patient:(hero;victim)]).


operator(1,
      go(CH,PL1),
      [
      alive(CH),
          not(kidnapped(_,CH)),
          not(kidnapped(CH,_)),
          current_place(CH,PL0),
          dif(PL0,PL1)
      ],
      [
          not(current_place(CH,PL0)),
          current_place(CH,PL1)
      ],
      10,
      [current_place(CH,PL1)],
      [],[]) :-
          db(character(CH)),
          db(nature(CH,KIND)),
          dif(KIND,0.0),
          db(place(PL1)).

operator(2,
      reduce_protection(VIC,PL),
      [
         current_place(VIC,PL),
         protection(PL,[KIND,LPROT]),
         nature(VIC,KIND),
        { LPROT>0.0, LPROT1=LPROT-10.0 }
      ],
      [
        not(protection(PL,[KIND,LPROT])),
        protection(PL,[KIND,LPROT1])],
      10,
      [protection(PL,[KIND,LPROT1])],
      [],[]):-
          db(victim(VIC)),
          db(place(PL)).
```

```
operator(3,
      kidnap(VIL,VIC),
      [
           alive(VIC), alive(VIL),
           nature(VIC,KIND1),
           not(kidnapped(VIC,_)),
           strength(VIC,VIC_S),
           current_place(VIC,PL),
           protection(PL,[KIND2,LP]),
           strength(VIL,VIL_S),
           current_place(VIL,PL),
           dif(PL,PL1),
           {VIL_S>VIC_S+LP*KIND1*KIND2}
      ],
      [
            kidnapped(VIC,VIL),
            not(current_place(VIC,PL)),
            not(current_place(VIL,PL)),
            current_place(VIC,PL1),
            current_place(VIL,PL1)
      ],
      10,
      [kidnapped(VIC,VIL)],
      [],[]) :-
          db(victim(VIC)),
          db(villain(VIL)),
          db(home(VIL,PL1)).

operator(4,
        attack(CH,PL),
        [
         alive(CH),
         not(kidnapped(CH,_)),
         current_place(CH,PL),
         protection(PL,[KIND2,L_PROT]),
         dif(KIND1,KIND2),
          {
            L_PROT>0.0,
            L_PROT1 = L_PROT-30.0
          },
          affection([CH1,CH],La)
        ],
        [
          not(protection(PL,[KIND2,L_PROT])),
          protection(PL,[KIND2,L_PROT1]),
          not(affection([CH1,CH],La)),
          affection([CH1,CH],-100.0)
        ],
        10,
        [protection(PL,[KIND2,L_PROT1])],
        [],[]):-
        (
         db(hero(CH));
         db(villain(CH))
         ),
        db(nature(CH,KIND1)),
        db(place(PL)),
```

```
        db(home(CH1,PL)).

operator(5,
        fight(CH1,CH2),
        [
           alive(CH1), alive(CH2),
           nature(CH1,KIND1),
           nature(CH2,KIND2),
           dif(KIND1,KIND2),
           dif(KIND1,0.0), dif(KIND2,0.0),
           strength(CH1,LS1), strength(CH2,LS2),
           {
             LS1>=10.0, LS2>=10.0
           },
           current_place(CH2,PL), current_place(CH1,PL),
           protection(PL,[KIND3,L_PROT]),
           {
             L_PROT=<0.0,
             NEW_LS1=LS1-LS2,
             NEW_LS2=LS2-LS1
           }
        ],
        [
           not(strength(CH1,LS1)), not(strength(CH2,LS2)),
           strength(CH1,NEW_LS1), strength(CH2,NEW_LS2)
        ],
        10,
        [strength(CH1,NEW_LS1), strength(CH2,NEW_LS2)],
        [],[]):-
        db(character(CH1)),
        db(character(CH2)).

operator(6,
      kill(CH1,CH2),
      [
       alive(CH1), alive(CH2),
       not(kidnapped(CH1,_)),
       nature(CH1, KIND1),
       nature(CH2, KIND2),
       dif(KIND1,KIND2),
       dif(KIND1,0.0), dif(KIND2,0.0),
       strength(CH1,LS1), strength(CH2,LS2),
       current_place(CH1,PL), current_place(CH2,PL),
       protection(PL,[KIND3,L_PROT]),
       {
        L_PROT*KIND3*KIND2=<0.0,
        LS2=<10.0, LS1>10.0
       }
      ],
      [not(alive(CH2))],
      10,
      [not(alive(CH2))],
      [],[]) :-
          db(character(CH1)),
          db(character(CH2)).
```

```
operator(7,
      free(HERO,VIC),
      [
        alive(HERO), alive(VIC),
        kidnapped(VIC,VIL), not(alive(VIL)),
        current_place(VIC,PL), current_place(HERO,PL),
       affection([VIC,HERO],LA)
      ],
      [
            not(kidnapped(VIC,VIL)), not(affection([VIC,HERO],LA)),
            affection([VIC,HERO],100.0)
         ],
      10,
      [not(kidnapped(VIC,VIL))],
      [],[]) :-
          db(hero(HERO)),
          db(victim(VIC)).

operator(8,
         marry(CH1,CH2),
      [
           alive(CH1), alive(CH2),
           affection([CH1,CH2],L1),
           {L1>80.0},
           affection([CH2,CH1],L2),
           {L2>80.0},
           current_place(CH1,'Church'),
           current_place(CH2,'Church'),
           not(married(CH1,_)),
           not(married(CH2,_))
      ],
      [
           married(CH1,CH2), married(CH2,CH1)
         ],
      10,
      [married(CH1,CH2), married(CH2,CH1)],
      [],[]) :-
          db(hero(CH1)),
          db(victim(CH2)).

operator(9,
         donate(CH1,CH2),
          [
            current_place(CH2,PL),
            alive(CH1),
            alive(CH2),
            affection([CH1,CH2],LA),
            strength(CH2,L1),
            {Alpha = max(0.0,min(1.0,LA+1.0))},
            {L2=Alpha*(L1+80.0)+(1.0-Alpha)*10.0}
          ],
         [
            not(strength(CH2,L1)),
            strength(CH2,L2)
          ],
      10,
      [strength(CH2,L2)],
```

```
      [],[]) :-
          db(donor(CH1)),
          db(home(CH1,PL)),
          db(hero(CH2)).

operator(10,
         bewitch(CH1,CH2),
      [
           nature(CH2,1.0),
           strength(CH2,LS),
           current_place(CH2,PL),
           alive(CH1),
           alive(CH2)
       ],
      [
           not(nature(CH2,1.0)),
           nature(CH2,-1.0),
           not(strength(CH2,LS)),
           strength(CH2,100.0)
           ],
      10,
      [nature(CH2,-1.0)],
      [],[]) :-
          db(donor(CH1)),
          db(home(CH1,PL)),
          db(character(CH2)).


/* templates for preparing legends */

template(go(CH,PL), [CH,' goes to the ',PL]).

template(reduce_protection(VIC,PL), [VIC,' dismisses guards from the
',PL]).

template(kidnap(VIL,VIC), [VIL,' kidnaps ',VIC]).

template(attack(CH,PL), [CH,' attacks the ',PL]).

template(fight(CH1,CH2), [CH1,' fights against ',CH2]).

template(kill(CH1,CH2), [CH1,' kills ',CH2]).

template(free(HERO,VIC), [HERO,' frees ',VIC]).

template(marry(CH1,CH2), [CH1,' and ',CH2,' get married']).

template(donate(CH1,CH2), [CH1,' gives strength to ',CH2]).

template(bewitch(CH1,CH2), [CH1,' bewitches ',CH2]).
```

Appendix IV

```
/* BEHAVIOURAL SCHEMA */


/* Goal-inference rules */


/* The strongest hero wants to become stronger
   than the villain */

rule(
     [
            e(i,strength(HERO,Lh)),
            e(i,villain(VIL)),
            e(i,strength(VIL,Lv)),
            h({Lh=<Lv})
     ],
     (
       [T],
      [
            h(T,strength(HERO,LS)),
            h({LS > Lv}),
            h(T>i)
     ],
     true
     )
  )
      :- findall(S,(db(strength(H,S)),db(hero(H))),Ss),
         max_list(Ss,Lh),
         db(hero(HERO)),
         db(strength(HERO,Lh)).


/* Victim spontaneously reduces the protection
   at her current location */

rule(
    [
         e(i,victim(VIC)),
         e(i,nature(VIC,KIND0)),
         e(i,current_place(VIC,PLACE)),
         e(i,protection(PLACE,[KIND1,PROT]))
     ],
     (
       [T],
        [
         h(T,current_place(VIC,PLACE1)),
         h(T,protection(PLACE1,[KIND2,PROT1])),
         h({(KIND2*KIND0*PROT1)<(KIND1*KIND0*PROT)}),
         h(T>i)
          ],
     true
     ) ).
```

```
/* If victim's protection is reduced, villain will
   want to kidnap her */

rule(
     [
         e(i,victim(VIC)),
         e(i,nature(VIC,KIND0)),
         e(i,current_place(VIC,PLACE1)),
         e(i,protection(PLACE1,[KIND1,PROT1])),
         e(i,villain(VIL)),
         h(g,alive(VIC)),
         h(g,current_place(VIC,PLACE2)),
         h(g,protection(PLACE2,[KIND2,PROT2])),
         h({(KIND2*KIND0*PROT2)<(KIND1*KIND0*PROT1)})
     ],
     (
       [T3],
      [
       h(T3,kidnapped(VIC,VIL))
      ],
      true
     )
   ).


/* If victim is kidnapped, hero will want to rescue her */

rule(
     [
        e(T1,kidnapped(VIC,VIL))
     ],
     (
       [T2],
      [
       h(T2,not(kidnapped(VIC,VIL))),
       h(T2>T1)
      ],
      true
     )
   ).


/* If victim is killed, hero will want to avenge her */

rule(
     [
         o(T1,kill(VIL,VIC)),
         h(T1,victim(VIC)),
         h(T1,villain(VIL))
     ],
     (
       [T2],
      [
       h(T2,not(alive(VIL))),
       h(T2>T1)
```

```
     ],
     true
    )
  ).


/* If the affection between two persons is high
   they will want to get married */


rule(
     [
        e(T,affection([CH1,CH2],L1)),
        h(T,affection([CH2,CH1],L2)),
        h(T,not(married(CH1,_))),
        h(T,not(married(CH2,_))),
        h({L2>95.0}), h({L1>95.0})
     ],
     (
      [T2],
      [
         h(T2,married(CH1,CH2)),
         h(T2>T)
      ],
     true
     )
  ).
```