Directorial Control in a Decision-Theoretic Framework for Interactive Narrative

Mei Si, Stacy C. Marsella, and David V. Pynadath

Institute for Creative Technologies
University of Southern California
Marina del Rey, CA 90292
meisi@ict.usc.edu, marsella@ict.usc.edu, pynadath@ict.usc.edu

Abstract. Computer aided interactive narrative has received increasing attention in recent years. Automated directorial control that manages the development of the story in the face of user interaction is an important aspect of interactive narrative design. Most existing approaches lack an explicit model of the user. This limits the approaches' ability of predicting the user's experience, and hence undermines the effectiveness of the approaches. Thespian is a multi-agent framework for authoring and simulating interactive narratives with explicit models of the user. This work extends Thespian with the ability to provide proactive directorial control using the user model. In this paper, we present the algorithms in detail, followed by examples.

1 Introduction

With the rapid development of computer technology, a new form of media – interactive narrative has received increasing attention in recent years. Interactive narrative allows the user to play a role in a story and interact with other characters controlled by the system. It has been widely applied for both pedagogical and entertainment purposes [1–10].

The support of interactivity distinguishes interactive narrative from other narrative forms. In traditional narratives, the relation of the audience (or reader) to the narrative is always passive. By allowing the user to interact, interactive narrative provides a potentially more engaging experience. Moreover, because different choices of the user can result in different stories, the author can tailor the experience for individual users.

On the other hand, user interactivity introduces tremendous challenges to the design process. As the author cedes partial control of the story to the user, it is much harder to control the development of the story for creating the author's desired effects [11]. For example, in non-interactive narratives, the dramatic effects are created by imposing conflicts and tensions on the characters and resolving them over the course of the story. However, the user in an interactive narrative has control over their character and may act to avoid such conflict and tension.

To control the development of the story in the face of user interaction, automated directorial control (drama management) is often applied. It continuously adjusts the story, so that the story's development is both coherent and leads to the author's desired effects.

Various approaches for directorial control have been proposed. In search-based approaches, the drama manager operates over a set of plot points with pre- and postconditions [12, 13, 4, 14]. Based on an author-specified evaluation function and in reaction to the user's actions, the drama manager reconfigures the story world to achieve best quality in the story. In contrast, in Mimesis the system may prevent the user's action from being effective [5]. When the user's action deviates from the pre-computed story plan, the system either replans or makes the user's action having no effect on story development. Similarly, in IDA [7], when there is a problem with the flow of the story, the director agent tries to get the story back on track. Façade [10] utilizes a beat-based drama management system, where beat is the smallest unit of a story. Based on a desired global plot arc, the drama manager chooses the next beat that is suitable to the context and whose dramatic value best matches the arc.

Most existing approaches do not model the user explicitly (IDA [7] is a notable exception.) The lack of a user model restricts the effectiveness of their directorial control in several ways. In existing works, directorial controls are often applied based on rules predefined by the author for a "standard user," and therefore cannot be adaptive to individuals who may react to the events differently. Further, the coherence of narrative, which requires the events in the story to be meaningfully connected in both temporal and causal ways [15], is crucial for ensuring that people can understand their experience [16, 17]. A key aspect of creating coherent narratives is that the characters' behaviors must be interpretable to the user. In interactive narratives, it is hard to avoid generating unnatural characters' behaviors when interacting with the user without a model of the user's beliefs and experiences. In addition, user modeling opens the possibility of simulating the user as a means to test the interactive narrative system [18].

Thespian [19, 3] is a multi-agent framework for interactive narratives. Thespian models each character in the story as a decision-theoretic goal-driven agent, with the character's personality/motivations encoded as the agent's goals. The user is also modeled using an agent, based on the character which the user takes the role of [18]. In modeling the user, not only the goals of the user's character are considered, but also the goals associated with game play, e.g. exploring the environment (see [18] for details.) This model allows other agents to form mental models about the user the same way as about other characters and the system to reason about the user's beliefs and experience.

Thespian facilitates the user's understanding of their experience in two ways. First, the ability of goal-based agents to decide their actions based on both the status of the interaction and their goals makes Thespian agents react to the user and behave with consistent motivations. Secondly, Thespian agents possess a "Theory of Mind" [19] and can model emotions [20] and social normative behaviors [21]. The Theory of Mind capacity allows the agents to reason about others' beliefs, goals and policies when deciding their own actions. Along with the modeling of emotions and social normative behaviors, these capabilities make the agents' behavior seem more socially aware and life-like.

This work extends Thespian to provide proactive directorial control. A director agent is designed to monitor the progress of the story, predict its future development and adjust the virtual characters' behaviors and beliefs if necessary to prevent violations to directorial goals. The evaluation of the achievements of directorial goals is tied to the model of the user. In addition, the adjustments to the virtual characters do not break their appearance of acting with consistent motivations. We present the algorithms in detail, followed by examples.

2 Example Domain

In this paper, the "Little Red Riding Hood" story is used to demonstrate our approach for directorial control. The user plays the role of the wolf. The story starts as Little Red Riding Hood (Red) and the wolf meet each other on the outskirt of a wood while Red is on her way to Granny's house. The wolf has a mind to eat Red, but dares not because there are some wood-cutters close by. The wolf, however, will eat Red at other locations where nobody is around. Moreover, if the wolf hears about Granny from Red, it will even go eat her. Meanwhile, the hunter is searching the wood for the wolf. Once the wolf is killed, people who were eaten by it can escape.

3 Thespian's Current Architecture

We developed Thespian as a multi-agent framework for authoring and simulating interactive narratives. Thespian is built upon PsychSim [22], a multi-agent system for social simulation based on Partially Observable Markov Decision Problems (POMDPs) [23].

3.1 Thespian Agent

Thespian's basic architecture uses decision-theoretic goal-based agents to control each character in the story, with the character's personality/motivations encoded as the agent's goals. Each agent can have multiple and potentially conflicting goals with different relative importance (weight). For example, the wolf can be modeled as having the goals of not starving itself and also keeping itself alive, with the latter being ten times more important. An agent's state keeps track of the agent's physical and social status in the story. State is defined by a set of state features, such as degree of hunger, whether being alive, and degree of affinity with another character. The values of state features can be changed by both the agent's own actions, e.g. eat, and other characters' actions, e.g. being killed. Thespian agents have beliefs (subjective view) about itself and others, which forms a "Theory of Mind". An agents' beliefs can also include other agents' subjective views of the world, a form of recursive agent modeling [24]. For example, the wolf can have beliefs about Red's beliefs about the wolf's goals. Currently, for decision-making all agents use a bounded lookahead policy. They project limited steps into the future to evaluate the effect of each available action, and choose the one with the highest overall expected utility (the sum of utilities at each future step). Because Thespian agents have a "Theory of Mind", it considers other agents' responses, and in turn its own responses when evaluating the utility of an action option.

3.2 Fitting Characters' Motivations

Thespian's fitting procedure [25, 19] allows the author or the director agent (see Section 4 for details) to configure the goals of an agent using linear story paths (sequence of characters' actions). The fitting procedure can tune the agent's goal weights according to the agent's role in the story paths, so that the agent's autonomous behaviors follow the story paths when the user's behavior also follows the paths. When the user deviates from the paths, the agent will use the same goals weights "learned" from the paths to motivate its actions.

The fitting procedure automatically extracts constraints on the agent's goal weights from the story paths and determines whether consistent goal preferences can be inferred, i.e. whether the same goals preferences can drive the agent to act as specified in the story paths. When fitting fails, it is impossible for a character with consistent motivations to follow all the story paths. The author or the director agent will need to modify either the paths or the design of the agent.

3.3 Suggest Changes to Character's Beliefs

Similar to fitting, the "Suggest" procedure can adjust an agent's configuration so that the agent prefers the action desired by the author or the director agent over other choices. "Suggest" achieves this functionality in the opposite way to fitting. In fitting, the relative goal weights of the agents are adjusted, and the agent's beliefs and state are untouched. The "suggest" procedure suggests changes to the agent's beliefs without affecting its goals. For example, to make the wolf not eat Granny, fitting may result in the wolf having a very low goal weight of not starving itself, and the "suggest" procedure may return a suggestion that the wolf should believe it is not hungry.

4 Directorial Control

Thespian utilizes a specialized agent – a director agent – to realize directorial control. Different from other agents, the director agent is not mapped to an on-screen character. The director agent also has accurate beliefs about all other agents including their beliefs about each other. In contrast, for modeling narratives it is often necessary for characters to have incorrect beliefs about each other. The function of the director agent is to monitor the progress of the story and adjust the virtual characters' behaviors if needed to achieve directorial goals. When the director agent is functioning, it takes over other agents' decision-making processes, decides the best movements for the story and causes other agents to perform the corresponding actions.

The director agent has a model of the user, which assumes that the user identifies with the character, and therefore adopts the character's goals to a certain degree. Directorial control is performed based on the director agent's expectations about the user's experience and choices. Of course, the user can always act unexpectedly to the director agent, but this does not affect the director agent's workflow (see Section 4.3 for details.)

4.1 Design Challenge and Approach

Thespian agents are goal-based agents. Their behaviors are decided by their beliefs and goals. Adjustments to the agents' behaviors often require adjustments to their beliefs and goals. Though the characters' belief changes are not directly visible, improper belief changes may result in the characters having sudden and unnatural behaviors, and may lead the user to interpret the characters as inconsistent. Therefore, the basic challenge for directorial control in a system that uses autonomous characters is how to ensure that the virtual characters exhibit consistent motivations throughout the story while modifying their beliefs and goals. Further, the characters' motivations should be consistent with the author's portrayals of the characters in the story paths used for configuring (fitting) the agents.

To address this challenge, the basic assumption is that the user will not have a precise mental model about the characters because the user's observations in the story will not support such precision. Typically a range of configurations of a character can be used to explain its behaviors. Therefore, as long as the adjustments to the character's goals and beliefs fall within the space of the user's mental model about the character, the user will not experience inconsistency in the character. The boundary of the range, i.e. how precise the user's mental model about the character is, is decided by the user's prior interactions with the character and the user's prior beliefs about the character. In this work, we use fitting-based procedures to decide whether a belief change is reasonable to happen for a character, and whether an action is consistent with a character's motivations exhibited in its prior interactions with the user, and with the author's design of the character.

4.2 Directorial Goals

Directorial goals are used by the author to indicate how they want the story to progress, such as when an action should happen, or a character should change its belief about another character. Thespian currently supports directorial goals expressed as a combination of temporal and partial order constraints on the characters' including the user's actions and beliefs. Table 1 lists the syntax for specifying directorial goals. Six different types of goals are supported. The events in the syntax can be either an action, e.g. "wolf-eat-Granny" or a character's belief, e.g. "wolf: wolf's hunger = 0 (the wolf believes that the value of the wolf's state feature hunger is 0) ." The author can combine any number of goals defined using this syntax. Table 2 gives an example of directorial goals.

4.3 Director Agent

Directorial control is applied every time after the user performs an action. Function **Directorial_Control** (Algorithm 1) contains the pseudo code for the overall workflow of directorial control.

Table 1. Syntax for Specifying Directorial Goals

orders =	[*event*1,*event*2]
	*event*2 should happen after *event*1
earlierThan =	[*event*,*step*]
	event should happen before *step* steps of interaction
laterThan =	[*event*,*step*]
	event should happen after *step* steps of interaction
earlierThan2 =	[*event*1,*event*2,*step*]
	*event*2 should happen within *step* steps after *event*1 happened
laterThan2 =	[*event*1,*event*2,*step*]
	*event*2 should happen after *step* steps after *event*1 happened
NoObjIfLater =	[*event*,*step*]
	if *event* hasn't happen after *step* steps of interaction, the constraint for it to happen if exists, does not apply any more

Table 2. Directorial Goals Example

orders =	[[“wolf-eat-Granny”, “anybody-kill-wolf”],
	[“wolf-eat-anybody”, “wolf: wolf's hunger = 0”]]
earlierThan =	[[“wolf-eat-red”, 50], [“wolf-eat-Granny”, 80]]
earlierThan2 =	[[“wolf-eat-Granny”, “anybody-kill-wolf”, 30]]
laterThan2 =	[[“wolf-eat-red”, “wolf-eat-Granny”, 10]]
NoObjIfLater =	[[“wolf-eat-red”, 60]]

Overview of the Workflow

The director agent maintains a list of objectives that it will try to reach. Each objective indicates the desirability of an event, such as “hunter-kill-wolf is desirable”, or “Red: Red's alive = 0 is undesirable”. Initially, this list is empty (line 1 in Algorithm 1). After each time the user acts, the director agent simulates *lookaheadSteps* steps of interaction in the future (line 8), examines whether the future development of the story is consistent with the author's directorial goals, and creates a list of objectives if it foresees violations (line 9).

Table 3 lists the objectives that will be created in function **Test_Violation** for violating each type of directorial goals. In general, if a partial order constraint is expected to be violated, the director agent will try to prevent the latter event from happening; if a temporal constraint is expected to be violated, the director agent will try to arrange the event to happen or not happen according to the constraint. For example, based on the directorial goals described in Table 2, if the wolf has not eaten Red by 50 steps of the interaction, a violation happens, and the objective of “wolf-eat-Red is desirable” will be added. As the last step of **Test_Violation**, the “NoObjIfLater” goals are applied – if there is an objective for *event* to happen, and *step* steps of interaction have already passed, the objective will be taken out.

Algorithm 1 DIRECTORIAL_CONTROL()

```
1: objectives ← [ ]
2: bestOption ← [ ]
3: minViolation ← ∞
4: futureSteps ← [ ]
5: for each i in range(Num_of_Tests) do
6:    if objectives != [ ] then
7:       Adjust_Config(objectives)
8:    futureSteps ← Lookahead(lookaheadSteps)
9:    objectives, numViolation ← Test_Violation(futureSteps)
10:   if numViolation < minViolation then
11:      bestOption ← futureSteps
12:      minViolation ← numViolation
13: return bestOption
```

When the list of objectives is not empty (line 6 in Algorithm 1), the director agent will try to tweak the characters' configurations for reaching the objectives. Function **Adjust_Config** (Algorithm 2) adjusts the characters' beliefs and goals for inducing actions or preventing actions from happening as indicated in the objectives. Adjusting the characters' beliefs often requires a fictional action to happen for inducing the belief change (see Algorithm 4 for details) and therefore may have side effects. Therefore, preference is given to adjusting the characters' goals for achieving the objectives. Function **Adjust_Config** first tries to fit the characters' goals to achieve the objectives. If none of the objectives can be reached this way, it will try to change the characters' beliefs and then fit the characters' goals again. If the objectives involve the user's actions (either being desirable or undesirable), the director agent will try to affect the user's decisions only by changing his/her beliefs.

As part of the process for adjusting the characters' beliefs, whether a belief change is reasonable is tested, and how to make the belief change happen is proposed. The same process is used for reaching the objectives that specify constraints on the characters' beliefs. The details of this process are given in Algorithm 4 and related discussions. However, the rest of this section is organized around how to achieve objectives regarding the characters' actions.

Table 3. Objectives if Directorial Goals are Violated

Violated Goal	Desirable Actions	Undesirable Actions
orders =[*event*1,*event*2]		*event*2
earlierThan =[*event*,*step*]	*event*	
laterThan =[*event*,*step*]		*event*
earlierThan2 =[*event*1,*event*2,*step*]	*event*2	
laterThan2 = [*event*1,*event*2,*step*]		*event*2

Algorithm 2 ADJUST_CONFIG(*objectives*)

```
1: fitting_result ← Fit_To_Objectives (objectives)
2: if fitting_result == false then
3:    beliefChanges ← Find_Suggestions(objectives,futureSteps)
4:    for each beliefChange in beliefChanges do
5:       if Find_Explanation(beliefChange) then
6:          Apply_Belief_Changes(beliefChange)
7:    Fit_To_Objectives(objectives)
```

Algorithm 3 FIT_TO_OBJECTIVES(*objectives*)

```
1: history: interaction history
2: paths: list of story paths designed by the author for configuring the characters
3: success ← false
4: for each objective in objectives do
5:    actor ← objective.action.actor
6:    if objective.desirable then
7:       all_paths = paths + history.append(objective.action)
8:       return Fit(actor, all_paths)
9:    else
10:      for each action in actor.actionOptions do
11:         if action != objective.action then
12:            all_paths = paths + history.append(action)
13:            if Fit(actor, all_paths) then
14:               success ← true
15: return success
```

After making all the adjustments, the director agent will again test whether the directorial goals will be violated in future interactions using lookahead projection (line 8-9 in Algorithm 1). This iterative process will stop when a satisfactory solution has been found or the maximum number of attempts has been reached, in which case the characters will act according to the $futureSteps$ with minimal violations of directorial goals.

Fitting Characters' Goals to Objectives

Function **Fit_To_Objectives** fits the agents' goals to the objectives. For each objective that contains a desirable action, the function tests to see whether it is possible for the action to happen. For each objective that contains an undesirable action, the function tests to see whether it is possible for the actor of that action to do something else, and possibly nothing. The function also tests whether doing an action is consistent with the author's design of the character by considering the story paths designed by the author for configuring (fitting) the characters.

Slightly different from the fitting procedure used in authoring, here it is okay for the utility of the desired action to be the same as other actions' utilities. In this case, even though the agent is not guaranteed to choose the action, which

is not ideal for authoring, the action is a reasonable choice for the character's goals. In fact, for characters who do not have a clear personality/motivation in the story, the constraints on utility can be further relaxed to accommodate more action options for the director agent. In the extreme case, the director agent can directly order the characters without testing whether the suggested actions are consistent with the characters' motivations, in which case the fitting procedure always returns true.

Adjusting Characters' Beliefs
When **Fit_To_Objectives** alone cannot achieve any objectives, the director agent will try to tweak the characters' beliefs. Function **Find_Suggestions** in Algorithm 2 calls the "suggest" procedure for each objective and for each character, and merges the results into a list of all the suggestions to the characters' beliefs. For example, to make Granny give some cake to the wolf, Granny having cake and being at the same location as the wolf are likely to be suggested. These suggestions make the achievement of the objectives possible, but do not guarantee it. To test the effect of applying the belief changes, one needs to either simulate lookahead projections or try to fit the characters to the objectives.

Further, whether the belief changes are reasonable needs to be tested before they can be applied. Function **Find_Explanation** looks for an action that is reasonable to happen at the moment or in the past, and can explain the belief change. If such action exists, even though the user does not see the action happening, the user may assume that it happened when he/she was not there. Therefore, changes in the characters' behaviors, because of their belief changes, will not seem sudden and unnatural to the user. See Example II in Section 5 for an example. Of course, the belief change cannot be caused by a user's action, because the user knows what he/she has done in the past. If the director agent wants the user's belief to change, it needs to arrange the action that can cause the belief change to actually happen. For example, for the user to believe that the hunter is close by, the director agent needs to let the hunter appear at the user's location. In general, all the suggested belief changes need to be tested by **Find_Explanation**. The author can make exemptions by specifying state features whose values are not important to the user. For example, in the Little Red Riding Hood story, the locations of the characters who are outside of the user's sight can be changed freely by the director agent.

5 Examples

This section provides two step-by-step examples of applying directorial control.

Example I starts by a potential violation of directorial goals being detected: the wolf will eat Red before Red gives the cake to Granny. The corresponding objective is added to the list: the director agent "wants" the wolf to choose an action other than eating Red. Since the wolf is played by the user, the director agent skips fitting the wolf's motivations for achieving the objective, and directly tries to change the characters' beliefs. It finds that if the wolf believes that the hunter is close by, the wolf will choose a different action than eating Red.

Algorithm 4 FIND_EXPLANATION(*beliefChange*)

```
1: history: interaction history
2: paths: list of story paths designed by the author for configuring the characters
3: for each character in story do
4:    if ! character == user then
5:       for each action in character.actionOptions do
6:          all_paths = paths + history.append(action)
7:          if Fit(character, all_paths) then
8:             if Effects(action) == beliefChange then
9:                return true
10: return false
```

Algorithm 5 Directorial Control: Example I

Lookahead projection :"wolf-eat-Red", "Red-doNothing", "hunter-walk" ...
Detect goal violation : order: ["red-give-cake-Granny","wolf-eat-red"]
Add objective : ["wolf-eat-Red", "undesirable"]
Adjust beliefs : hunter's location = wolf's location
Simulate user's lookahead : "wolf-run"
Request characters to act : "hunter-walk-towards-wolf"

Assuming the belief change has happened, the director agent tests for potential goal violations again with lookahead projection. This time the director agent expects no violations. It then proceeds by applying the belief change – in this case by physically re-locating the hunter – and ordering the characters, other than the wolf, to act according to its last lookahead projection. After the user responds, the director agent will start another round of directorial control, starting by detecting potential goal violations.

The basic procedure in Example II is the same as in Example I, and therefore will not be repeated. Note that in this example because the directorial goal is defined with the keyword "anybody", the director agent can try to make any character kill the wolf. Often, there are many alternatives for adjusting the characters. The author can specify priorities, e.g. always try to adjust Red's configurations first. Otherwise the system will randomly pick an order. In this example, the director agent starts by trying to make Red kill the wolf. It failed

Algorithm 6 Directorial Control : Example II

Lookahead projection : "wolf-walk", "Red-walk", "hunter-walk" ...
Detect goal violation : earlierThan2: ["wolf-eat-Granny", "anybody-kill-wolf", 30]
Add objective : [("anybody-kill-wolf", "desirable")]
Fit characters to the objectives : failed to fit Red
Adjust beliefs : Red's location = wolf's location; Red's power > wolf's power
Find explanation : "hunter-give-gun-Red" → Red's power > wolf's power
Fit characters to the objectives : succeed, "Red-kill-wolf"
Request characters to act : "Red-walk-towards-wolf"

to fit Red's motivations. It then proceeds to adjust the characters' beliefs. It finds that when Red is next to the wolf and believes that she is stronger than the wolf, Red can be fitted to do the action. The director agent also has tested whether the belief changes are reasonable. **Find_Explanation** returns that in order for Red to believe that she is stronger than the wolf, the hunter should have given Red a gun and this is a reasonable action for the hunter. The director therefore decides the belief changes to Red are feasible to happen.

6 Future Work

Our future work is planned in three difference directions. First, we will enrich the syntax for specifying directorial goals so that more complex constraints can be expressed, such as constraints that are contingent on the current state of the story. Secondly, we will allow the author to specify priorities among the directorial goals. So that when there are conflicts among the goals, the director agent can make decisions based on the importance of the goals. For example, in Example II if there is another directorial goal that requires the wolf to eat the cake, a conflict exists between the two goals: the wolf can not eat the cake if it is killed. Currently, when conflicts happen, an implicit priority is assumed based on the order the author lays out the goals. Finally, directorial control does not always succeed. Whether a set of directorial goals can be achieved depends on the algorithms used by the director agent, the model of the characters, the type of user who is interacting with the system and even the initial state of the characters and the user. We plan to conduct an evaluation on the effectiveness of the directorial control we deployed in the Little Red Riding story.

7 Conclusion

In this work, we present a unique approach for realizing directorial control in interactive narratives based on the model of the user. This approach is implemented within the Thespian framework for interactive narratives. Thespian's director agent monitors the progress of the story and adjusts the virtual characters' behaviors and beliefs if necessary to achieve directorial goals. The explicit modeling of the user enables our approach to avoid accidentally creating inconsistent characters during directorial control, and to better predict the user's experience. For example, our approach allows the author to specify directorial goals regarding the characters' states and beliefs, so that the directorial control can directly target the user's cognitive and affective experience. The algorithms used by the director agent are presented in this paper, followed by examples of applying directorial control in the Little Red Riding Hood story.

References

1. Marsella, S.C., Johnson, W.L., Labore, C.: Interactive pedagogical drama for health interventions. In: AIED. (2003)
2. Paiva, A., Dias, J., Sobral, D., Aylett, R., Sobreperez, P., Woods, S., Zoll, C.: Caring for agents and agents that care: Building empathic relations with synthetic agents. In: AAMAS. (2004) 194–201

3. Si, M., Marsella, S.C., Pynadath, D.V.: Thespian: An architecture for interactive pedagogical drama. In: AIED. (2005)
4. Kelso, M.T., Weyhrauch, P., Bates, J.: Dramatic presence. Presence: Teleoperators and Virtual Environments **2** (1993)
5. Riedl, M.O., Saretto, C.J., Young, R.M.: Managing interaction between users and agents in a multi-agent storytelling environment. In: AAMAS. (2003) 741–748
6. Cavazza, M., Charles, F., Mead, S.J.: Agents' interaction in virtual storytelling. In: Proceedings of the International WorkShop on Intelligent Virtual Agents. (2001) 156–170
7. Magerko, B., Laird, J.E.: Mediating the tension between plot and interaction. (In: AAAI Workshop Series: Challenges in Game Artificial Intelligence.) 108–112
8. Szilas, N.: IDtension: a narrative engine for interactive drama. In: the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment, Darmstadt Germany (2003)
9. Braun, N.: Storytelling in collaborative augmented reality environments. In: Proceedings of the 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision. (2003)
10. Mateas, M., Stern, A.: Integrating plot, character and natural language processing in the interactive drama Façade. In: the International Conference on Technologies for Interactive Digital Storytelling and Entertainment. (2003)
11. Murray, J.H.: Hamlet on the Holodeck - The Future of narrative in Cyberspace. MIT Press, Cambridge (1997)
12. Bates, J.: Virtual reality, art, and entertainment. Presence: Teleoperators and Virtual Environments. **2**(1) (1992) 133–138
13. Weyhrauch, P.: Guiding Interactive Drama. PhD thesis, Carnegie Mellon University (1997) Technical Report CMU-CS-97-109.
14. Lamstein, A., Mateas, M.: A search-based drama manager. In: the AAAI-04 Workshop on Challenges in Game AI. (2004)
15. Onega, S., Landa, J.A.G.: Narratology: An Introduction. Longman, London and New York (1996)
16. Bruner, J.: Acts of Meaning. Cambridge, Mass : Harvard University Press. (1990)
17. Ochs, E., Capps, L.: Living narrative. Creating lives in everyday storytelling. Cambridge, MA, Harvard University Press (2001)
18. Si, M., Marsella, S.C., Pynadath, D.V.: Proactive authoring for interactive drama: An author's assistant. In: IVA, Paris, France (2007)
19. Si, M., Marsella, S.C., Pynadath, D.V.: Thespian: Using multi-agent fitting to craft interactive drama. In: AAMAS. (2005) 21–28
20. Si, M., Marsella, S.C., Pynadath, D.V.: Modeling appraisal in theory of mind reasoning. JAAMAS. (2009)
21. Si, M., Marsella, S.C., Pynadath, D.V.: Thespian: Modeling socially normative behavior in a decision-theoretic framework. In: IVA. (2006)
22. Marsella, S.C., Pynadath, D.V., Read, S.J.: PsychSim: Agent-based modeling of social interactions and influence. In: Proceedings of the International Conference on Cognitive Modeling. (2004) 243–248
23. Smallwood, R.D., Sondik, E.J.: The optimal control of partially observable Markov processes over a finite horizon. Operations Research **21** (1973) 1071–1088
24. Gmytrasiewicz, P., Durfee, E.: A rigorous, operational formalization of recursive modeling. In: ICMAS. (1995) 125–132
25. Pynadath, D.V., Marsella, S.C.: Fitting and compilation of multiagent models through piecewise linear functions. In: AAMAS. (2004) 1197–1204