Generating Educational Interactive Stories in Computer Role-playing Games

Marko Divéky, Mária Bieliková

Institution of Informatics and Software Engineering,
Faculty of Informatics and Information Technologies,
Slovak University of Technology, Ilkovičova 3, 842 16 Bratislava, Slovakia
markod@nextra.sk, bielik@fiit.stuba.sk

Abstract. The aim of interactive storytelling is to tell stories with the use of computers in a new and interactive way, which immerses the reader inside the story as the protagonist and enables him to drive its course in any desired direction. Interactive storytelling thus transforms conventional stories from static structures to dynamic and adaptive storyworlds. In this paper, we describe an innovative approach to interactive storytelling that utilizes computer role-playing games, today's most popular genre of computer games, as the storytelling medium in order to procedurally generate educational interactive stories.

Keywords: Interactive storytelling, story generation, educational stories, computer games, role-playing games

1 Introduction

Stories and the art of storytelling have played an inevitable role in our lives ever since the earliest days of language. Historians found first records of storytelling in ancient cultures and their languages, in which stories served as the vehicle by which cultural knowledge was communicated from one generation to the next [1].

Even in today's modern times, such utilitarian use of stories has endured and their educational potential is utilized in many different areas, e.g., in business to spread knowledge amongst employees in order to help them become more productive and to collaborate with one another [2]. Stories are also used in many other ways: in policy, in process, in pedagogy, in critique and as a foundation and as a catalyst for change [3]. The reason why storytelling proves to be an effective medium for educating is that our minds are programmed much more for stories than for abstract facts [4].

Many aspects of stories have changed since the very first records of storytelling in ancient times. However, even in today's modern world, the fundamental idea behind *conventional stories* remains unaltered. Let us suppose that the storyteller seeks to communicate some truth or information (principle) to a desired audience. Instead of just telling the principle, the storyteller translates it into an instantiation (a story), then communicates the story to the audience, which in turn translates the instantiation back into the principle [1], as depicted in Fig. 1.

Fig. 1. The process of telling a conventional story.

Interactive storytelling differs from conventional stories in a way that the process of transforming the principle into the instance (story) is delegated to the computer. It is best described as "a narrative genre on computer where the user is one main character in the story and the other characters and events are automated through a program written by an author. Being a character implies choosing all narrative actions for this character" [5].

In interactive storytelling, the computer transforms the principle into a *storyworld*, which operates on rules rather than on predefines events (see Fig. 2). A single playing of a storyworld generates a single story. In other words, when a player goes through a storyworld, he produces a linear sequence of events that makes a story. Different playing of the storyworld can yield many different stories, but all of them share one common principle [1], as depicted in Fig. 2.

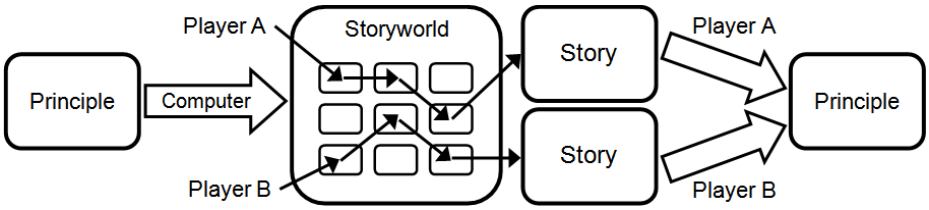

Fig. 2. The process of playing through an interactive storyworld.

In this paper, we propose an approach that enables to generate educational interactive stories in *computer role-playing games*. Researchers in the field of interactive storytelling often disregard computer games as a suitable storytelling medium [1]. Still, some researchers see them as the ideal form for storytelling given their practically unlimited degree of interactivity and visual attractiveness [6].

Throughout the many diverse genres of computer games available today, role-playing games (RPGs) have the most detailed and involving storyline, thus being the most appropriate genre for storytelling [7]. Moreover, computer role-playing games have originally derived from tabletop role-playing games and are considered as being one of today's most popular genres of computer games [7].

2 Related Work

Despite the large amount of work that has already been done in the field of interactive storytelling, there have been only a few working solutions that found practical use

other than being proof-of-concept demonstrations. However, it is possible to divide the most notable solutions into three categories described below:

- *Systems that generate complex, but only text-based interactive stories*, e.g., Storytron [8] (formerly Erasmatron) developed by Chris Crawford since 1991. Storytron utilizes a drama manager agent to direct the generated stories, which are, however, very cumbersome to define and have only text-based visualization.
- *Systems that generate visually attractive, but not interactive stories*, e.g., I-Storytelling [9] developed by Marc Cavazza and Fred Charles at the University of Teesside. Their solution utilizes Hierarchical Task Network (HTN) planning and Heuristic Search Planning (HSP) to generate stories via autonomous behavior of character agents. Users, however, have very limited or no means of interacting with and directly influencing the generated stories.
- *Systems that generate interactive stories, but only with a single dramatic conflict*, e.g., Façade [10] developed by Michael Mateas and Andrew Stern. Façade uses Natural Language Processing (NLP) to parse user-inputted actions and visualizes stories with a custom proprietary 3D graphics engine. Façade solves almost all problems that relate to interactive storytelling; however, it is able to generate only a single dramatic scene.

Consequently, our goal is to devise a new approach to interactive storytelling that not only builds on present-day techniques and formalisms in a way that eliminates the above-mentioned drawbacks of existing interactive storytelling solutions, but also enables to generate educational interactive stories in computer role-playing games.

3 Overview of the Proposed Concept

The aim of the hereby-described concept is to programmatically generate educational interactive stories with computer role-playing games as their medium, therefore combining the dynamic and enthralling storyworlds created by interactive storytelling with the visual appearance, gameplay and popularity of computer role-playing games. The presented concept can be broken into the following three logical layers:

- *Character Behavior Layer* describes interpersonal relationships and conditional reasoning of characters,
- *Action Planning Layer* realizes planning and replanning of narrative actions,
- *Visualization Layer* utilizes computer role-playing games for story visualization.

All generated interactive stories initiate in the topmost logical layer named the *Character Behavior Layer*, since all stories are about people, despite the fact that references to them are often indirect or symbolic [1]. The behavior of characters results in creating plans and planning actions that move the story forward. The middle layer, called the *Action Planning Layer*, handles all the planning and replanning. All created plans eventually break down into actions that are visualized by the lowest logical layer called the *Visualization Layer*.

All three logical layers operate on easy-to-define rules and data structures, which are described in detail in the following sections.

3.1 Visualization Layer

The *Visualization Layer* provides graphical visualization of the generated interactive stories to the players. It uses the concept of computer role-playing games as the visualization and storytelling medium. The most important aspects of computer role-playing games that are important for the scope of this paper are described below.

Themes. Games based on the role-playing genre are most often set in a fictional fantasy world closely related to classic mythology, or in a science-fiction world set somewhere in the future. Historical and modern themes are also common [11].

Avatars. In role-playing games, a player controls one in-game character called the *avatar*, and uses him as an instrument for interacting with the game world [12].

Character Development. Besides having the option to fully customize the appearance of his in-game character, the player is allowed to choose various attributes, skills, traits and special abilities that his avatar will possess. These are given to players as rewards for overcoming challenges and achieving goals, most commonly for completing *quests*. Character development plays, together with stories, a key role in today's computer role-playing games.

Quests. A *quest* in role-playing games can be defined as a journey across the game world in which the player collects *items* and talks to *non-player characters* "in order to overcome challenges and achieve a meaningful goal" [13]. Quests often require the player to find specific items that he needs to correctly use or combine in order to solve a particular task, and/or require the player to choose a correct answer from a number of given answers to a certain question [14]. Upon solving a quest, the player is often presented with a reward, which can have many forms – i.e. ranging from a valuable item to a new skill, trait or ability for the player's avatar.

Many quests are optional, allowing for freedom of choice in defining the player's goals. Moreover, a set of quests may be mutually exclusive with another set, therefore forcing the player to choose which set of quests he will solve, having in mind the possible long-term effects these quests will have on the game world. Some quests can be solved in more than just one way and thus bring non-linearity into the game.

What is noteworthy and important to realize with regard to the focus of this work is the fact that quests are a fundamental structure by which the player moves the storyline forward in computer role-playing games. In other words, a quest is a conceptual bridge between the open structure of role-playing games and the closed structure of stories, thus being an ideal vehicle for interactive storytelling in computer role-playing games. Consequently, it is possible to use computer role-playing games as a medium for interactive storytelling by dynamically generating non-linear quests.

Non-player Characters. Role-playing game worlds are populated by *non-player characters* (NPCs) that cannot be controlled by the player. Instead, their behavior is scripted by the game designers and executed by the game engine. Players interact with non-player characters through dialogue. Most role-playing games feature branching dialogue (or *dialogue trees*). As a result, when talking to a non-player character, the player may choose from a list of dialogue options where each choice often results in a different reaction. Such choices may affect the player's course of the game, as well as latter conversations with non-player characters.

Items, Containers and Objects. Throughout playing role-playing games, quests require the player to find, collect and properly use various *items* scattered throughout the game world. Special items may be equipped on the player's avatar, improving his abilities, skills or other attributes. All items can be either created from other items, or obtained from *containers*, i.e. *objects* that can hold or carry items. The player himself is an example of a container, since he can carry items in his inventory. Non-player characters and objects representing treasure chests are also examples of containers.

Atomic Actions. We have formalized computer role-playing games from an interactive storytelling point of view into a set of *atomic actions* having narrative impact that a player (or a non-player character) is able to commit inside the game world. Each atomic action has the following structure:

- *Name:* A string concisely describing the atomic action.
- *Source:* The type of an in-game element that can commit the atomic action, e.g., the player, or a non-player character.
- *Target:* The type of an in-game element that this atomic action is committed upon, e.g., an object or a container.
- *Parameter:* The type of an in-game element that the atomic action operates on. The presence of the parameter is optional.
- *Preconditions:* A set of statements regarding the specified source, target and parameter defining the circumstances under which the atomic action can be committed in the game world.
- *Effects:* A set of changes to the game world that are a consequence of the atomic action having been committed.

The typical set of atomic actions that describes a common computer role-playing game is shown in Table 1.

Table 1. Atomic actions that describe a typical computer role-playing game.

Source	Atomic Action[1]	Target	Description
Character	`GIVE (Item)`	Container	The character specified as the source gives the item to a targeted container.
Character	`TAKE (Item)`	Container	The source character takes the item from the target container.
Character	`USE (Item)`	Element	The atomic action's source character uses the item on the targeted element.
Character	`EQUIP ()`	Item	The character specified as the source equips the targeted item.
Character	`WALK_TO ()`	Element	The source character walks near the targeted element.
Character	`TALK_TO ()`	Character	The atomic action's source initiates a dialogue with the targeted character.

[1] The atomic actions are written in a shortened textual form with the following structure: `NAME (parameter).`

As an example, we elaborate the second atomic action mentioned in Table 1, which is defined as follows:

Name:	`TAKE`
Source:	Character
Target:	Container
Parameter:	Item
Preconditions:	Target has parameter
Effects:	Target ~~has parameter~~ (negation of the action's precondition) Source has parameter

3.2 Action Planning Layer

From a top-down perspective, the role of the middle logical layer is to transform character goals set by the *Character Behavior Layer* into plans consisting of actions that are to be executed and visualized by the bottommost logical layer – the *Visualization Layer*. From a bottom-up perspective, the *Action Planning Layer* is responsible for processing actions committed by the player and all non-player characters that were reported back from the *Visualization Layer*.

Since the described process is done on-the-fly while the player is playing a computer role-playing game, the hereby described layer creates new quests based on the previous actions committed by the player and seamlessly integrates them into the game, thus dynamically creating an interactive story perceived by the player.

The *Action Planning Layer* utilizes Hierarchical Task Network planning in a similar way as described in [15]. Our algorithm searches for appropriate actions based on recursive matching of their effects with required preconditions. In other words, the planner finds all actions resulting in the required preconditions (see section 3.4).

The *Action Planning Layer* operates on *simple actions*, *complex actions* and *action bindings*, all of which are described below.

Simple Actions. A *simple action* refines the usage of exactly one atomic or simple action by specializing its source, target, parameter or by adding additional preconditions or effects. Unlike atomic actions, simple actions can alter *attributes* of characters and *interpersonal relationships* – features of the *Character Behavior Layer*. Every simple action consists of the following structure:

- *Name:* A string concisely describing the simple action.
- *Base action:* An atomic or simple action that this simple action refines.
- *Source:* The base action's source type or its subtype (see below).
- *Target:* The base action's target type or its subtype.
- *Parameter:* The type of an in-game element that this simple action operates on. The parameter is equal to or a subtype of the base action's parameter.
- *Preconditions:* A set containing preconditions inherited from the base action plus additional preconditions related to this simple action.

— *Effects:* A set containing effects inherited from the base action plus additional effects related to this simple action.

Below is an example of a simple action named `REPAIR` that refines the atomic action `USE` (mentioned in Table 1) and enables the player to repair a malfunctioning computer with a spare circuit board:

Name:	`REPAIR`
Base action:	`USE`
Source:	Player (a character subtype)
Target:	Computer (an object subtype, see below)
Parameter:	Circuit board (an item subtype)
Preconditions[2]*:*	[Source has parameter] Target is "malfunctioning"
Effects[2]*:*	[Source has parameter] Target is "malfunctioning"

The second effect marks the target computer as "not malfunctioning," since it is defined as being an object subtype, which has a "malfunctioning" property defined. An object instance can belong to multiple object subtypes, each having defined custom properties that can be set on the object instance.

Likewise, subtypes of all other core in-game elements (described in section 3.1), i.e. items, containers and characters are also supported, with each subtype having its own custom properties defined.

Complex Actions. A *complex action* encloses multiple actions in a sequence with atomic, simple or complex actions being its elements. Every complex action has the following structure:

— *Name:* A string concisely describing the complex action.
— *Source:* Type of an in-game element that is the source of all enclosed actions.
— *Targets:* Types of in-game elements that are targets of the enclosed actions.
— *Parameters:* A set of in-game elements that the enclosed actions operate on. Each parameter is identified by a unique name distinguishing it from the others.
— *Enclosed actions:* A set containing atomic, simple or other complex actions that this complex action encloses.
— *Preconditions:* A filtered[3] set containing preconditions of all enclosed actions plus additional preconditions related to this complex action.
— *Effects:* A filtered set containing effects of all enclosed actions plus additional effects related to this complex action.

[2] The preconditions and effects inherited from the base action are enclosed in [square brackets].
[3] The sets do not contain preconditions nor effects that are created and at the same time annulled during the sequential execution of actions enclosed by the complex action.

Below is an example of a complex action, with which a non-player character unlocks a locked item for the player if the particular non-player character likes the player:

Name:	`UNLOCK_LOCKED_ITEM_FOR_PLAYER`
Source:	Non-player character
Targets:	Player Lockable item `i` (an item subtype)
Parameters:	Lockable item `i`

Enclosed actions[4]*:*	`Source : TAKE (i) → Player`
	`Source : UNLOCK() → i`
	`Source : GIVE (i) → Player`

Preconditions:	[Player has `i`] [`i` is "locked"] Source "likes" player
Effects:	[Player has `i`] [`i` ~~is "locked"~~]

Action Bindings. The purpose of optional *action bindings* is to explicitly permit or disallow bindings of atomic, simple, and complex actions to actual in-game entities. Action bindings therefore define what can or cannot be done with all defined story elements, and what exactly can or cannot the player and all non-player characters do.

For example, the action bindings given below[4] denote that the player can steal the item "thyme syrup" from the non-player character representing an evil doctor and cannot steal it from a non-player character representing a good doctor:

- `[CAN] Player : STEAL ("Thyme syrup") → "Evil Doctor"`
- `[CAN'T]Player : STEAL ("Thyme syrup") → "Good Doctor"`

3.3 Character Behavior Layer

The *Character Behavioral Layer* is responsible for generating goals of all in-game characters, i.e. the player and all non-player characters. These goals are created according to *interpersonal relationships* among the player and non-player characters by matching their existing values[5] to a set of *behavior patterns* (see below) and picking the resulting goals from the best matching patterns. All generated characters' goals are afterwards translated to plans by the *Action Planning Layer*.

Behavioral Patterns. A *behavioral pattern* defines the circumstances that lead to a change in the behavior of characters from a narrative point of view. The set of all

[4] Each action is written in the form: `Source : NAME (`*parameter(s)*`) → Target`.

[5] The storyteller sets the initial values of interpersonal relationships, if necessary. Otherwise, all relationships are initialized to their default values.

behavioral patterns defines the conditional reasoning of all in-game characters. Each behavioral pattern has the following structure:

- *Description:* An optional text concisely describing the behavioral pattern.
- *Preconditions:* A set of in-game elements, including their properties and relationships among them. Also included are the preconditions of actions that represent goals. Undesirable preconditions may be explicitly excluded.
- *Goals:* A set containing goals describing the change in characters' behavior as a consequence to the situation portrayed by the pattern's preconditions. Goals can be represented by actions of any type, or by changes in properties of in-game elements and in relationships among them. Each goal is bound to a character.
- *Effects:* A set containing effects of all identified goals. Undesirable effects may be explicitly excluded from the set.

Below is an example behavioral pattern that describes the situation in which John, a husband of Mary with a respiratory infection, cures his wife with thyme syrup that the player had given him:

Preconditions:	Non-player characters "John" and "Mary" John is "husband" to Mary, Mary is "wife" to John Mary is "sick with respiratory infection" […preconditions of the two actions that represent goals…]
Goals:	`Player : GIVE ("Thyme syrup") → John` `John : USE ("Thyme syrup") → Mary`
Effects:	[…effects of the two actions that represent goals…] Mary ~~is "sick with respiratory infection"~~

This behavioral pattern contains examples of two actual *interpersonal relationships*, namely "is husband to," "is wife to" and an *attribute* "sick with respiratory infection." Following is a detailed description of all types of interpersonal relationships and attributes that the *Character Behavior Layer* operates on.

NPC → Character Relationships. *Interpersonal relationships* oriented from a non-player character[6] to any type of in-game character (i.e. the player or another non-player character) are identified with their unique label.

Under normal circumstances, NPC → Character relationships are either absent (the default initial value) or present[7]. However, there are cases when two relationships are mutually exclusive (meaning that the presence of one disallows the presence of the other and vice-versa). Such pairs of relationships are merged into *relationships with*

[6] Relationships oriented from the player are not considered, because monitoring such relationships and limiting the number of narrative actions the player can commit based on their presence would degrade the player's experience of the generated stories.

[7] Certain types of NPC → Character relationships can be set to have a numerical value instead of being either present or absent. Such mathematical relationships require additional apparatus for expressing preconditions that is not described in this paper.

two complementary values. These interpersonal relationships can have only one of their two values present at a time, e.g., if the precondition "Tom likes player" is true at a given time, then the precondition "Tom dislikes player" is false at the same time.

Character Roles. Another type of relationships between in-game characters are *character roles*. Unlike NPC → Character relationships, character roles can be oriented from the player, as well. Moreover, character roles can be bilateral.

Similarly to NPC → Character relationships, character roles are either absent (the default initial value) or present. If a *bilateral character role* is present, then preconditions testing either end evaluate to true, e.g., "John is husband to Mary" and "Mary is wife to John" are either both true, or both false in a given time.

Character Attributes. Every in-game character, whether being the player or a non-player character can have various *attributes* defined. Such attributes are either unnumbered or numbered.

Unnumbered character attributes do not have any numerical value and are either absent (the default initial value) or present. An example of an unnumbered character attribute is "sick with respiratory infection" referenced in the aforementioned behavioral pattern example.

Numbered character attributes are present in every in-game character (i.e. the player and all non-player characters) and store a numerical value (individually for every character and from a custom-defined interval), unlike unnumbered attributes. An example of numbered attributes found in the majority of computer role-playing games is located in the left column of Table 2.

Table 2. Examples of various numbered character attributes.

Common Attributes	Domain-specific Attributes
– Agility	– C++ proficiency
– Endurance	– Java proficiency
– Intelligence	– Lisp proficiency
– Strength	

In addition, attributes of characters can be easily made domain-specific, e.g., an interactive storyworld created in the domain of teaching programming languages can have defined numbered character attributes located in the right column of Table 2.

3.4 The Story Generation Cycle

The three logical layers described in the preceding sections are cyclically used to generate interactive stories, as depicted in Fig. 3.

Preparation Phase. Before any interactive stories can be generated, the author of the stories, i.e. the storyteller creates his domain-specific rules and input data based on which the proposed storytelling system operates. In other words, the storyteller defines the necessary simple and complex actions, action bindings, types of possible character properties (attributes, relationships and roles between characters) and behavioral patterns – as described in the previous sections.

Initialization Phase. After having defined all necessary custom domain-specific rules comes the initialization phase, in which the storyteller defines the storyworld that he wants the generated stories to take place in. He therefore creates the desired non-player characters along with their initial attributes, roles and relationships between them. The storyteller can also optionally scatter objects, containers and items throughout the storyworld as desired.

Story Generation Phase. After the storyteller had defined the initial state of the storyworld, the *Character Behavior Layer* analyzes the storyworld and chooses one or more goals that best match its current state, i.e. goals having all preconditions met.

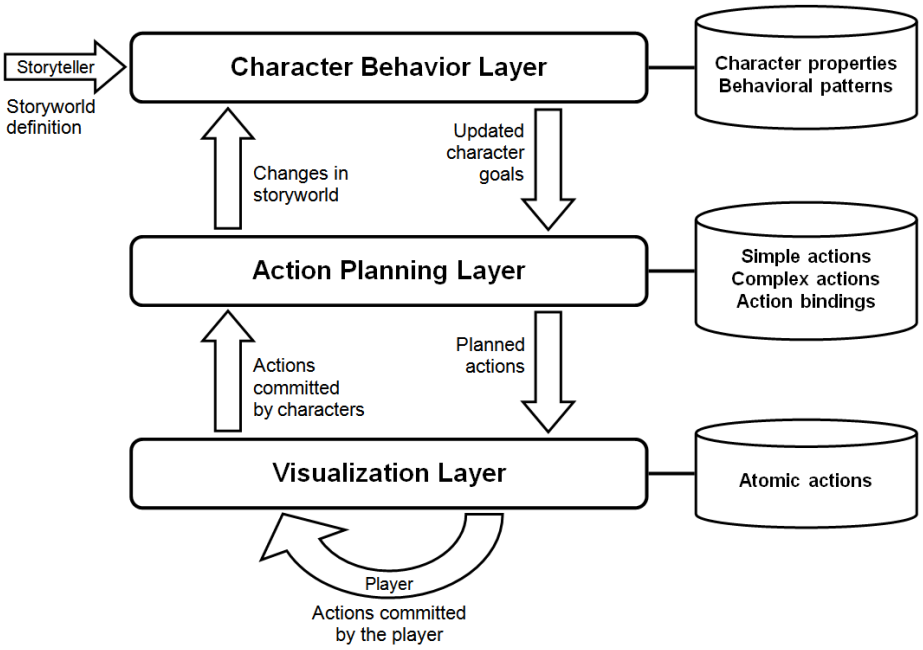

Fig. 3. The story generation cycle.

Such goals are afterwards processed by the *Action Planning Layer*, which translates all player's and non-player characters' active goals to plans. The planning process is based on the Hierarchical Task Network (HTN) planning formalism that recursively finds appropriate actions, effects of which meet the required conditions.

The planning process starts out with the effects of each active goal and finds any complex, simple or atomic actions (in this order) that have their effects match the goal's effects. The whole planning process afterwards runs recursively to search for actions that have their effects match the preconditions[8] of the newly found actions.

[8] Since the planner matches the required preconditions with effects of candidate actions, all actions depicted in Fig. 4 have preconditions placed below them and effects above.

As shown in Fig. 4, the resulting plan for every active goal is a tree-like graph that contains multiple paths of complex, simple or atomic actions, which result in accomplishing the particular goal when followed in the storyworld.

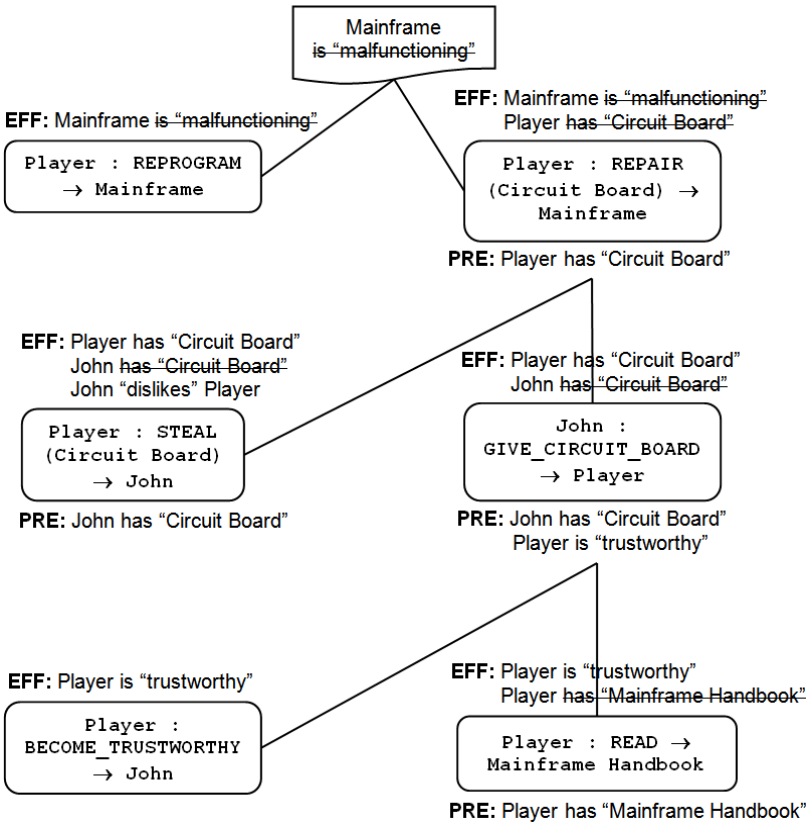

Fig. 4. An example plan[9] consisting of six simple actions that are interconnected by common preconditions (PRE) and effects (EFF), with the goal being the plan's root. The plan is constructed from top to bottom, but carried out from bottom to top.

After plans for the player and all non-player characters are constructed, a subset of all available player's planned paths is chosen and delegated to the *Visualization Layer*, along with any planned actions that each non-player character should commit.

The subset of the player's plan is chosen according to the player's user model, which contains records of all actions that he had previously successfully committed in

[9] The plan is from a storyworld in an educational domain related to the history of computing and programming basics, in which the player is to repair a malfunctioning mainframe computer, either by reprogramming it (see Fig. 6) or by repairing it with a spare circuit board, which he can either steal or get from John, a non-player character, upon becoming trustworthy by successfully answering a mainframe-related question or by reading a mainframe handbook.

other storyworlds. Thanks to these records, it is possible to filter out and ignore planned paths containing similar actions that the player previously successfully committed (*positive personalization*), or actions he had not yet committed or failed to commit successfully (*negative personalization*). If the player's user model contains no records, then the subset of player's planned paths is selected randomly.

In the example plan shown in Fig. 4, presuming positive personalization and that the player's user model states that the player had previously committed the simple action `REPROGRAM`, then the path with the simple action `REPROGRAM` is chosen and delegated to the *Visualization Layer*, as shown in Fig. 5.

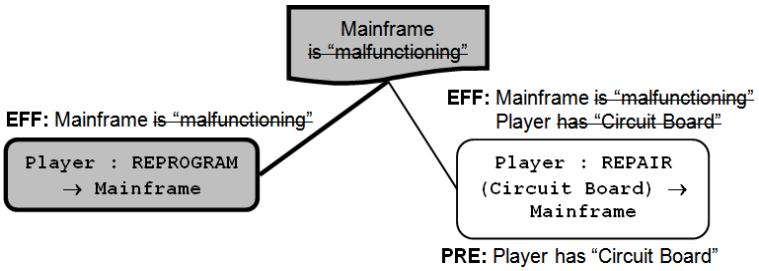

Fig. 5. An example of a chosen path, colored grey (the rest of the player's plan is omitted).

The *Visualization Layer* receives planned actions for both the player and non-player characters (NPCs) from the *Action Planning Layer* and the actions destined for NPCs are committed by the particular NPCs, whereas the actions planned for the player are presented to him as multiple options via the game's graphical interface.

The player commits, either successfully or unsuccessfully, his desired action (that may not have been planned, but must have its preconditions met), what is afterwards signaled back to the *Action Planning Layer*, along with actions that were, either successfully or unsuccessfully, committed by any non-player characters.

All committed actions are processed by the *Action Planning Layer* that matches all committed actions to player's and non-players' existing plans, which remain either unaffected, or are replanned according to the new state of the storyworld, i.e. new planned paths are again chosen based on the player's user model, as described in the preceding paragraphs.

Any alterations to the state of the storyworld, such as changes in characters' attributes, roles or relationships are signaled to the *Character Behavior Layer*, which collects all changes to the storyworld from the *Action Planning Layer* and determines whether any of the existing goals are affected (in terms of having been accomplished or not being accomplishable anymore), or whether preconditions for any new goals are met. The current set of both the player's and non-player characters' goals is updated and any changes are delegated back to the *Action Planning Layer*.

The hereby-described process is repeated in a cyclic manner until the player has successfully accomplished all of his goals, in which case the story ends. An ending also occurs if the player is unable to accomplish all of his existing goals and there are no more possible paths left to do so.

4 Conclusions and Future Work

We have described an innovative approach to interactive storytelling that uses techniques common in this field in such a unique way that, in contrast to all existing solutions, operates on simple-to-define rules, and enables to programmatically generate interactive stories in computer role-playing games.

We have implemented a prototype system based on the approach described in this paper. The domain that we have selected for prototyping is education related to the history of computing and programming basics.

The prototype system reads from an input file a set of rules and data structures (described in detail in [16]) defining a storyworld from which an educational interactive story is generated and presented to the player through a computer role-playing game (see Fig. 6). The story progresses on-the-fly by reacting to narrative actions that the player had committed.

Fig. 6. Screenshots from an educational interactive story generated by the prototype system. The top-left picture shows the player while reprogramming a mainframe computer by constructing valid program code from multiple choices of possible commands. The bottom-right picture depicts an example environment in which the generated interactive stories take place.

Our future work consists of evaluating the prototype system empirically by players who will fill out questionnaires regarding various narrative and educational aspects of the generated interactive stories.

Acknowledgments. This work was partially supported by the Cultural and Educational Grant Agency of the Slovak Republic, grant No. KEGA 3/5187/07 and by the Scientific Grant Agency of Slovak Republic, grant No. VG1/0508/09.

References

1. Crawford, C.: Chris Crawford on Interactive Storytelling. New Riders Games, (2004)
2. Denning, S.: The Springboard: How Storytelling Ignites Action in Knowledge-Era Organizations. Butterworth-Heinemann, (2000)
3. Sandercock, L.: Out of the Closet: The Importance of Stories and Storytelling in Planning Practice. Planning Theory & Practice, vol. 4, no. 1, 11–28 (2003)
4. Schank, R.: Tell Me a Story: Narrative and Intelligence. Northwestern University Press, Evanston (1995)
5. Szilas, N.: The Future of Interactive Drama. In: Proceedings of the Second Australasian Conference on Interactive Entertainment, pp. 193–199. Creativity & Cognition Studios Press, Sydney (2005)
6. Murray, H.J.: From Game-Story to Cyberdrama. FirstPerson: New Media as Story, Performance and Game, pp. 2–11. MIT Press, Cambridge (2004)
7. Rollings, A., Adams, E.: Andrew Rollings and Ernest Adams on Game Design. New Riders Games, (2003)
8. Storytron, http://www.storytron.com/
9. Cavazza, M., Charles, F., Mead, S.J.: Sex, Lies, and Video Games: An Interactive Storytelling Prototype. In: Proceedings of the 2002 AAAI Spring Symposium, pp. 13–17. AAAI Press, Menlo Park (2002)
10. Façade. http://www.interactivestory.net/
11. Barton, M.: Dungeons and Desktops: The History of Computer Role-playing Games. A K Peters Ltd, Wellesley (2008)
12. Tychsen, A.: Role Playing Games – Comparative Analysis Across Two Media Platforms. In Proceedings of the Third Australasian Conference on Interactive Entertainment, pp. 75–82. Murdoch University, Perth (2006)
13. Howard, J.: Quests: Design, Theory, and History in Games and Narratives. A K Peters Ltd, Wellesley (2008)
14. Bieliková, M., Divéky, M., Jurnečka, P., Kajan, R., Omelina, Ľ.: Automatic Generation of Adaptive, Educational and Multimedia Computer Games. Signal, Image and Video Processing, vol. 2, no. 4, 371–384 (2008)
15. Cavazza, M., Charles, F., Mead, S.J.: Planning Characters' Behaviour in Interactive Storytelling. Journal of Visualization and Computer Animation, vol. 13, no. 2, 121–131 (2002)
16. Divéky, M., Bieliková M.: An Approach to Interactive Storytelling and its Application to Computer Role-playing Games. In Návrat, P., Chudá, D., (eds.) Znalosti 2009: Proceedings of the eighth annual conference, pp. 59–70. STU, Bratislava (2009)