Modeling Interactive Storytelling Genres as Application Domains

Angelo E. M. Ciarlini[1], Marco A. Casanova[2], Antonio L. Furtado[2], Paulo. A.S. Veloso[3]

[1]UNIRIO – Departamento de Informática Aplicada
[2]Pontifícia Universidade Católica do Rio de Janeiro – Departamento de Informática
[3]UFRJ – COPPE Sistemas
Rio de Janeiro - Brasil

Abstract: In this paper, we introduce a formalism to specify interactive storytelling genres in the context of digital entertainment, adopting an information systems approach. We view a genre as a set of plots, where a plot is a partially ordered sequence of events, taken from a fixed repertoire. In general, the specification of a genre should allow to determine whether a plot is a legitimate representative of the genre, and also to generate all plots belonging to the genre. The formalism divides the specification of a genre into static, dynamic and behavioral schemas, that reflect a plan recognition / plan generation paradigm. It leads to executable specifications, supported by **LOGTELL**, a prototype tool that helps users generate, modify and reuse plots that follow a genre specification. To illustrate the use of the formalism, we specify a simple Swords & Dragons genre and show plots generated by the tool.

Keywords: Interactive Storytelling; Plots; Planning; Digital Entertainment; Logic Programming.

1. Introduction

In this paper, we propose a formalism to specify interactive storytelling genres in the context of digital entertainment, which is in particular appropriated to interactive TV and computer games. The formalism adopts an information systems approach, and leads to executable specifications, supported by **LOGTELL**, a prototype tool that helps users generate, modify and reuse plots that follow a genre specification. To illustrate the use of the formalism, we specify a simple Swords & Dragons genre and show plots generated by **LOGTELL**.

The role of storytelling in computer games has long been the subject of lively debates [50]. Although some authors believe that storytelling and games are in direct opposition [14], most agree that successful narrative in games is possible, and a few argue for the importance of story creation as part of gameplay [48]. However, a different sort of story is required: it must be non-linear and play-centric, that is, it must revolve around the player's experience [37]. The player is no longer a mere consumer of the story, but both a consumer and a (co-) producer of the story. However, the game designer typically selects a genre to restrict the accepted stories [48].

The investigation of possible forms of interaction in digital TV is a very recent research area. With the dissemination of this technology, many possibilities are emerging to take advantage of the yet poorly explored capacity of direct interaction with thousands of spectators. In this new context, there are many opportunities and challenges for interactive narratives [48]. The idea of interacting with the content that is being presented is very appealing to the spectator. Differently however from conventional games, where stories are told to contextualize challenges for the player, telling stories is the main goal in an interactive TV context. The importance of the diversity and coherence of the stories therefore tends to be much greater than in conventional computer games. Hence, a formal model for the intended genre can be useful to explore its possibilities and to guarantee the coherence of the stories.

To better understand the concept of genre, we observe that studies in narratology [2] suggest that the composition of stories is a three-layered process and that each layer can be analyzed separately: fabula, story and text. Informally speaking, a fabula is a series of events happening in a real or fictional world. A story corresponds to how a fabula is reported by the author. Finally, a text is a materialization of a story in some medium, such as writing, motion picture, or animation.

We deal exclusively with the fabula layer. Accordingly, we view a genre as a set of plots, taking the word *plot* in the sense of a partially ordered sequence of events. In addition, we restrict the types of events allowed in the genre to a fixed repertoire, as proposed by the Russian literary theoretician Vladimir Propp in his seminal work on the fairy-tales genre [39]. This decision classifies our approach as primarily plot-based, in the terminology of storytelling research [25][39], as opposed to a character-based orientation [6]. However, it will become clear, in the course of the paper, that we also contemplate some character-based aspects.

We propose to specify a genre with the help of *static*, *dynamic* and *behavioral schemas*, which respectively define: (a) the mini-world wherein the plots take place; (b) what events the participants can enact; and (c) what motives guide their behavior. We combine several modeling notions, borrowed primarily from information systems [36], and also from literary theory [1][2][3][17][39][45]. Plots belonging to a genre are, to a certain extent, comparable to sentences belonging to a language, which might suggest the use of some Chomsky grammar, such as Rumelhart's story grammar [43], as a mechanism to accomplish purposes (a) and (b) above. However, we opted for a plan-recognition / plan-generation paradigm [23], and adopted a formalism based on first-order temporal logic [11] which is directly translatable to a clausal notation that the **LOGTELL** tool is able to execute. In fact, the decisions that led to the formalism adopted directly reflect the overall goal of creating executable specifications that users can experiment with to automatically or semi-automatically generate plots that belong to a genre, and to modify and reuse previously defined plots, stored in a plot library.

In [21], we argued that temporal databases are, in general, repositories of narratives about the agents and objects involved. In [23], we showed that plan-generation (and plan-recognition) algorithms can be used in the context of database narratives for decision support. An operational description of the modeling issues of this scenario can be found in [22], while a characterization of the machinery involved in the process is given in [8][10]. Our formal framework can be compared to the event calculus [29], especially to one of its variants, the event calculus with preconditions [7]. The description of narratives with situation calculus [35] enables us to better reason about the hypothetical situations that result from hypothetical actions. The difference in our approach is that we focus on the creation of coherent narratives: we reason about the situations that hold or may hold along a narrative in order to infer goals that will bring about new actions, which will in turn conduct the narrative.

The development of tools that generate plots is not new. Earlier systems include **Tale-Spin** [34], which generates plots by simulating character behavior starting with specific goals, **Universe** [30] and **Minstrel** [46], which added restrictions capturing the author’s goals. More recently, a few computational systems and approaches have been proposed to support interactive storytelling. In these new systems, plots can be created and then told with the help of sophisticated graphics, and there are different forms of user interaction. Some approaches focus on the interaction among autonomous characters [6], whereas others focus on plot structure and coherence [26]. Few others attempt to combine both. Façade [32][33], for instance, keeps characters autonomy most of the time, but their goals and their behavior can be changed by a drama manager, to move the plot forward.

Planning algorithms proved to be a useful alternative to help create narratives by exploring different chains of events to achieve the characters' or the storytellers' goals [13][42]. In [6], hierarchical task network (HTN) planning is used to control the way characters achieve their goals in accordance with user intervention. HTN planning tends to be efficient but less general, demanding the previous construction of a task hierarchy and methods to perform each task. In **Façade,** a reactive planning language is used to support the personality of believable agents. In **Mimesis** [41][52], a planner combining HTN and partial-order planning is used to create a storyline beforehand. Techniques of mediation are used at run time to guarantee the coherence, including the adoption of alternative story lines or interventions for forcing the failure of user's actions. **LOGTELL** [13][38], the system that motivated the research described in this paper, extensively uses a partial-order planning algorithm to simulate narratives in accordance with a formal logical model of a story genre. The **PlotBoard** tool, described in [9], also supports the experimentation with genre specifications to generate storyboards.

The main difference of the approach adopted in **LOGTELL** from that of other planning-based interactive storytelling systems lies in the goal of the system. Instead of working on alternatives for a basic story line, **LOGTELL** tries to enable the generation of a maximum of different and coherent stories of a certain genre by means of multiple simulation stages, combined with user intervention. A formal model for capturing the logics of the story genre is then necessary to allow users to fully explore the coherent alternatives.

To illustrate the use of the formalism, we specify an elementary Swords & Dragons genre. In the example, princesses, knights, dragons and magicians play the roles of victims, heroes and villains. They perform actions such as attack, kidnap, fight, kill and marry. The **LOGTELL** tool was able to generate (and recognize) dozens of quite different plots, all of them fully compatible with the formal specification. This simple example demonstrated that the formalism provides a solid background for interactive storytelling, since it guarantees the coherence of the stories. In addition, the generation of unexpected stories was useful to point out that we were implicitly assuming constraints that had not been formalized. In this aspect, the formal specification proved to be useful to better understand the genre. We refer the reader to [8][11][12] for a full discussion and for the complete Prolog code of our Swords & Dragons example.

The paper is organized as follows. Sections 2, 3 and 4 introduce the static, dynamic and behavioral schemas we propose. Section 5 describes **LOGTELL**, a prototype tool to experiment with genres, and discusses some of the key features of the underlying plot generation algorithm. Section 6 contains the conclusions. The appendix exemplifies the formalism and the use of **LOGTELL** for the Swords & Dragons genre

2. Static schema

2.1 Definition

To model a genre, we start by providing a description of the mini-world wherein the plots take place, which we call the *static schema* of the genre. The formal specification of static schemas follows Ciarlini and Furtado [11] and is based on the standard syntax and semantics of first-order languages. However, we also introduce special concepts that help understand the tool described in Section 5.

Informally, a static schema defines entity classes, (binary) relationship classes and attributes [15]. An entity class may have an identifying attribute. Attributes have types, including *Boolean*, and may be composite. A static schema also assigns roles, in the theatrical sense and in the sense of the agent concept, to certain entities.

A fact is an assertion about the existence of entities, the values of the attributes of entities, the existence of relationships, the values of the attributes of the relationships, or the assignment of roles to entities. The set of facts holding at a given instant of time constitutes a state.

Formally, a *static language* is a first-order language, with equality, whose alphabet contains a set of *database symbols* partitioned into:

entity class names: unary predicate symbols to denote entity classes

Boolean entity attribute names: unary predicate symbols to denote Boolean entity class attributes

simple entity attribute names: binary predicate symbols to denote (non-composite) entity class attributes

composite entity attribute names: n-ary predicate symbols to denote composite entity class attributes

relationship names: binary predicate symbols to denote relationship classes

Boolean relationship attribute names: binary predicate symbols to denote Boolean relationship class attributes

simple relationship attribute names: binary predicate symbols to denote (non-composite) relationship class attributes

composite relationship attribute names: n-ary predicate symbols to denote composite relationship class attributes

role names: unary predicate symbols to denote entity roles

database constants: constants to denote data values

Besides the database symbols, the alphabet of a static language contains *constraint predicate symbols*, corresponding to constraint predicates over concrete domains (such as equalities and inequalities over the Real numbers), and *constraint function symbols*, corresponding to functions over concrete domains (such as addition and subtraction over the Real numbers).

For example, the static language of our Swords & Dragons genre has the following database symbols:

entity class names: `creature, person, knight, princess, magician, dragon, place`

Boolean entity attribute names: `alive`

simple entity attribute names: `name, nature, strength, place_name`

composite entity attribute name: `protection` (a ternary predicate symbol, indicating the kind and level of protection of a place)

relationship names: `home, current_place, acquaintance, married, kidnapped`

simple relationship attribute names: `affection`

role name: `hero, victim, villain, donor`

database constants: `'Marian', 'White_Palace', 'Hoel'`

In what follows, let $\mathcal{L}$ be a static language with a set of database symbols $\mathcal{D}$. The syntax and semantics of $\mathcal{L}$ follow as for first-order languages, so that we will concentrate just on the details that matter to our discussion.

A *substitution* is a function *i* that maps each variable of $\mathcal{L}$ into a term. A substitution is *ground* iff it maps each variable of $\mathcal{L}$ into a variable-free term.

An *expression e* is a formula or a term of $\mathcal{L}$. Given a substitution *i*, we use *e[i]* to denote the expression obtained by applying *i* to *e*. A *(ground) instance* of *e* is an expression obtained by applying a (ground) substitution to *e*.

A *literal* is an expression of the form $p(t_1,...,t_n)$ or of the form $\neg p(t_1,...,t_n)$, where *p* is an n-ary predicate symbol of $\mathcal{L}$ and $t_1,...,t_n$ is a list of terms of $\mathcal{L}$. A *database literal* is a literal whose predicate symbol is in $\mathcal{D}$, and a *database fact* is a ground positive database literal. Analogously, a *constraint literal* is a literal whose predicate symbol is a constraint predicate symbol.

Sample database facts are:

`princess('Marian')` (`'Marian'` is an instance of entity class `princess`)

`strength('Marian',10)` (the value of attribute `strength` for `'Marian'` is `10`)

`alive('Marian')` (the value of attribute `alive` for `'Marian'` is `True`)

`acquaintance('Marian','Hoel')`
(`'Marian'` and `'Hoel'` are instances related by the relationship `acquaintance`)

`affection('Marian','Hoel',0)`
(the value of attribute `affection` for `'Marian'` and `'Hoel'` is `0`)

`place('White_Palace')` (`'White_Palace'` is instance of entity class `place`)

`protection('White_Palace',1,70)`
(the value of the composite attribute `protection` for `'White_Palace'` is `1` and `70`, respectively indicating the kind and level of protection)

`victim('Marian')` (`'Marian'` plays the role of `victim`)

A *safe conjunction* is a conjunction of constraint literals, positive database literals or possibly universally quantified negative database literals such that any variable occurring in the conjunction is either governed by a local universal quantifier or occurs in a positive database literal. For example, the conjunction `place(P)∧∀K∀L(¬protection(P,K,L))` is safe, but not the conjunction `place(P)∧¬protection(P,K,L)`. The possibility of quantifying variables in negative database literals is useful when defining operations and goal-inference rules (see Sections 3).In the Prolog implementation of our tool, their evaluation is based on the closed world assumption [40]. The quantification of positive literals is more complicated to implement and is not allowed.

In what follows, we use the notation $\forall\bar{x}(\neg L[\bar{x},\bar{y}])$ to indicate that $\neg L$ is a negative literal with variables partitioned into two lists, $\bar{x}$ and $\bar{y}$, such that the variables in $\bar{x}$ are universally quantified. Likewise, $\forall\bar{x}(\neg L[\bar{x},\bar{b}])$ denotes the result of substituting the variables in $\bar{y}$ by the constants in the list $\bar{b}$.

A *structure* M for $\mathcal{L}$ assigns to each symbol *s* of $\mathcal{L}$ an interpretation s^M as usual. We assume that M assigns the usual interpretation to constraint predicate symbols and constraint function symbols (such as the interpretation assigned by arithmetic to numeric constraints). Given a formula σ and a set of formulas Σ of $\mathcal{L}$, we use $\models_M \sigma$ to denote that M *satisfies* σ, or that σ *holds* in M, and $\Sigma \models \sigma$ to denote that σ is a *logical consequence* of Σ.

A *possible fact* of M is a database fact of $\mathcal{L}$ that holds in M. A *possible database state* of M is a set of possible facts of M.

We define the notion of a safe conjunction *holding* at a possible database state $\boldsymbol{s}$ as follows:

- a ground positive database literal L *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \vDash_M L$, iff $L \in \boldsymbol{s}$
- a ground negative database literal $\neg L$ *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \vDash_M \neg L$, iff $L \notin \boldsymbol{s}$
- a ground constraint literal C holds in s for M, denoted $\boldsymbol{s} \vDash_M C$, iff C is true in the usual interpretation
- a formula of the form $\forall \bar{x}(\neg L[\bar{x},\bar{b}])$ *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \vDash_M \forall \bar{x}(\neg L[\bar{x},\bar{b}])$, iff, for all ground substitutions i, $L[\bar{a},\bar{b}] \notin \boldsymbol{s}$, where $\bar{a}$ are the constants that i assigns to $\bar{x}$
- a safe conjunction of the form $L_1 \wedge \ldots \wedge L_n$ *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \vDash_M L_1 \wedge \ldots \wedge L_n$, iff, for all ground substitutions i, for each $k \in [1,n]$, $\boldsymbol{s} \vDash_M L_k[i]$

The previous definitions outline the basic syntax and semantics of static languages, but they do not capture the semantics of the genre mini-world. We therefore define a *static schema* as a pair $\mathcal{S}=(\mathcal{L},\mathcal{A})$ such that $\mathcal{L}$ is a static language and $\mathcal{A}$ is a set of formulas of $\mathcal{L}$, called the *axioms* of $\mathcal{S}$, that includes the following:

- *model constraints:*
  - for each identifier defined for an entity class, $\mathcal{A}$ includes exactly an *identifier axiom* that captures the uniqueness property of the identifier
  - for each attribute defined for an entity or relationship class (including the Boolean and composite attributes), $\mathcal{A}$ includes exactly an *attribute axiom* that captures the domain and range of the attribute
  - for each relationship, $\mathcal{A}$ includes exactly a *relationship axiom* that captures which entity classes the relationship involves
  - *is-a axioms* indicating which classes are specializations of other classes
  - *role axioms* that constrain roles to denote sets of entity instances in the union of collections of entity sets (that play the role in the static schema)
- *application domain constraints* that capture other properties of the application domain

The formal definition of axioms similar to the model constraints, framed in Description Logic, can be found in [4][5]. Furthermore, we note that the Prolog tool discussed throughout the paper was implemented to automatically take into account the model constraints.

Finally, a *model* of $\mathcal{S}=(\mathcal{L},\mathcal{A})$ is defined as a structure for $\mathcal{L}$ that satisfies all axioms in $\mathcal{A}$.

2.2 Concrete notation

The formal specification of a genre is directly translatable to a concrete notation that **LOGTELL**, the prototype tool described in Section 5, is able to execute. In other words, the specification of a genre, in what we call *concrete notation*, is executable and can be used for experimentation, in the sense that the user can invoke **LOGTELL** to automatically or semi-automatically generate plots that belong to a genre, and to modify and reuse previously defined plots, stored in a plot library.

The concrete notation for a static schema is based on Prolog clauses of the following forms:

```
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-name>,(<entity-class>;...;<entity-class>)).
```

Note that we use square brackets for conjunctive lists (with "," as separator) and regular parentheses for disjunctions (with ";" as separator).

A set of Prolog clauses of the above forms has the double purpose of defining the alphabet of the static language and indicating which model constraints apply.

The concrete notation of a database fact `L` is a Prolog clause of the form `db(L')`, where `db` is a special unary predicate symbol, and `L'` is fact `L` rewritten as a term with the help of above Prolog concrete notation for lists. In more detail, we have:

Schema clause:	`entity(<entity class>,<identifying attribute name>).`
Database fact:	`db(<entity class>(<identifier>)).`
Schema clause:	`attribute(<entity class>,<attribute name>).`
Database fact:	`db(<attribute name>(<identifier>,<attribute value>)).`
Schema clause:	`relationship(<relationship name>, [<entity class>,<entity class>]).`
Database fact:	`db(<relationship name>([<identifier>,<identifier>])).`
Schema clause:	`attribute(<relationship name>,<attribute name>).`
Database fact:	`db(<attribute name>([<identifier>,<identifier>], <attribute value>)).`
Schema clause:	`role(<role name>,<entity class>).`
Database fact:	`db(<role name>(<identifier>)).`

The specification of the static schema of the Swords & Dragons genre, in concrete notation, is:

```
/* Static Schema */

1.   entity(creature,name).
2.   entity(person,name).
3.   entity(knight,name).
4.   entity(princess,name).
5.   entity(magician,name).
6.   entity(dragon,name).
7.   entity(place,place_name).
8.   is_a(person,creature).
9.   is_a(knight,person).
10.  is_a(princess,person).
11.  is_a(magician,person).
12.  is_a(dragon,creature).
13.  attribute(creature,nature).
14.  attribute(creature,strength).
15.  attribute(creature,alive).
16.  attribute(place,protection).
17.  boolean(alive).
18.  composite(protection,[kind,level]).
19.  relationship(home,[creature,place]).
```

```
20.  relationship(current_place,[creature,place]).
21.  relationship(acquaintance,[creature,creature]).
22.  relationship(married,[person,person]).
23.  relationship(kidnapped,[person,creature]).
24.  attribute(acquaintance,affection).
25.  role(hero,knight).
26.  role(victim,(princess;knight)).
27.  role(villain,(dragon;knight)).
28.  role(donor,magician).

/* Sample Database Facts */

29.  db(princess('Marian')).
30.  db(strength('Marian',10)).
31.  db(alive('Marian')).
32.  db(protection('White_Palace',[1,70])).
33.  db(acquaintance(['Marian','Hoel'])).
34.  db(affection(['Marian','Hoel'],0)).
35.  db(victim('Marian')).
```

Note that, according to the clause in line 14, `strength` is an attribute of `creature` and, in view of the clauses in lines 8 and 10, also an attribute of `person` and of `princess`. Hence, the database fact in line 30 is acceptable. Also note the special notation for the database facts in lines 31 and 32, in view of the declaration in line 17 of `alive` as a Boolean attribute and the declaration in line 18 of `protection` as a composite attribute. We stress that the Prolog tool interprets the clauses in lines 1 to 28 to automatically generate the model constraints of the static schema.

3. Dynamic schema

3.1 Definition

After defining the static schema, the specification of a genre continues with the definition of the *dynamic schema*, covering operation and case declarations.

We restrict state changes to what can be accomplished by applying a limited repertoire of operations, defined in terms of their parameters and pre- and post-conditions [18]. Pre-conditions are conjunctions of positive or negative database 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 retracted as a consequence of executing the operation. Guaranteeing the transition between valid states therefore depends on a careful adjustment of the interplay among pre- and post-conditions over the entire repertoire of operations.

The specification of an operation is extended to associate pragmatic information, especially concerning *agency*, with the parameters of the operation. Out of the various choices of cases [19], we employ here*:* agent, coagent, recipient, patient, object, and destination*.* In the parameter list of an operation, each parameter is associated with a case, paired with either the entity class or the role involved. Indicating a role, instead of an 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.

Formally, a *dynamic language* $\mathcal{L}$ is a static language whose alphabet is extended to include a new set of symbols, the *operation names*, with an associated *arity*, and the *case names*, and whose set of expressions is extended to include operation and case declarations.

An *operation declaration* for an n-ary operation name o is an expression σ of the form $\{P\}o(x_1,...,x_n)\{Q\}$, where

- $o(x_1,...,x_n)$ is the *input declaration* of σ, where $x_1,...,x_n$ is a list of variables of $\mathcal{L}$
- P is the *pre-condition* of σ, defined as a safe conjunction
- Q is the *post-condition* of σ, defined as a conjunction of positive or negative database literals

An operation is associated with an operation frame, which further constrains the semantics of the operation. A *case declaration* is an expression of the form $c:(e_1,...,e_m)$ where c is a case name and $(e_1,...,e_m)$ is a list of entity or role names. Given an n-ary operation name o, an *operation frame for o* is an expression of the form $o(c_1,...,c_n)$ where $c_1,...,c_n$ is a list of case declarations.

A structure M for $\mathcal{L}$ is defined as for static languages, except that M assigns to each operation name o a set o^M of pairs of possible database states of M, specifying the possible state transitions related to o.

Given a substitution i, we use $\sigma[i]$ to denote the *substituted version* of an operation declaration σ, obtained by uniformly applying i to the input declaration and to the pre- and post-conditions of σ. We say that σ is *ground* iff the input declaration, the pre- and the post-conditions of σ have no free variables.

Let σ be an operation declaration for an n-ary operation name o and assume that σ is of the form $\{P\}o(x_1,...,x_n)\{Q\}$. Let i be a ground substitution of $\mathcal{L}$. Then, a pair $(\boldsymbol{s},\boldsymbol{t})$ of possible database states in o^M *satisfies* $\sigma[i]$ *for* M iff

- $\boldsymbol{s} \vDash_M P[i]$ (the pre-conditions are satisfied in $\boldsymbol{s}$ for M)
- $\boldsymbol{t} \vDash_M Q[i]$ (the post-conditions are satisfied in $\boldsymbol{t}$ for M)
- for every possible database fact f of M, if neither f nor $\neg f$ occur in $Q[i]$, then $\boldsymbol{t} \vDash f$ iff $\boldsymbol{s} \vDash f$ (which is the frame requirement: preservation of satisfaction from $\boldsymbol{s}$ to $\boldsymbol{t}$ for ground database literals that are neither established nor negated by the post-condition $Q\theta$, which is ground by assumption)

Furthermore, M *satisfies* σ iff, for every pair $(\boldsymbol{s},\boldsymbol{t})$ of possible database states in o^M, there is a ground substitution i of $\mathcal{L}$ such that $(\boldsymbol{s},\boldsymbol{t})$ satisfies $\sigma[i]$ for M. Note that, substitutions do not affect universally quantified variables that may appear in negative pre-conditions.

If an operation name o is associated with an operation frame of the form $o(c_1:(e_{11},...,e_{1m1}),...,c_n:(e_{n1},...,e_{nmn}))$ and an operation declaration σ of the form $\{P\}o(x_1,...,x_n)\{Q\}$ then we redefine the notion that M *satisfies* σ, or that σ *holds* in M, iff M satisfies $\sigma[i]$, for every ground substitution i of $\mathcal{L}$ such that $x_k[i]^M \in e_{k1}^M \cup ... \cup e_{kmk}^M$.

An example of an operation frame and an operation declaration from our running example would be the expressions f and $\{P\}o(x_1,x_2)\{Q\}$, where:

```
f = reduce_protection(agent:(victim),destination:(place))

o(x1,x2) = reduce_protection(V,P)  /* V is a victim and P is a place        */

P = {current_place(V,P)∧      /* V is currently at P                      */
     protection(P,K,L)∧       /* P has protection of kind K and level L  */
     nature(V,K)∧             /* the nature of V is K                     */
     L>0.0}                   /* the level of protection is positive     */
Q = {¬protection(P,K,L))∧    /* P no longer has protection K and L      */
     protection(P,K,L-10.0))}/* P now has protection K and L-10.0       */
```

Finally, we define a *dynamic schema* as a pair $\mathcal{D}=(\mathcal{L},\mathcal{A})$ such that $\mathcal{L}$ is a dynamic language and $\mathcal{A}$ is a set of formulas of $\mathcal{L}$, called the *axioms* of $\mathcal{D}$, that includes model constraints, application domain constraints, operation declarations and operation frames such that, for each operation name *o* of $\mathcal{L}$, there is exactly one operation declaration for *o* in $\mathcal{A}$ and at most one operation frame for *o* in $\mathcal{A}$.

A *model* of $\mathcal{D}=(\mathcal{L},\mathcal{A})$ is defined as a structure for $\mathcal{L}$ that satisfies all axioms in $\mathcal{A}$.

3.2 Concrete notation

The concrete notation for a dynamic schema is based on Prolog clauses of the following forms, which specify operation frames and operation declarations:

```
   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>,
         <post-conditions>,
         <estimated cost of operation>,
         <main effects>,
         [],[]).
```

The notion of main effects is new, conveys pragmatic information to the plot generation algorithm (see Section 5.2), and does not affect the semantics of the operation. The pre-conditions, post-conditions and main effects are conjunctive lists of literals, denoted using square brackets and "`,`" as separator. The purpose of the last two parameters, shown above as empty lists, denoted as "`[]`", will be explained at the end of Section 4.2.

The sample operation declaration of Section 3.1, in Prolog concrete notation, becomes:

```
operator_frame(2,reduce_protection,[agent:victim,object:place]).

operator(2,
      reduce_protection(V,P),/* V is a victim and P is a place          */
      [
        current_place(V,P),   /* V is currently at P                    */
        protection(P,[K,L]),  /* P has protection of kind K and level L  */
        nature(V,K),          /* the nature of V is K                   */
        L>0.0                 /* the level of protection is positive     */
      ],
      [
        not(protection(P,[K,L])), /* P no longer has protection K and L */
        protection(P,[K,L-10.0])  /* P now has protection K and L-10.0  */
      ],
        10,                       /* (estimated cost of operation)      */
      [protection(P,[K,L-10.0])], /*(main effect of the operation)      */
      [],[]).
```

We refer the reader to [12] for the complete Prolog code of our running example.

4. Behavioral schema

4.1 Definition

The final stage of the specification of a genre consists of defining the *behavioral schema*. As the name implies, it defines the behavior of animated agents, and consists of a set of goal-inference rules that capture the goals that motivate the agents' actions when certain situations occur during a narrative. The behavioral schema may also contain a library of predefined plots, also called complex operations.

The goal-inference rules convey useful information to plot generation. Whenever goals are inferred but not achieved within a plot, a plot composition procedure considers the possible extensions to the plot so that these goals are fulfilled. In the LOGTELL tool, as discussed in Section 5, plots are incrementally generated by means of multiple simulation stages, with phases of goal-inference and planning. At a given initial state, the algorithm first applies goal-inference rules to determine goals for the various agents. Then, it creates a partial plot to achieve the goals of each agent, taking into account their possible mutual interferences. The planned events result in new states wherein, again, the plot generation algorithm applies the goal-inference rules, until no new goal is brought about by the states in the plot. **LOGTELL** allows users to interfere with the process of plot generation either by choosing alternatives generated by the planner or by artificially trying to force situations or the occurrence of events. Whatever is the user interference, it is incorporated only if the simulation can find means to keep the plot coherent and within the bounds of the specification of the genre.

If a library of predefined plots is available, a plot generation algorithm (or the user, directly) may select a predefined plot from the library, adapting it if necessary, to achieve the goals of an agent. Using predefined plots as building blocks corresponds, in Artificial Intelligence terminology, to start from commonly used *scripts*, rather than constructing new plots from scratch with the primitive operations [44]. In literary terminology, there is a notion analogous to scripts, namely *types* and *motifs*, as in the monumental index compiled by Aarne and Thompson [1].

Plots are modeled as partially ordered sets of events. Partial rather than total ordering is a consequence of the use of partial-order planning during the simulation, establishing temporal constraints only when necessary, which makes the conciliation of goals easier. As a consequence, the truth of a fact at a certain time might depend on the final total order that will be chosen later.

Formally, a *behavioral language* $\mathcal{L}$ is a dynamic language whose set of expressions is extended to include *temporal formulas* recursively defined as formulas for first-order languages, with one additional syntactic rule:

- if α and β are temporal formulas, then so are the expressions:

$\Diamond\alpha$	α eventually holds
$\Box\alpha$	α always holds
$\circ\alpha$	α holds next
$\alpha\,\mathcal{U}\,\beta$	α holds until β

The *goal-inference rules* are a special set of temporal formulas, recursively defined as follows:

- a temporal formula of the form $\Diamond\boldsymbol{L}$ is a goal-inference rule, where $\boldsymbol{L}$ is a safe conjunction

- a temporal formula of the form $(\boldsymbol{L} \Rightarrow \Gamma)$ or of the form $\Box(\boldsymbol{L} \Rightarrow \Gamma)$ is a goal-inference rule, where $\boldsymbol{L}$ is a safe conjunction and Γ is a goal-inference rule

The quantification of variables in goal-inference rules obeys the following rules:

- the variables occurring only within the scope of a temporal operator $\Diamond$ are locally existentially quantified; and
- all other variables are globally universally quantified.

Using these rules, quantifiers can be left implicit in the goal-inference rules. Examples of goal-inference rules are:

```
/* A hero wants to become stronger than the villain is
   at the initial state */

      (villain(VIL)∧ strength(VIL,Lv) ∧
       hero(HERO) ∧ strength(HERO,Lh)
       ⇒ ◊(strength(HERO,LS) ∧ LS > Lv)

/* If a villain kidnaps a victim, he or she will be eventually rescued */

     □(kidnapped(VIC,VIL)⇒ ◊¬kidnapped(VIC,VIL))
```

The rest of this section discusses the semantics of behavioral languages, which is far more complex, and formally introduces the concepts of events, logs and plots.

Let $\mathcal{L}$ be a behavioral language and M be a structure for $\mathcal{L}$. A *temporal database* of M is a sequence $\boldsymbol{S}=(\boldsymbol{s_0},\boldsymbol{s_1},...)$ of possible database states of M. Given $k\in[0,|\boldsymbol{S}|]$, the k^{th} *continuation* of $\boldsymbol{S}$ is the temporal database $\boldsymbol{S}^k=(\boldsymbol{s_k},\boldsymbol{s_{k+1}},...)$, that is, the suffix of $\boldsymbol{S}$ starting on the k^{th} element of $\boldsymbol{S}$. Note that a continuation $\boldsymbol{S}^k$ of $\boldsymbol{S}$ is indeed a temporal database, and that the temporal database consisting of just the last state of $\boldsymbol{B}$ is the n^{th} continuation of $\boldsymbol{S}$, where $n=|\boldsymbol{S}|$.

We define the notion of *holding* at a temporal database $\boldsymbol{S}$ of M, as follows:

- a first-order formula α *holds* in $\boldsymbol{S}$ for M, denoted $\boldsymbol{S} \vDash_M \alpha$, iff $\boldsymbol{s_0} \vDash_M \alpha$ (that is, α holds in the first state $\boldsymbol{s_0}$ of $\boldsymbol{S}$)
- a formula of the form $\Diamond\alpha$ *holds* in $\boldsymbol{S}$ for M, denoted $\boldsymbol{S} \vDash_M \Diamond\alpha$, iff for some $k\in[0,|\boldsymbol{S}|]$, $\boldsymbol{S}^k \vDash_M \alpha$ (that is, α holds in some continuation $\boldsymbol{S}^k$ of T)
- a formula of the form $\Box\alpha$ *holds* in $\boldsymbol{S}$ for M, denoted $\boldsymbol{S} \vDash_M \Box\alpha$, iff for all $k\in[0,|\boldsymbol{S}|]$, $\boldsymbol{S}^k \vDash_M \alpha$ (that is, α holds in all continuations $\boldsymbol{S}^k$ of $\boldsymbol{S}$)
- a formula of the form $\bigcirc\alpha$ *holds* in $\boldsymbol{S}$ for M, denoted $\boldsymbol{S} \vDash_M \bigcirc\alpha$, iff $\boldsymbol{S}^1 \vDash_M \alpha$ (that is, α holds in $\boldsymbol{S}^1$, the continuation starting on the second state of $\boldsymbol{S}$)
- a formula of the form $\alpha\mathcal{U}\beta$ *holds* in $\boldsymbol{S}$ for M, denoted $\boldsymbol{S} \vDash_M (\alpha\mathcal{U}\beta)$, iff there is $p\in[0,|\boldsymbol{S}|]$ such that $\boldsymbol{S}^p \vDash_M \beta$ and, for all $q\in[0,p)$, $\boldsymbol{S}^q \vDash_M \alpha$ (that is, α holds in all continuations until reaching, but excluding, a continuation where β holds)

We extend the notion of holding at a temporal database to complex temporal formulas as usual. Using this notion, we can understand the meaning of a goal-inference rule like $\Box(\boldsymbol{C_1} \Rightarrow \Box(\boldsymbol{C_2} \Rightarrow \Box(\boldsymbol{C_3} \Rightarrow \Diamond\boldsymbol{G})))$. Due to the implications used, the formula holds at a temporal database $\boldsymbol{S}$ of M iff, whenever there are three states $\boldsymbol{S}_{c1}$, $\boldsymbol{S}_{c2}$, $\boldsymbol{S}_{c3}$, with $c_1 \leq c_2 \leq c_3$, such that $\boldsymbol{C_i}$ holds in $\boldsymbol{S}_{ci}$, for i=1,2,3, then there must be a state $\boldsymbol{S}_g$ with $c_3 \leq g$, such that $\boldsymbol{G}$ holds in $\boldsymbol{S}_g$. Intuitively, $\boldsymbol{C_1}$, $\boldsymbol{C_2}$, $\boldsymbol{C_3}$ define a sequence of conditions that, if satisfied, implies that the goal $\boldsymbol{G}$ must be satisfied in a future state (with respect to the state where $\boldsymbol{C_3}$ is satisfied).

A *behavioral schema* is a pair $\mathcal{B}=(\mathcal{L},\mathcal{A})$ such that $\mathcal{L}$ is a behavioral language and $\mathcal{A}$ is a set of formulas of $\mathcal{L}$, called the *axioms* of $\mathcal{B}$, that includes constraints, domain constraints, operation declarations and goal-inference rules, such that $\mathcal{A}$ satisfies the same restrictions as for dynamic schemas. A *model* of $\mathcal{B}$ is a structure for $\mathcal{L}$ that satisfies all axioms in $\mathcal{A}$.

Let $\mathcal{B}$ be a behavioral schema with language $\mathcal{L}$. Let M be a model of $\mathcal{B}$. Recall that, to each operation name o of $\mathcal{L}$, the model M assigns a set o^M of pairs of database states. Given a substitution i of $\mathcal{L}$ and an operation declaration σ, also recall that $\sigma[i]$ denotes the *substituted version* of σ, obtained by uniformly applying i to the input declaration, the pre- and the post-conditions of σ. In particular, i may be a ground substitution of $\mathcal{L}$.

An *event* is an expression e of the form $o(t_1,...,t_n)$, where o is an n-ary operation name of $\mathcal{L}$ and $t_1,...,t_n$ is a list of terms of $\mathcal{L}$. The *parameter substitution* of e is the partial substitution i_e that maps x_i into t_i, for $i\in[1,n]$, and is undefined for the other variables of $\mathcal{L}$. We define an interpretation in M for e as the set e^M consisting of all pairs $(\boldsymbol{s},\boldsymbol{t})$ of possible database states in o^M such that $(\boldsymbol{s},\boldsymbol{t})$ satisfies $\sigma[j]$ for M, where $j = i_e\circ g$, for some ground substitution g.

Let e be an event of the form $o(t_1,...,t_n)$, σ be the operation declaration for o, and i_e be the parameter substitution of e. Then, $\sigma[i_e]$ is the *specification for e induced by* σ. A structure M *satisfies* $\sigma[i_e]$ *with respect to* σ iff M satisfies every ground instance $\sigma[j]$ of σ, where $j = i_e\circ g$, for some ground substitution g.

Let e be an event of $\mathcal{L}$ and assume that e is of the form $o(t_1,...,t_n)$. One can prove that, if M is a model of $\mathcal{B}$, then M satisfies $\sigma[i_e]$ with respect to the (unique) operation declaration for o that occurs in $\mathcal{A}$.

We prefer to introduce logs and plots as meta-level concepts, rather than expressions of behavioral languages, to avoid complex syntactical structures. Let $\mathcal{L}$ be a behavioral language and M be a structure of $\mathcal{L}$ in what follows.

A *log* is a possibly empty finite sequence $\boldsymbol{E}=(\boldsymbol{e_1},\boldsymbol{e_2},...,\boldsymbol{e_n})$ of ground events of $\mathcal{L}$. A temporal database $\boldsymbol{S}=(\boldsymbol{s_0},\boldsymbol{s_1},...)$ of M *satisfies* $\boldsymbol{E}=(\boldsymbol{e_1},\boldsymbol{e_2},...,\boldsymbol{e_n})$, denoted $\boldsymbol{S}\models_M \boldsymbol{E}$, iff $|\boldsymbol{S}| > n$ and, for each $i\in[1,n]$, $(\boldsymbol{s_{i-1}},\boldsymbol{s_i})\in\boldsymbol{e_i}^M$ (i.e, $\boldsymbol{e_i}$ caused the transition from $\boldsymbol{s_{i-1}}$to $\boldsymbol{s_i}$).

A *plot* is a pair $P=(P_E,P_D)$, where P_E is a finite set of events and $P_D \subseteq P_E\times P_E$ is a partial order over P_E. A log $\boldsymbol{E}=(\boldsymbol{e_1},\boldsymbol{e_2},...,\boldsymbol{e_n})$ is *consistent with* $P=(P_E,P_D)$ iff there is a ground substitution g such that, for each event e in P_E, the ground instance $e[g]$ of e occurs in $\boldsymbol{E}$ and, for every $e_1,e_2\in P_E$, if $(e_1,e_2)\in P_D$, then $e_1[g]$ precedes $e_2[g]$ in $\boldsymbol{E}$.

A temporal database $\boldsymbol{S}=(\boldsymbol{s_0},\boldsymbol{s_1},...)$ of M *satisfies* $P=(P_E,P_D)$, denoted $\boldsymbol{S}\models_M P$, iff there is a log $\boldsymbol{E}=(\boldsymbol{e_1},\boldsymbol{e_2},...,\boldsymbol{e_n})$ such that $\boldsymbol{S}$ satisfies $\boldsymbol{E}$ and $\boldsymbol{E}$ is consistent with P. Finally, a plot P is *consistent* with a set Π of goal-inference rules with respect to a structure M iff, for every temporal database $\boldsymbol{S}$ of M, if $\boldsymbol{S}$ satisfies P then $\boldsymbol{S}$ also satisfies Π.

We note that Ciarlini and Furtado [11] describe a computational method for checking the satisfaction of goal-inference rules with respect to a plot without constructing the corresponding temporal databases. We also refer the reader to [8][10] for the formalization of complex operations, which is outside the scope of this paper.

4.2 Concrete notation

The concrete notation for a behavioral schema is based on Prolog clauses specifying goal-inference rules and complex operations.

The concrete notation for a goal-inference rule is a Prolog clause of the form

```
rule(<situation>,<goal>)
```

where the situation and the goal are conjunctive lists of literals, denoted using square brackets and "`,`" as separator. The concrete notation does not use modalities, but adopts timestamp variables and the following meta-predicates to relate the truth value of literals, or the occurrence of events, to timestamp variables:

`h(T,L)`	- the literal `L` is necessarily true at time `T`
`p(T,L)`	- the literal `L` is possibly true at time `T`
`e(T,L)`	- the literal `L` is established at time `T`
`o(T,E)`	- the event `E` occurred at time `T`

As an example, consider again the following goal-inference rule, listed in Section 4.1:

$\Box$(kidnapped(VIC,VIL) $\Rightarrow$ $\Diamond\neg$kidnapped(VIC,VIL))

In Prolog concrete notation, the rule is written as (once again, we refer the reader to [12] for the complete Prolog code of our running example):

```
/* If a villain kidnaps a victim, he or she will be eventually rescued */
rule( [e(T1,kidnapped(VIC,VIL))],
      ([T2],[ h(T2,not(kidnapped(VIC,VIL))),h(T2>T1)],true) ).
```

The concrete notation for complex operations is the same as that for basic operations, introduced in Section 3.2, except that the two last parameters (shown as empty lists in the `operator` clause pattern of section 3.2) contain, respectively, the list of component operations, each with a different *tag*, and a list of tag pairs defining a partial order between the component operations.

5. LOGTELL - A Tool to Experiment with Genre Specifications

We developed a tool, **LOGTELL**, to interactively generate and dramatize stories, alternating stages of goal inference, plot generation, user intervention and 3D visualization. In this section, we outline the architecture of **LOGTELL,** describe how the user interacts with the tool, and summarize the main features of the interactive plot generator module. Section A.5 of the appendix illustrates the use of **LOGTELL** in the context of our Swords & Dragons running example.

5.1 The Architecture of LOGTELL

Figure 1 shows the architecture of **LOGTELL**. The **Plot Manager** exposes an interface that the user accesses to interactively generate plots that conform to a genre specification.

The **Plot Manager** in turn invokes the **Interactive Plot Generator** (**IPG**) to generate plots and to enforce coherence. **IPG** accepts user goals to guide plot generation, based on the genre specification. It also allows the user to interfere with the plot generation process.

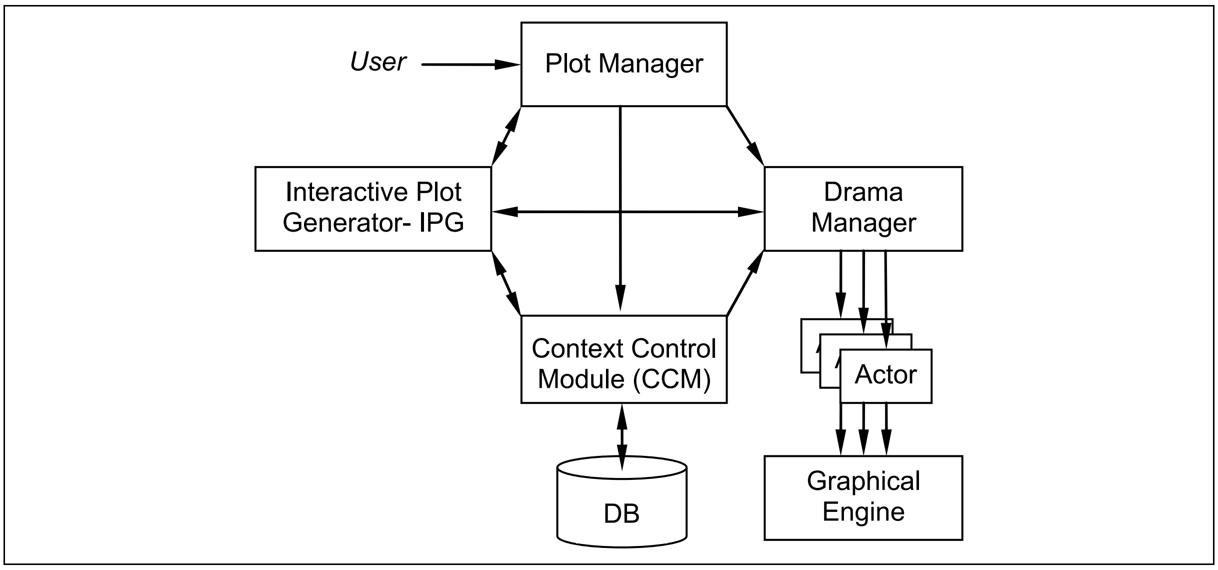


Figure 1: The architecture of **LOGTELL**.

The **Drama Manager** is responsible for the dramatization of the plot. During the dramatization, the **Drama Manager** invokes **IPG** to keep the coherence between logical and graphical representations.

The **Drama Manager** controls actors for each character in a 3D environment running on the **Graphical Engine**. The **Graphical Engine** does not perform any intelligent processing, but merely renders, at each frame, the scene and the current actors' aspect and movements, resulting from real-time interactions with the scene and, occasionally, with other actors. Section A.5 of the appendix contains a scene of our Swords & Dragons running example synthesized by the **Graphical Engine**. We refer the reader to [13] for the full details of the dramatization of a plot.

Finally, the **Context Control Module** stores all data in a single database and provides data access to the other modules.

5.2 User Interaction

To start the plot generation process, the user selects a context for the stories, composed of a genre and an initial state that must conform with the static schema of the genre. The user may either define goals that the characters initially aim to achieve or allow **IPG** to automatically generate the initial goals.

IPG then starts the plot generation process, simulating the execution of the plot from the initial state. At each simulation step, **IPG** inserts new events into the current plot to achieve the current goals, and sends them to the **Plot Manager** for the user's consideration.

User interventions may be either *weak* or *strong*. In a weak intervention, the user just selects (or rejects) partially-generated plots that seem interesting (or not) from his/her perspective to continue the simulation. This weak form of intervention usually leads the plot to situations that the author of the context has devised beforehand. The strong form of user intervention leads to the generation of more personalized plots.

Plot Manager supports weak user intervention through two commands: **continue** and **another**. The command **continue** asks **IPG** to try to infer new goals, using the goal-inference rules, and continue the simulation process; the command **another** requests **IPG** to generate an alternative solution to achieve the same goals of the step just finished.

Plot Manager supports strong user intervention by means of two commands: **insert situation** and **insert event**. 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**. For performance reasons, **IPG** may fail to obtain a valid computable plot if the search limits are exceeded. As for weak intervention, the user may confirm the solution, by issuing the **continue** command, or request an alternative, through the **another** command.

The **insert event** command permits the user to explicitly insert events into the plot. To validate the insertions, the user must invoke **IPG** through the **continue** command. **IPG** then runs the plot generation algorithm to check whether the user defined events just inserted are consistent with the ongoing plot. If not, **IPG** tries to fulfill possible unsatisfied constraints by inserting further events. The user may also remove user defined events that were not yet incorporated to (or were rejected by) the plot generation algorithm.

Finally, the user must manually convert the plot, which is a partially-ordered set of events, into a strict sequence of events, respecting the temporal constraints supplied by **IPG**. 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. Once this stage of the process is finished, the user may issue the **render** command to invoke the **Drama Manager** to dramatize the linear sequence of events.

The interaction method described here corresponds to the *step-by-step* interaction mode of LOGTELL. In order to use LOGTELL in the context of interactive TV, a new version has been developed [14], which also supports a *continuous* mode, in which a plot is generated in parallel with dramatization and user interaction. In this case, each simulation phase is a chapter, whose total order of events is automatically generated. In this mode, weak and strong interventions can also occur. Strong interventions are incorporated, if coherent, in the next chapter of the story.

5.3 Plot Generation

This section expands the discussion in Section 5.2 on how plots are generated. **IPG** uses a partial-order planner, implemented in Prolog, adapted from [53] with extensions. A partial-order planner is better suited for plot generation because it uses a least-commitment strategy. Constraints (including the order of events) are established only when necessary, facilitating the conciliation of various goals. Features to permit the abandonment of goals were included, as well as constraint programming techniques [20] to deal with numerical pre-conditions not handled by standard Prolog.

The planner starts the generation of a plot with the user-specified goals or by inferring goals from the initial state, using the goal-inference rules. Given this initial input, the planner inserts events in the plot in order to fulfill the goals. When the planner detects that all goals have been either achieved or abandoned, the first stage of the process is finished.

IPG sends the partial plot thus generated to the **Plot Manager**, which presents it to the user. If the user does not like the partial plot, he may direct **IPG** to generate another alternative. If the user accepts the plot generated thus far, the process continues by inferring new goals from the situations generated in the first stage. If new goals are inferred, the planner is activated again to fulfill them. The process alternates goal-inference, planning and user interference until the user decides to stop or no new goal is inferred.

Notice that, in this process, we mix forward and backward reasoning. In the goal-inference phase, we adopt forward reasoning: situations in the past generate goals to be fulfilled in the future. In the planning phase, an event inserted in the plot to achieve a goal might have unsatisfied pre-conditions, handled through backward reasoning. To establish them, the planner might insert preliminary events with further unfulfilled pre-conditions, and so on.

User intervention is allowed at the beginning of the process and in between two simulation cycles. The user may interfere with plot generation by forcing the occurrence of events at certain times. For instance, in our running example, the user may insert “the wedding of the knight with the princess”. The user may also specify that some situations must be true at certain times along the plot, leaving it to the planner to expand the plot to bring about such situations. For example, the user may specify that “the knight will be weaker than the dragon at a certain time”. The planner then tries to conciliate both the inferred goals and the user-specified events and situations.

Plots are not restricted to incorporating only successful sub-plots. In trying to provide adequate means for handling negative interactions happening along a plot, we realized that the solution of conflicts and competitions sometimes requires the presence of fully or partially failed plans, which conventional planners would reject. When a goal is abandoned, events occurring prior to the abandonment must be kept as part of the plot, and thus influence its continuation. We use two main (extra-logical) mechanisms to handle goal abandonment and competitive plan execution: conditional goals and limited goals. 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 are tried once only, and have an associated limit (expressed as a natural number). The limit restricts the number of new events that can be inserted to achieve the goal.

Finally, the planner also handles a library of predefined plots that allows plot recognition by a method proposed by Kautz [28]. The method consists of matching a few intended events against the library, trying to find one or more plots of which the intended events may be part. This additional feature also serves to guide the user in his manual interventions, since, once a retrieved plot is found and displayed, the user can determine the insertion of one or more of its events into the plot being composed [13].

6. Concluding remarks

In this paper, we proposed to specify interactive storytelling genres with the help of static, dynamic and behavioral schemas, which respectively define: (a) the mini-world wherein the plots take place; (b) what events the participants can enact; and (c) what motives guide their behavior. The formalism was strongly influenced by a plan-recognition / plan generation paradigm [23], covering the *semantic* and *pragmatic* aspects as well, since:

(a) By feeding the mini-world factual description, provided by the static schema, into the definition of operations – expressing their pre-conditions and post-conditions in terms of such facts – we are able to determine the meaning of an entire plot, by simulating the successive state changes operated in the mini-world.
(b) On the other hand, plots do not emerge by blind chance; they are set in motion by the goals of the participants. They can be regarded therefore as intentional sequences of actions, coherent with the different inclinations of the characters involved.

Such goals often exhibit a mutual dependence, determined by certain peculiar conventions of fictional genres. The assigned role largely determines what kind of conduct is expected from a given character, which in turn can only be deployed if the other characters also act as they are supposed to, always in accordance to their respective 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, shown in the Appendix, the plots would fail to converge towards an appropriate outcome. Culler's insightful observation is helpful here (on page 209 of [17]): "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.

Although it seems adequate for characterizing genres where the stories exhibit a high degree of regularity, our proposal would not cope with the complexities of genres wherein the degree of variability is high. And, even for a genre that can still be treated, it would be presumptuous to claim that our specification would correspond exactly to the intuition of ordinary readers. We can define a genre G* merely as the set of plots P that our plan-based specification can recognize or generate. Surely we should try, as much as possible, to assess its closure with respect to 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. In particular, in this paper, we explored the combination of structuring constructs of literary origin (especially Propp's functions) with models familiar to Computer Science. All features described here have been tested through the **LOGTELL** [13] and **PlotBoard** [9] prototype tools.

References

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

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

[3] Bloom, H. (2003) *A Map of Misreading*. Oxford University Press, 2003.

[4] Borgida, A., and Brachman, R.J. (2003) “Conceptual Modeling with Description Logics.” In: Baader, F.; Calvanese, D.; McGuiness, D.L.; Nardi, D.; Patel-Schneider, P.F. (Eds) *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, UK.

[5] Calvanese, D.; De Giacomo, G. (2003) “Description Logics for Conceptual Data Modeling in UML”. In: Proc. 15th European Summer School in Logic Language and Information, August 2003, Vienna (Austria), pp. 18-29.

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

[7] Cervesato, I; Franceschet, M.; Montanari, A. (1997) “Modal Event Calculi with Preconditions”. In: Proc. 4th. International Workshop on Temporal Representation and Reasoning, Daytona Beach, FL, USA, pages 38-45, May 1997.

[8] Ciarlini, A.E.M. (1999) *Geração Interativa de Enredos*. Ph.D Thesis, Departamento de Informática, PUC-Rio, Brazil, 1999. (in Portuguese)

[9] Ciarlini, A.E.M.; Barbosa, S.D.J.; Casanova, M.A.; Furtado, A.L. (2009) "Event relations in plan-based plot composition". In: *ACM Computers in Entertainment*, Vol. 7, Issue 3.

[10] Ciarlini, A.E.M.; Furtado, A.L. (1999) “Simulating the Interaction of Database Agents”. In: Proc. DEXA’99 - Database and Expert Systems Applications Conference, Florence, Italy, Sept. 99.

[11] Ciarlini, A.E.M.; Furtado, A.L. (2002) “Understanding and Simulating Narratives in the Context of Information Systems”. In: Proc. 21st. International Conference on Conceptual Modeling, Tampere, Finland, Oct. 2002

[12] Ciarlini, A.E.M.; Casanova, M.A.; Furtado, A.L.; Veloso, P.A.S. (2007) "Treating Literary Genres as Application Domains". Technical Report MCC 19/2007. Dept. of Informatics, PUC-Rio.

[13] Ciarlini, A.; Pozzer, C.; Furtado, A.; Feijó, B. (2005) A Logic-Based Tool for Interactive Generation and Dramatization of Stories. In: Proc. ACM SIGCHI International Conf on Advances in Computer Entertainment Technology-ACE, 2005.

[14] Ciarlini, A.E.M. ; Camanho, M.M. ; Doria, T.R. ; Furtado, A.L. ;Pozzer, C.T. ; Feijó, B. (2008). "Planning and Interaction Levels for TV Storytelling". In: Proc. ICIDS'08 - 1st. Joint International Conference on Interactive Digital Storytelling, Erfurt, Germany.

[15] Chen, P.P. (1976) "The Entity-Relationship Model: Towards a Unified View of Data". *ACM Trans. on Database Systems,* v. 1, n. 1, p. 9 - 36.

[16] Costikyan, G. (2002) "I have no words and I must design: Toward a Critical Vocabulary for Games". In Proc. of Computer Games and Digital Cultures.

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

[18] Fikes, R.E.; Nilsson, N.J. 91971) "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". *Artificial Intelligence*, v. 2, n. 3-4, p. 189-208.

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

[20] Frühwirth, T. and Abdennadher, S. (2003) *Essentials of Constraint Programming*. Springer Verlag.

[21] Furtado, A.L. (1999) "Narratives and Temporal Databases: An Interdisciplinary Perspective". In: P.P.Chen, J.Akoka, H.Kangassalo, B.Thalheim (eds.), Conceptual Modeling: Current Issues and Future Directions, Springer-Verlag, 1999.

[22] Furtado, A.L.; Ciarlini, A.E.M. (1999) "Operational Characterization of Genre in Literary and Real-life Domains". In: Proc. 18th International Conference on Conceptual Modeling, Paris, France, November 1999.

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

[24] Furtado, A.L.; Ciarlini, A.E.M.; Feijó, B.; Pozzer, C.T. (2005) "Conceptual Modelling for Storytelling (with a Case Study)". Technical Report MCC 17/2005. Dept. of Informatics, PUC-Rio.

[25] Glassner, A. (2004) *Interactive Storytelling*. Natick: A K Peters, 2004.

[26] Grasbon, D.; Braun, N. (2001) "A morphological approach to interactive storytelling". In: Proc. CAST01, Living in Mixed Realities. Special issue of Netzspannung.org/ Journal, the Magazine for Media Production and Inter-media Research.

[27] Karlsson, B.; Ciarlini, A.E.M.; Feijó, B.; Furtado, A.L. (2006) "Applying a Plan-Recognition / Plan-Generation Paradigm to Interactive Storytelling". In: Proc. Workshop on AI Planning for Computer Games and Synthetic Characters, ICAPS06, English Lake District, 2006.

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

[29] Koch, P. (1999) "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.

[30] Lebowitz, M. (1985) "Story-telling as planning and learning". *Poetics*, v. 14, n. 6, 1985.

[31] Lloyd, W. (1987) *Foundations of Logic Programming*. Springer.

[32] Mateas, M.; Stern, A. (2000) "Towards integrating plot and character for interactive drama". In: Socially Intelligent Agents: the Human in the Loop, AAAI Fall Symposium.

[33] Mateas, M.; Stern, A. (2003) “Façade: An Experiment in Building a Fully-Realized Interactive Drama”. In: Proc. Game Developers Conference, p. 4-8, 2003.

[34] Meehan, J. (1977) “TALE-SPIN, an interactive program that writes stories”. In: Proc. 5th International Joint Conference on Artificial Intelligence, p. 91-98, 1977.

[35] Miller, R.; Shanahan, M. (1994) “Narratives in the Situation Calculus”. Journal of Logic & Computation, Vol. 4, Number 5, 1994.

[36] Oinonen, K.; Theune, M.; Nijholt, A.; Uijlings, J. (2006), "Designing a Story Database for Use in Automatic Story Generation". In: Proc. 5th International Conference on Entertainment Computing.

[37] Pearce, C. (2002) “Emergent authorship: the next interactive revolution”. In: *Computers & Graphics*, 26.

[38] Pozzer, C. (2005) *Um Sistema para Geração, Interação e Visualização 3D de Histórias para TV Interativa*. Ph.D Thesis, Departamento de Informática, PUC-Rio, Brazil, 2005.

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

[40] Reiter, R. (1978) “On Closed World Databases”. In Logic and Databases, H. Gallaire and J. Minker (eds.), Plenum Press, 1978, pp. 55-76.

[41] Riedl, M. (2004) *Narrative Planning: Balancing Plot and Character*. Ph.D Thesis, North Carolina State University, 2004.

[42] Riedl, M.; Young, R.M. (2004) “An intent-driven planner for multi-agent story generation”. In: Proc. of 3rd Int’l. Conf. on Autonomous Agents and Multi-Agent Systems.

[43] Rumelhart, D.E. (1975) “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.

[44] Schank, R.C.; Abelson, R.P. (1977) *Scripts, Plans, Goals and Understanding*. Hillsdale: Erlbaum, 1977.

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

[46] Turner, S. (1992) *MINSTREL: a computer model of creativity and storytelling*. Ph.D Thesis, Computer Science Department, University of California, 1992.

[47] Velasquez, J.D. (1997) “Modeling emotions and other motivations in synthetic agents”. In: Proc. 14th National Conference on Artificial Intelligence, Providence, 10-15, 1997.

[48] Ursu, M.F.; Thomas, M.; Kegel, I., et al. (2008) “Interactive TV Narratives: Opportunities, Progress, and Challenges”. ACM Transactions on Multimedia Computing, Communications, and Applications. Vol. 4, Issue 4 (October 2008).

[49] Wallis, J. (2008) *Making Games that Make Stories*. Electronic book review [www.electronicbookreview.com].

[50] Wardrip-Fruin, N.; Harrigan, P. (eds.) (2004). *First Person: New Media as Story, Performance, and Game*. The MIT Press.

[51] Warren, D.H.D. (1974) WARPLAN: a System for Generating Plans. Edinburgh: University of Edinburgh, Department of Computational Logic, memo 76.

[52] Young, R. (2001) “An Overview of the Mimesis Architecture: Integrating Intelligent Narrative Control into an Existing Gaming Environment. The Working Notes of the AAAI Spring Symp. on Artificial Intelligence and Interactive Entertainment, 2001.

[53] Yang, Q., Tenenberg, J., Woods, S. (1996) “On the Implementation and Evaluation of Abtweak”. In: *Computational Intelligence Journal*, 12(2): 295-318.

Appendix A - Example: a Swords and Dragons Genre

A.1 Example Scenario

The example scenario consists of 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, a knight called Hoel. The Gray Castle is the home of Hoel and of Brian, an even worthier knight. 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.

These creatures, both the persons and the dragon, can be described in terms of their good or evil nature and of their strength. As to nature, the princess and the knights are reputed to be on the side of goodness, whereas the dragon is evil; contrasting with all others, the magician is neutral. In the beginning, unsurprisingly, all creatures 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 creatures 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 subsequently kidnap the princess, and keep her under custody. The creatures are all in their homes at the beginning (with the single exception of Hoel), and 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.

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, the attacker often has to penetrate through the group of guardians surrounding the prospective victim’s present location; this may be quite hard, unless the victim foolishly dismisses a number of guardians. And the combat proper will consume the energies of a fighter. Is the attacker's strength enough to defeat and kill the adversary? If not, he should 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 ill-disposed when approached without due 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 a hero successfully frees her, then gratitude and admiration should change her inclination.

Many actions are closely associated with places. So, to say that a villain kidnaps a victim means that he brings her to his lair, and marriage can only be celebrated at a church. All characters, except donors, continually move across the scene to accomplish their missions.

The various characters are motivated 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. By contrast, Princess Marian does not even imagine that there may be any possibility of violence, and she 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, are impelled to either rescue or, in the worst case, avenge her. If released alive from captivity, the princess will be full of tender feelings for her savior. Both would love each other and would thus be anxious 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.

Finally, the magician Turjan does not seem to wish anything. He stands still in the forest, where people sometimes seek 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 initiates any plans. But, when one least expects, he can with a gesture transmute a kind person into a powerful evil creature.

A.2 Description of the mini-world – static schema

Figure 3 displays an ER diagram that helps understand the static schema. The static language contains the following database symbols:

- `creature`, an entity class, identified by `name`, with attributes `nature`, `strength`, `gender` and `alive` (of Boolean type)
- `person` and `dragon`, specializations of `creature`
- `princess`, `knight`, and `magician`, specializations of `person`
- `place`, an entity class, identified by `place_name` with a composite attribute `protection`, composed of `kind` and `level`
- `home` and `current_place`, two n-1 relationships between `creatures` and `places`
- `acquaintance`, a relationship involving `creature` twice, with attribute `affection`
- `married`, a 1-1 relationship involving `person` twice
- `kidnapped`, a n-1 relationship between `persons` and `creatures` (more than one `person` can be simultaneously held by one kidnapper)
- `hero`, `victim`, `villain` and `donor`, the roles adopted

For our purposes, we have not specialized `place`, but this is largely a matter of taste; one might readily come up with a variety of distinguishing criteria applicable to our scenario.

The choice of a convenient type for attribute values is crucial. For example, one could at first consider `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 A.3). An even more important choice was for the type of `affection`. Again, the intuitive preference might be some word indicating, for a pair of creatures *A* and *B*, in this order, the current feeling of *A* for *B*. Here, our choice was motivated by what is practically a consensus in affective computing [47] 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 gradual 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 used, we write all numbers as real numbers, although we are only concerned with integer values.

Our choice of roles – `hero`, `victim`, `villain` and `donor` – is a subset of the seven *dramatis personae* proposed by Vladimir Propp [39] 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`, we also allow in our specified genre that `knights` may figure as `victims` or `villains`. Any entity instance to which one or more roles have been assigned is a `character` (one of the *dramatis personae*) in the story.

A number of static integrity constraints are assumed. The most obvious is that whatever attribute a `creature` may have should only retain any significance while it is `alive`. All attributes here are single-valued. If a `creature` is playing the role of `villain`, his nature must be `-1`, whereas `heroes` and `victims`, who act as "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`. 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 in Figure 2 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`, here playing the role of `donor`, the `current_place` must coincide with his `home` at every state, a restriction that does not affect the other `creatures`. Moreover, `married` can only hold between `persons` of opposite gender.

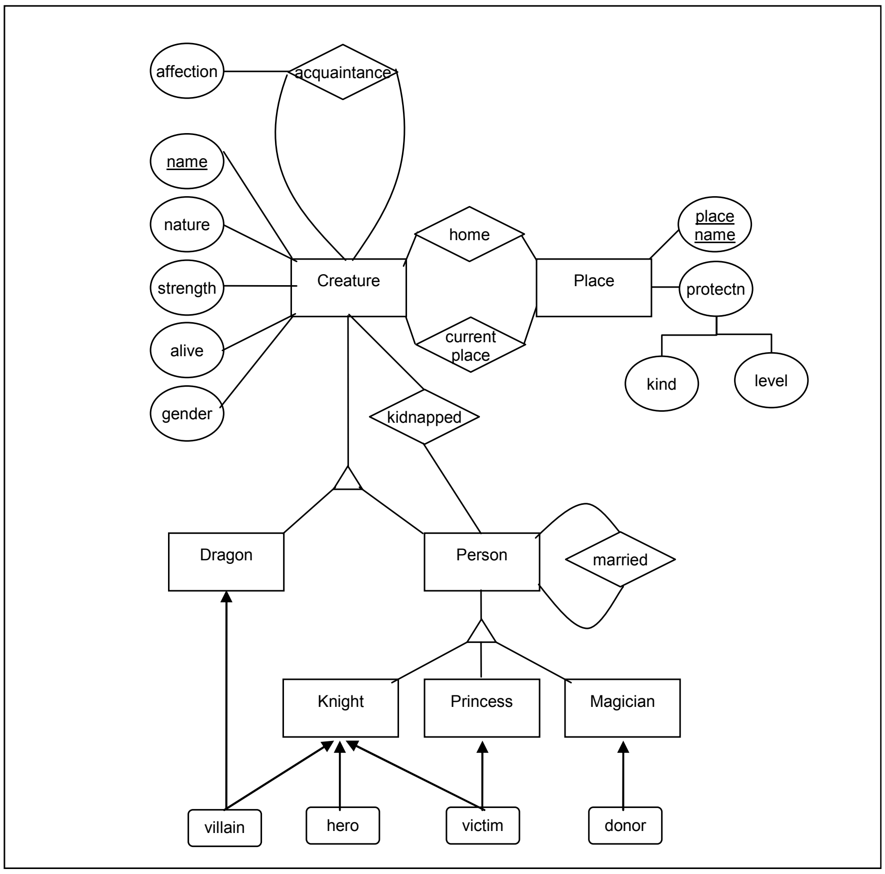


Figure 2: Entity-Relationship diagram

A genre is of course compatible with an ample choice of (valid) initial states. Different initial states lead to the creation of possibly very different narratives, all of which are constrained to remain within the limits of the defined genre. In our sample scenario, we assume that, in the initial database, the villainous Draco is stronger than the two `knights`, of whom Brian is the most vigorous, 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 neither good nor evil (`0` for `nature`).

The closed world assumption [40], familiar to database practioners, justifies the conclusion that no one is `married` or `kidnapped` at the initial state, simply because no such facts are explicitly recorded in the database.

A.3 Events that change the mini-world – dynamic schema

The dynamic schema of our Swords and Dragons genre has ten operations:

```
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 must not be in a `kidnapped` status, wherein his freedom to act would be necessarily limited. 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 (see [12] for the complete Prolog code):

- 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 the character `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.
- Only the potential `victim` can imprudently dismiss some of the guardians of the `place` where she currently is, as `agent` of the `reduce_protection(CH,PL)` operation, whose `object` is a `place`. The current 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`, in the required real-number format). The exact decrement will be determined at the *dramatization* stage.
- `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.

- 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 stands as the object of the action. The nature of the agent must be the contrary of the kind of protection of the attacked place. The current level of protection (associated with the number of guardians), which must be positive, 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).
- Two characters of opposite 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.
- 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.
- 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.
- In our version of marry(CH1,CH2), the agent CH1 must be a hero and the coagent, CH2, a victim, usually representing the proverbial motif of the "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 originally single. To acquire the married status, their presence at the Church is required.
- 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 a previous 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 the previous value was.
- 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 dreadful double effect of the operation is to instill an evil nature into CH2 who, at the same time, is made very strong (a strength of 100).

It is work some but not too hard to check how the combined interplay of pre-conditions and post-conditions in this repertoire contributes to the preservation of the static and dynamic integrity constraints, once the validity of the postulated initial state has been verified. As a trivial example with a static constraint, one can readily see that, at every state reachable from the initial state 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` beforehand, so as to reduce the `strength` of the adversary. 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 (or less). So, there is no need to fight if `CH2` already has `strength` no greater than 10. On the other hand, 10 is the minimum required to start fighting, which may induce an ill-advised character to challenge another with no chance to win (recall 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 fighting).

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* [2][17] should be included, plausible or not entirely so, as a prompt to adventurous deployments. For instance, this "maiden in distress" situation 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- 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 participants retain their `nature` throughout their lives. Again, fiction has a latitude that one would hardly admit in business application domains.

A.4 Motivation of the dramatis personae – behavioral schema

The dynamic behavior of our Swords and Dragons genre has six goal-inference rules and a library of typical plots.

The definition of each rule, in the notation of Section 4.2, is followed by a brief discussion.

- The first two in our example are activated right at the initial state. The first rule refers to the `heroes`. At least one `hero` should be prepared for future missions and so, if there exists some `villain` stronger than him, he will try to acquire an even superior `strength`.

```
/* (1) The strongest hero wants to become stronger than the villain */
 (villain(VIL)∧ strength(VIL,Lv) ∧
  hero(HERO) ∧ strength(HERO,Lh)
  ⇒ ◊(strength(HERO,LS) ∧ LS > Lv)
```

- The second rule applies to the `victim`. It is very common in folktales that a `victim` can be blamed as partly responsible for the villainy 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 sets its reduction as a goal. As already seen in the pre- and post-conditions of the 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. A different variable, `PLACE2`, denotes the location of the `victim` at future time; this allows *two* possibilities for achieving less `protection`: the planner can either apply (one or more times) the `reduce_protection` operation to the original `PLACE1` – 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`.

```
/* (2) Victim reduces protection of her current location
*/
 (victim(VIC) ∧ nature(VIC,KIND0) ∧
  current_place(VIC,PLACE1) ∧
  protection(PLACE1,KIND1,PROT1))
  ⇒ ◊(current_place(VIC,PLACE2) ∧
       protection(PLACE2,KIND2,PROT2) ∧
       KIND2*KIND0*PROT2 < KIND1*KIND0*PROT1)
```

- If the goal of the second rule is reached, the third rule is triggered, producing in the `villain` a desire to take advantage of the weaker 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 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 (described later).

```
/* (3) If victim's protection is reduced, villain will want to
   kidnap her */

 (victim(VIC) ∧ nature(VIC,KIND0) ∧
  current_place(VIC,PLACE1) ∧
  protection(PLACE1,KIND1,PROT1))
  ⇒ □(current_place(VIC,PLACE2) ∧
       protection(PLACE2,KIND2,PROT2) ∧
       KIND2*KIND0*PROT2 < KIND1*KIND0*PROT1
         ⇒ ◊(alive(VIC) ∧
              kidnapped(VIC,VIL)))
```

- 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 can accomplish the deed, but the overall specification of the genre calls for their participation, effectively excluding any other character.

```
/* (4) If victim is kidnapped, hero will want to rescue her */
□ (kidnapped(VIC,VIL) ⇒ ◊¬kidnapped(VIC,VIL))
```

- The fifth rule applies to 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. If both this rule and third rule 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 plot generation algorithm.

```
/* (5) If victim is killed, hero will want to avenge her */

□((victim(VIC) ∧ villain(VIL) ∧ kill(VIL,VIC))⇒ ◊¬alive(VIL))
```

- The sixth and last rule, if ever activated, will 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` 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 restricts marriage to a hero and a victim, roles that are respectively assigned in the example initial state to each `knight` and to the `princess`, thus enforcing the opposite gender requirement.

```
/* (6) If the affection between two persons is high, then they will
   want to get married */

□ ((affection(CH1,CH2,L1) ∧ affection(CH2,CH1,L2) ∧
    ¬married(CH1,X) ∧ ¬married(CH2,Y) ∧ L2>95.0 ∧ L1>95.0)
   ⇒ ◊married(CH1,CH2))
```

The library of typical plots is organized in *is-a* and *part-of* hierarchies. Figure 4 shows both hierarchies, where single arrows denote composition (*part-of* link) and double arrows denote generalization (*is-a* link) (we refer the reader to [27] for a detailed description of the notation adopted).

As shown in Figure 3, the library has 4 levels of operations, numbered from 0 to 3, the level 3 being occupied by the basic operations introduced in Section A.3:

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

- Level 1 – `rescue`, `avenge`: These are the two specializations of `adventure`. The `rescue` variety has components: `abduct`, `liberate`, `marry`, `accompany`, `donate`. The other variety, `avenge`, has components: `murder`, `execute`, `accompany`, `donate`. As Figure 2 shows, there are connecting edges to only some of the components; such edges are unnecessary for `accompany` and `donate`, which 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, the pair murder and execute for avenge) in fact overrules the *is-a* implicit inheritance discipline. 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: Operation 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 main 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 actions listed here, and noting that kill should rather be reserved as a prerogative of the main hero. A clever false_helper is likely to enter the battlefield only when the struggle is over, and surreptitiously 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 (level 4) from this hierarchy. Pervasive as it is when physical events are contemplated, operation go is in fact an ultimate component of practically all actions, and therefore is assumed to be present even if not indicated explicitly. On the other hand, bewitch was deliberately excluded. As noted before, plots including bewitch are to be considered *transgressive* rather than typical in the context of our genre. They reveal the magician's inclination to subvert an until then innocent world, by acting as a *trickster*.

A.5 Example of a Plot

Figure 4 shows a simple plot generated by the plan generation prototype (a number of other examples are shown in [24]). The contents of the boxes indicate the executed operations (and also the goals, in "gen_goal" clauses) prefixed by numerical tags, used internally to record the partial order requirements. The connecting edges are manually inserted by the user to choose a fully linearized sequence compatible with the partial order requirements, which must be done as a preliminary step to *dramatization*.

Upon traversing the plot, a simple-minded template-based facility can "read" it and produce the coarse text of Figure 5. The resulting animation is illustrated in Figure 6.

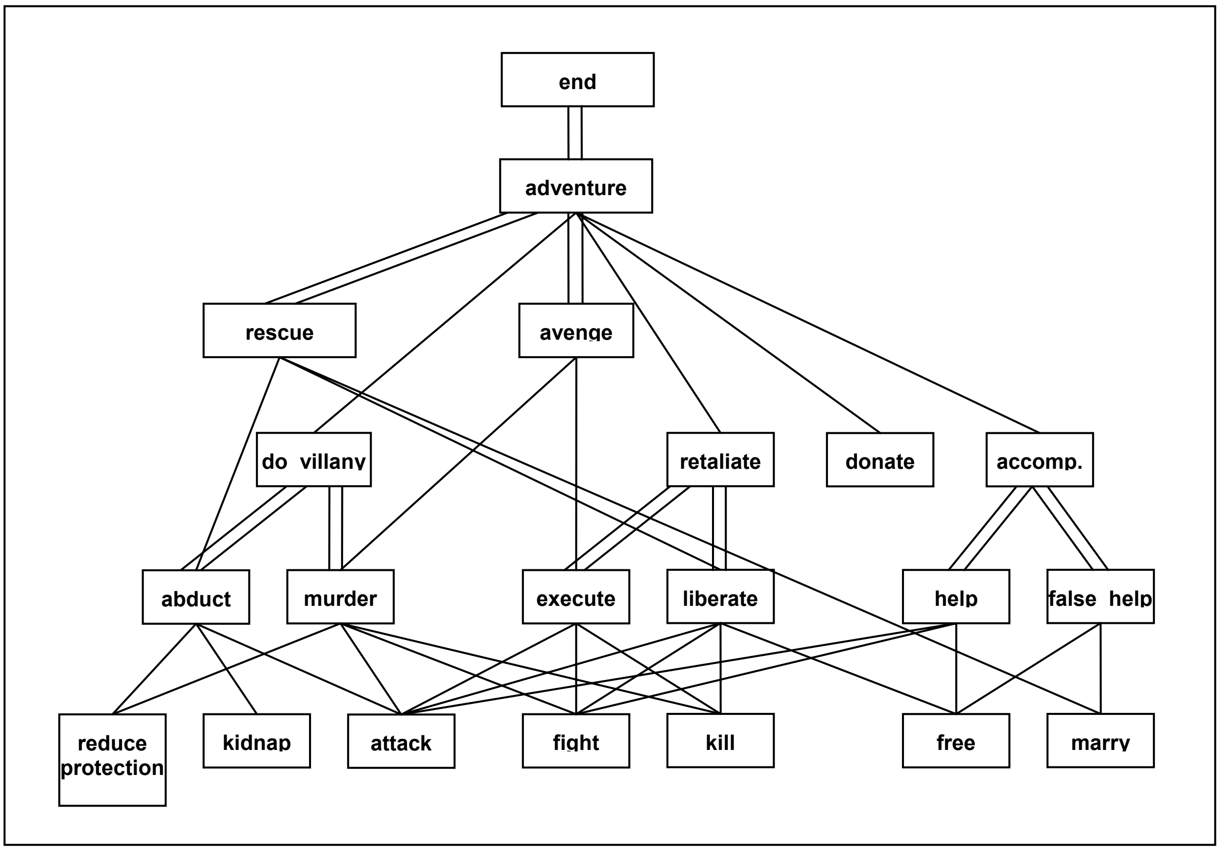
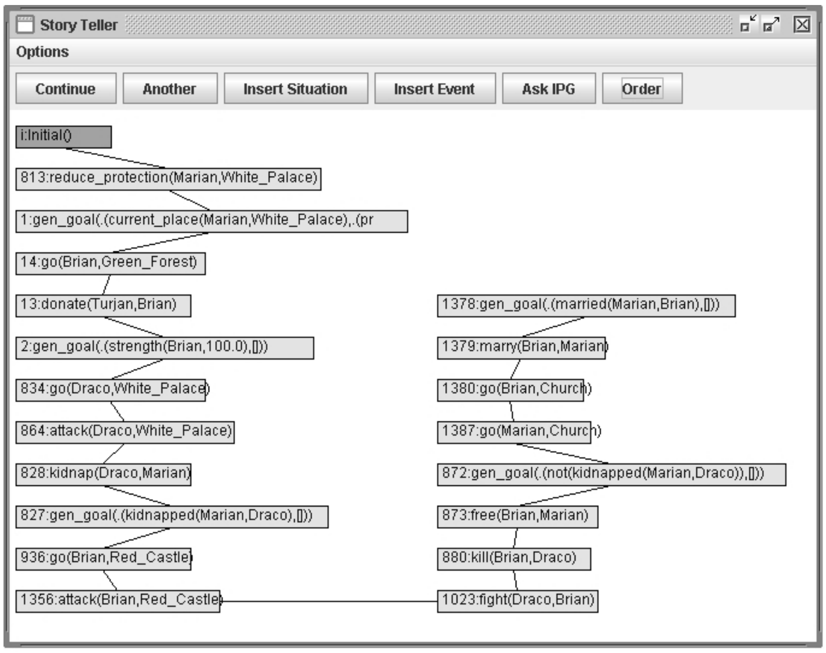


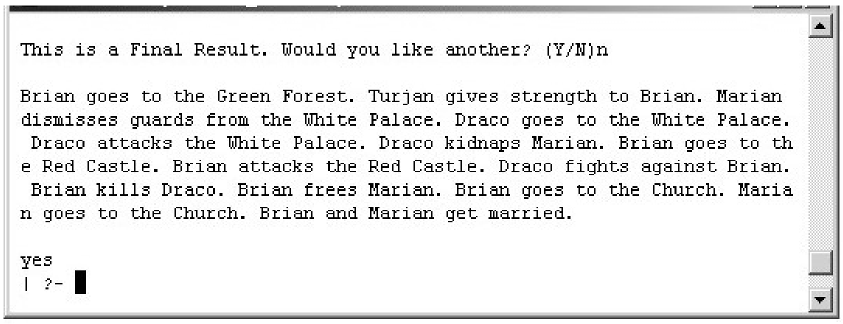
Figure 3: Hierarchy of typical plans.



```
i:Initial()
813:reduce_protection(Marian,White_Palace)
1:gen_goal(.(current_place(Marian,White_Palace),.(pr
14:go(Brian,Green_Forest)
13:donate(Turjan,Brian)
2:gen_goal(.(strength(Brian,100.0),[]))
834:go(Draco,White_Palace)
864:attack(Draco,White_Palace)
828:kidnap(Draco,Marian)
827:gen_goal(.(kidnapped(Marian,Draco),[]))
936:go(Brian,Red_Castle)
1356:attack(Brian,Red_Castle)

1378:gen_goal(.(married(Marian,Brian),[]))
1379:marry(Brian,Marian)
1380:go(Brian,Church)
1387:go(Marian,Church)
872:gen_goal(.(not(kidnapped(Marian,Draco)),[]))
873:free(Brian,Marian)
880:kill(Brian,Draco)
1023:fight(Draco,Brian)
```

Figure 4: Example plot.



```
This is a Final Result. Would you like another? (Y/N)n

Brian goes to the Green Forest. Turjan gives strength to Brian. Marian
dismisses guards from the White Palace. Draco goes to the White Palace.
 Draco attacks the White Palace. Draco kidnaps Marian. Brian goes to th
e Red Castle. Brian attacks the Red Castle. Draco fights against Brian.
 Brian kills Draco. Brian frees Marian. Brian goes to the Church. Maria
n goes to the Church. Brian and Marian get married.

yes
| ?-
```

Figure 5: Template-based text.

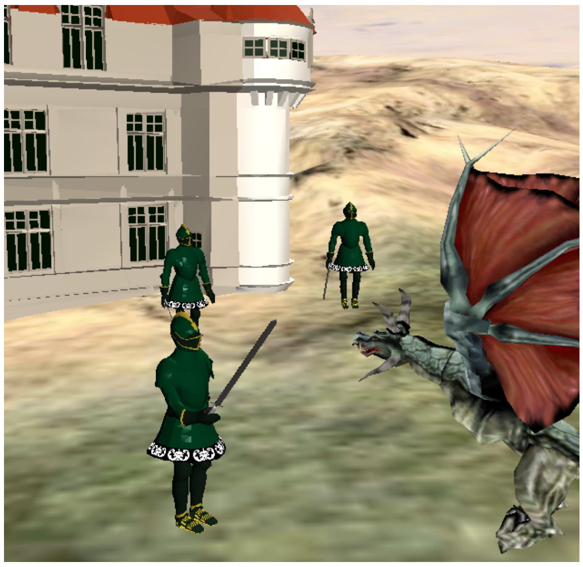


Figure 6: Draco attacks the White Palace.